

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Система визначення оптимального маршруту у
вебзастосунку визначних місць «Vikha»»**

Виконала студентка групи ЗІПЗм-22-1
ПЕРЕВАЛОВА Анастасія Дмитрівна

Керівник роботи:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
ЛЕВКІВСЬКИЙ Віталій Леонідович

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«18» вересня 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ПЕРЕВАЛОВА Анастасія Дмитрівна**

Керівник роботи: **ЧИЖМОТРЕ Олексій Володимирович**

Тема роботи: **«Система визначення оптимального маршруту у вебзастосунку визначних місць «Vikha»»,**

затверджена Наказом закладу вищої освіти від **«18» вересня 2023 р. № 481/с**

Вихідні дані для роботи: **предметом дослідження є поєднання модифікованих алгоритмів на графах, як інструменту для визначення оптимальності маршруту. Об'єктом дослідження є процес прокладання оптимального маршруту між визначними місцями.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	05.04.2023р.	05.04.2023р.
2	Чижмотря О.В.	20.09.2023р.	20.09.2023р.
3	Чижмотря О.В.	08.11.2023р.	08.11.2023р.

РЕФЕРАТ

Індивідуальне завдання кваліфікаційної магістерської роботи полягає в науковому дослідженні методів визначення та прокладання оптимального маршруту між місцями на мапі, які будуть використані для подальшої розробки системи зі створення користувачами путівників з автовідтворенням аудіо опису визначних місць.

Кваліфікаційна магістерська робота на тему «Система визначення оптимального маршруту у вебзастосунку визначних місць «Vikha»» складається з теоретичного та програмного комплексу системи та пояснювальної записки. Пояснювальна записка до кваліфікаційної магістерської роботи складається з 31 найменування, 5 додатків, 10 таблиць та 34 ілюстрацій по тексту. Повний обсяг роботи становить 85 сторінок, з них 76 сторінок основного тексту.

КЛЮЧОВІ СЛОВА: ГАМІЛЬТОНІВ ШЛЯХ, ГРАФ, МАРШРУТ, ВЕБЗАСТОСУНОК, МАПА, API.

ABSTRACT

The individual task of the master's qualification work involves a scientific investigation of methods for determining and planning the optimal route between locations on a map. This research will be utilized for the further development of a system for creating user guides with automatic playback of audio descriptions of landmarks.

The master's qualification work on the topic «System for determining the optimal route in a web application for landmarks «Vikha»» consists of a theoretical and software complex of the system and an explanatory note. The explanatory note for the master's qualification work includes 31 headings, 5 appendices, 10 tables, and 34 illustrations in the text. The total volume of the work is 85 pages, with 76 pages comprising the main text.

KEYWORDS: HAMILTONIAN PATH, GRAPH, ROUTE, WEB APPLICATION, MAP, API.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1. МОДЕЛІ ТА МЕТОДИ ВИЗНАЧЕННЯ ОПТИМАЛЬНИХ МАРШРУТІВ	9
1.1. Аналіз методів пошуку оптимального маршруту	9
1.2. Аналіз аналогів системи.....	17
1.3. Постановка задачі дослідження.....	25
Висновки до розділу 1	28
РОЗДІЛ 2. РОЗРОБКА СИСТЕМИ ВИЗНАЧЕННЯ ОПТИМАЛЬНОГО МАРШРУТУ	29
2.1. Визначення архітектурних рішень системи.....	29
2.2. Розробка бази даних системи	34
2.3. Розробка алгоритму створення путівника з прокладанням оптимального маршруту.....	37
2.4. Реалізація відображення та передачі даних	45
Висновки до розділу 2	54
РОЗДІЛ 3. ПОРЯДОК РОБОТИ, ІНТЕРФЕЙС ТА ТЕСТУВАННЯ СИСТЕМИ...	55
3.1. Порядок встановлення та налаштування системи.....	55
3.2. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	56
3.3. Опис інтерфейсу системи.....	62
3.4. Тестування роботи системи	69
Висновки до розділу 3	71
ВИСНОВКИ.....	72
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	74
ДОДАТКИ.....	77

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- API – Application Programming Interface (Прикладний програмний інтерфейс).
- CRP – Customizable Route Planning.
- CRUD – create, read, update, delete (створення, читання, оновлення, вилучення).
- CSS – Cascading Style Sheets (Каскадні таблиці стилів).
- DAG – directed acyclic graph (Орієнтований ациклічний граф)
- GDAL – Geospatial Data Abstraction Library.
- GIS – Geographic information system (Геоінформаційна система).
- HTML – HyperText Markup Language, стандартизована мова розмітки документів в мережі.
- HTTP – HyperText Transfer Protocol (Протокол передачі гіпертекстових документів).
- JSON – JavaScript Object Notation, текстовий формат даних, заснований на JavaScript.
- MTV – Model, Template, View (Модель, Шаблон, Представлення).
- MVP – Model-View-Presenter.
- MVVM – Model-View-ViewModel, шаблон проєктування архітектури додатку.
- NP – Non-deterministic Polynomial time (Недетермінований поліноміальний час).
- ORM – Object-Relational Mapping (Об'єктно-реляційне відображення).
- OSM – OpenStreetMap.
- OSRM – Open Source Routing Machine.
- REST – Representational State Transfer (передача репрезентативного стану).
- TTS – Text-to-Speech (Перетворення тексту в мовлення).
- URL – Uniform Resource Locator (Уніфікований локатор ресурсів).
- БД – База даних.
- ООП – об'єктно орієнтоване програмування, методологія програмування, заснована на представленні програми у вигляді сукупності об'єктів.
- ОС – операційна система.

ВСТУП

Актуальність теми. Актуальність обраної теми визначається потребою вдосконалення сучасних підходів у розробці вебзастосунків для прокладання маршрутів. Зокрема, основна увага приділяється розробці системи для автоматичного прокладання оптимального маршруту між визначними місцями у вебзастосунку «Vikha». Актуальність полягає в розв’язанні проблеми ефективного використання геоінформації та наданні користувачам можливості самостійно створювати та оптимізовувати маршрути.

Зазначена тема актуальна в контексті сучасних технологій та практичних потреб користувачів. Розробка вебзастосунку, який дозволяє створювати персоналізовані маршрути для відвідування визначних місць, має великий практичний потенціал для мандрівників та тих, хто цікавиться історією та культурою різних регіонів.

Вузьконаправлений та спеціалізований підхід до розробки системи надає їй конкурентну перевагу у порівнянні з аналогами. Малий конкурентний простір та вузька спрямованість дозволяють зосередитися на конкретних потребах користувачів, забезпечуючи при цьому легку та зручну навігацію без зайвих компонентів. Такий фокус на конкретній задачі робить «Vikha» унікальним та важливим інструментом для тих, хто цінує ефективність та простоту в плані планування путівників.

Мета і завдання дослідження. Метою дослідження є впровадження функціоналу та інтерфейсу для автоматичного прокладання оптимального маршруту між визначними місцями у вебзастосунку «Vikha».

Встановлена мета обумовлює необхідність виконання наступних **завдань**:

- вибрати систему для отримання геоінформації;
- формалізація та структурування вхідних даних;
- розробка алгоритму пошуку оптимального маршруту;
- реалізація алгоритму пошуку оптимального маршруту;
- розробка механізму отримання вхідних даних та передачі вихідних даних;

— розробка механізму відображення маршруту на мапі.

Об’єктом дослідження є процес прокладання оптимального маршруту між визначними місцями.

Предметом дослідження є поєднання модифікованих алгоритмів на графах, як інструменту для визначення оптимальності цього маршруту.

Методи дослідження. У роботі було використано результати теорії графів, методи об’єктно-орієнтованого проєктування, об’єктно-орієнтованого програмування та вебпрограмування.

Інформаційна база дослідження. Інформаційну базу дослідження складають інформаційні ресурси спільного користування та інформаційні ресурси науково-технічної інформації, що розміщені в мережі інтернет.

Наукова новизна. У даній роботі запропоновано підхід до прокладання оптимізованого маршруту між визначними місцями у форматі вебзастосунку. Суть даного підходу полягає у спільному використанні модифікованого алгоритму Дейкстри та вирішенні задачі Гамільтонового шляху. Спільне використання цих алгоритмів застосовано в програмному продукті, де оптимальний маршрут розраховується між списком визначних місць безпосередньо сформованим користувачем.

Практичне значення одержаних результатів. Практичне значення роботи полягає у можливості самостійного створення путівника користувачем, оптимального проходження всіх визначних місць з автоматичним аудіовідтворенням інформації про об’єкт та у виборі методу оптимізації шляху.

Апробація. За темою кваліфікаційної магістерської роботи було опубліковано 2 тез конференцій [1, 2].

РОЗДІЛ 1. МОДЕЛІ ТА МЕТОДИ ВИЗНАЧЕННЯ ОПТИМАЛЬНИХ МАРШРУТІВ

1.1. Аналіз методів пошуку оптимального маршруту

Використання методів визначення найоптимальніших маршрутів є важливою складовою для багатьох вебзастосунків та сервісів, особливо для тих, які пов'язані з навігацією та подорожами. Методи цієї області відіграють вирішальну роль у побудові шляхів для користувачів, які хочуть досягти своїх місць призначення якнайшвидше та з найменшими зусиллями.

Теорія графів, як галузь інформатики та математики, виникла під впливом задачі щодо Кенігсберських мостів, яку успішно вирішив Леонард Ойлер у 1736 році. З появою обчислювальної техніки з'явилися нові завдання, але водночас відкриваються й нові методи для їх вирішення. При вивченні різноманітних джерел можна виявити різноманітні тлумачення, тому в даному розділі ми розглянемо ті, що лежать в основі цієї роботи. Джерелом для нас буде посібник Кормена [3], хоча з невеликими коригуваннями.

Орієнтований граф, відомий також як оргграф (англ. directed graph), є структурою, яка складається з множини вершин V , з'єднаних між собою ребрами з множини E , де ребра вказують напрямок руху по графу. Рух можливий лише у визначеному напрямку. Формально, орієнтований граф є впорядкованою парою $G = (V, E)$. Ребро – це впорядкована пара двох вершин (v, w) , де вершина v є початковою вершиною, а вершина w – кінцевою вершиною (рис. 1.1). Вершина w є сусідньою вершиною відносно вершини v , якщо в графі G існує ребро (v, w) . Такий зв'язок називається відношенням суміжності.



Рис. 1.1 - Дві вершини з напрямленим ребром

Стрілка, яка представляє ребро, направлена від початкової вершини до кінцевої вершини. Зазвичай вершини використовуються для зображення об'єктів, а ребра між ними відображають взаємозв'язок між цими об'єктами. Наприклад,

об'єктами можуть бути міста, а ребрами – автобусні маршрути між цими містами, як зображено на рисунку 1.2.

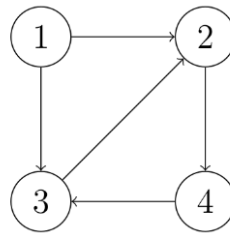


Рис. 1.2 - Орієнтований граф

Степенем виходу $\text{outdeg}(v)$ називається кількість ребер, які виходять із вершини v . Степенем входу $\text{indeg}(v)$ називається кількість ребер, які входять до вершини v . У нашому випадку, ми маємо отримати граф, який матиме степінь 1 виходу та степінь 1 входу для кожної вершини, окрім початкової та кінцевої.

Неорієнтований граф $G = (V, E)$ – це структура, що складається зі скінченного набору вершин V та набору ребер E , де напрямок зв'язків між вершинами не має значення, важливо лише існування зв'язку. Ребро графа без напрямку містить пару двох вершин $\{v, w\}$. У графі без напрямку відношення між сусідами є симетричним. Кожне ребро може містити точно дві різні вершини, оскільки петлі заборонені. Степінь вершини $\text{deg}(v)$ у графі без напрямку – це кількість ребер, що з'єднані з вершиною v .

Джерелом називається вершина з нульовим напівстепенем заходу, а вершина з нульовим степенем виходу називається стоком [4].

Шляхом від s до t у графі $G = (V, E)$ називається послідовність вершин $(v_0, v_1, v_2, \dots, v_n)$, де $v_0 = s$, $v_n = t$, і (v_{i-1}, v_i) ($i = 1, \dots, n$) є ребрами з E . Довжина шляху дорівнює n . Шлях, що складається з однієї вершини, має довжину нуль. Якщо існує шлях від s до t , то ми кажемо, що t є досяжним з s . Простий шлях – це шлях, у якому всі вершини є різними.

Цикл – це шлях, у якому перша і остання вершина є однаковими, $v_0 = v_n$. Простий цикл – це цикл, у якому всі вершини різні, за винятком останньої. Граф, що не містить простих циклів, називається ациклічним. Орієнтований ациклічний граф називається DAG (англ. directed acyclic graph).

Зважений граф (англ. weighted graph) – це граф, у якому кожне ребро має свою вагу. Ваги представляють собою числові значення, які можуть відображати відстань, пропускну здатність каналу, вартість тощо. Вагу ребра (s, t) або $\{s, t\}$ позначаємо як $w(s, t)$. Вагою шляху/циклу/дерева в зваженому графі називаємо суму ваг відповідних ребер.

Неорієнтовний граф є зв'язним (англ. connected), якщо кожна вершина досяжна з усіх інших вершин. Орієнтований граф є сильно зв'язним (англ. strongly connected), якщо будь-які дві вершини досяжні одна з іншою.

Матриця суміжності – це квадратна матриця, яка використовується для представлення зв'язків між вершинами у графі. Матриця суміжності для графу G зі скінченною кількістю вершин n (пронумерованих числами від 1 до n) є квадратною матрицею A розміром n , де кожний елемент a_{ij} відображає кількість ребер, які з'єднують i -ту вершину графу з j -тою вершиною.

Поліноміальний час – це поняття в теорії обчислювальної складності, що відноситься до складності алгоритму в залежності від розміру вхідних даних. Якщо алгоритм вирішує задачу за поліноміальний час, то означає, що існує поліном, такий що тривалість виконання алгоритму не перевищує значення цього полінома при заданому розмірі вхідних даних.

NP (від англ. Non-deterministic Polynomial time) – це клас задач, для яких можна швидко (тобто за поліноміальний час) недетермінованими алгоритмами перевірити правильність рішення, якщо воно дано.

NP-складні задачі – це ті задачі, для яких будь-яка задача з класу NP може бути зведена до даної задачі за поліноміальний час. Іншими словами, якщо було б знайдено поліноміальний алгоритм вирішення хоча б однієї задачі з цієї категорії, тоді всі NP-задачі могли бути вирішені за поліноміальний час.

Таким чином, NP-складність є позначенням для задач, які є як мінімум так само складними, як і найскладніші задачі в NP. Якщо задача є NP-складною і водночас належить до класу NP, вона відноситься до класу NP-повних задач (рис. 1.3). Таким чином, NP-повні задачі утворюють підмножину «найскладніших» задач у класі NP.

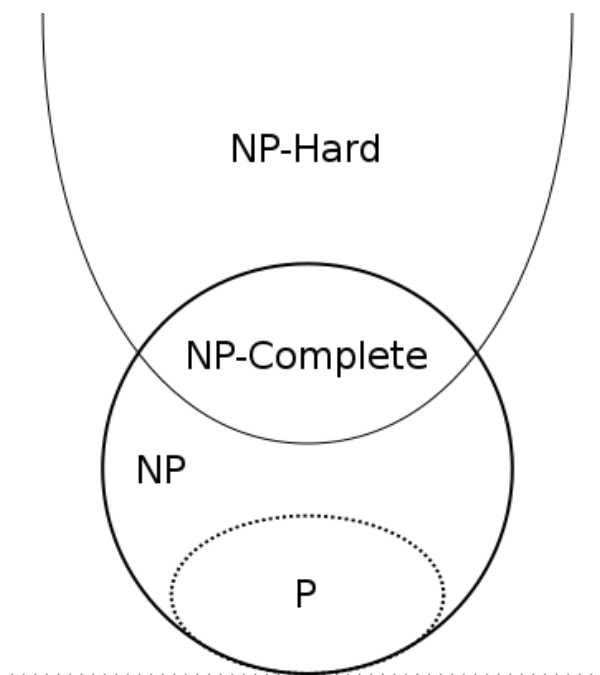


Рис. 1.3 - Діаграма класів складності

Евристичні алгоритми – це методи прийняття рішень, що використовуються при вирішенні складних задач, для яких точні алгоритми або не існують, або займають надмірно великий обсяг часу або ресурсів. Вони розроблені з метою наближеного вирішення задачі за рахунок точності.

Вирішення завдання пошуку маршруту на карті можна представити у вигляді орієнтованого графа, як показано на рисунку 1.4. У цьому графі вершини відображають об'єкти на карті, такі як музеї, пам'ятники та інші, а ребра відповідають дорогам та вулицям, які з'єднують ці об'єкти. Кожне ребро має свою вагу, яка може визначати відстань або час, необхідний для подолання цієї відстані.

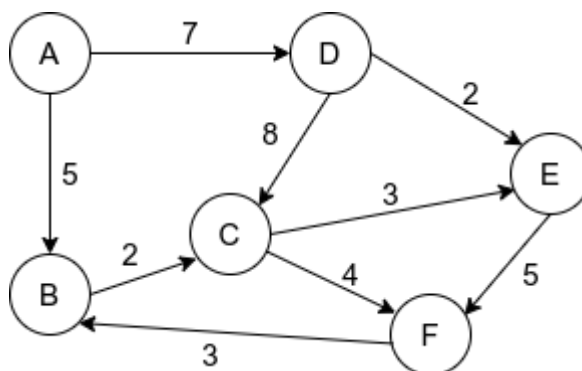


Рис. 1.4 - Орієнтований зважений граф

Матричне представлення для графа (див. рис. 1.4) виглядає наступним чином:

$$M = \begin{pmatrix} 0 & 5 & 0 & 7 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 4 \\ 0 & 0 & 8 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 3 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Отже, ця проблема полягає у визначенні найкоротшого або найшвидшого, та найбільш оптимального маршруту на графі. Пошук найкоротшого маршруту ґрунтується на виявленні шляху між заданими вершинами графа так, щоб сума ваг ребер була якомога меншою.

Дорожні мережі є орієнтованими, позитивно зваженими графами $G = (V, E, \text{len})$, де V – це множина вершин, що представляють перехрестя або точки на дорозі, $E \subseteq V \times V$ – множина ребер, що представляють сегменти доріг між початковою вершиною u та цільовою вершиною v . Функція довжини $\text{len} : E \rightarrow \mathbb{R}_{\geq 0}$, також відома як метрика G , призначає позитивну вагу ребру для всіх ребер $(u, v) \in E$. Прикладами метрик у дорожніх мережах є відстань або час подорожі. Граф G вважається направленим, оскільки ребра $(u, v) \in E$ є упорядкованими кортежами. Граф G називається ненаправленим, якщо для кожного ребра обернене ребро (v, u) також належить E , і $\text{len}(u, v) = \text{len}(v, u)$. Для направленого зваженого графа $G = (V, E, \text{len})$ відповідний обернений граф – це $G^R = (V, E^R = \{(u, v) | (v, u) \in E\}, \text{len}^R(u, v) = \text{len}(v, u))$.

Шляхи в графі – це упорядковані списки вершин $p = (v_1, v_2, \dots, v_n)$, де $v_1, \dots, v_n \in V$, $(v_i, v_{i+1}) \in E$. Довжина шляху p визначається як $\text{len}(p) = \sum_{i=1}^{n-1} \text{len}(v_i, v_{i+1})$, сума ваг ребер між вершинами у шляху. Якщо для $s, t \in V$, $p = (s, \dots, t)$ є s - t -шляхом, і для будь-якого іншого $p' = (s, \dots, t)$ довжина p менша або рівна довжині p' , то p є найкоротшим шляхом. Довжину найкоротших s - t -шляхів позначається як відстань $\text{dist}(s, t)$. Хоча найкоротший шлях не обов'язково єдиний, достатньо знайти один шлях для вирішення проблеми найкоротших шляхів від точки до точки [5].

Алгоритм Дейкстри був вперше представлений у 1959 році Едсгером В. Дейкстрою [6] для обчислення найкоротших шляхів у графах з позитивною вагою ребер. Алгоритм проскановує всі вершини $u \in V$ в порядку їх відстані $\text{dist}(s, u)$ для

заданої початкової вершини s . Він є основою багатьох технік прискорення для задачі найкоротших шляхів від точки до точки.

Алгоритм Дейкстри працює так:

1. Він знаходить найкоротший шлях від джерела до найближчої вершини, потім до наступної найближчої вершини і т. д.
2. На кожному i -му кроці алгоритм вже визначив найкоротший шлях до $i-1$ інших найближчих вершин. Ці вершини, разом з джерелом і ребрами найкоротших шляхів до них, формують піддерево T_i в заданому графі.
3. Найближчій до джерела вершини шукаються серед вершин, які є сусідніми до вершин у T_i . Вони називаються «граничними вершинами».
4. Усі інші вершини можна розглядати як граничні, що підключені до вершин дерева ребрами з нескінченно великими вагами.
5. Алгоритм вираховує суму відстані від кожної граничної вершини u до найближчої вершини дерева v і довжину шляху від джерела до v . Гранична вершина із найменшою такою сумою стає наступною найближчою вершиною [7].

Customizable Route Planning (CRP) – це спеціалізований метод обчислення найкоротших шляхів в дорожніх мережах, який оптимізований для швидкого налаштування за допомогою різних метрик.

Багаторівневий поділ – це стратегія організації даних у графі, яка використовується в CRP. Вона включає створення впорядкованої сім'ї поділів з L рівнів. Поділ означає, що ми ділимо вершини V графа G на роз'єднані комірки C_i так, що об'єднання всіх цих комірок становить V .

У багаторівневому поділі, кожен рівень містить свій поділ вершин. Цей поділ є вкладеним, що означає, що вершини, які є в одній і тій же комірці на певному рівні, повинні бути в одній і тій же комірці на всіх вищих рівнях.

Для кожної комірки на рівні, крім останнього, існує «надкомірка» на наступному рівні, котра є єднанням її «підкомірок».

Багаторівневий алгоритм Дейкстри (MLD) є модифікацією алгоритму Дейкстри, яка враховує багаторівневий розподіл та використовує попередньо

обчислені комірки накладання для евристичного прискорення обчислень. Перехід до накладення зводиться до питання, які ребра вершини слід розглядати для відпущення. Залежно від початкової та цільової вершини s, t кожна вершина $u \in V$ має визначений рівень. Це найвищий рівень, на якому вона не має спільної клітини ані з s , ані з t .

Задача комівояжера є однією з найважливіших та найбільш досліджуваних проблем в галузі комбінаторної оптимізації та транспортної логістики. Ця задача має безліч застосувань у реальних сценаріях, включаючи маршрутизацію транспорту, планування робочих візитів та організацію роботи флотів транспортних засобів.

Евристичні алгоритми для задачі комівояжера не надають гарантії знаходження оптимального рішення. Отримане рішення є приблизним, але час його знаходження досить короткий. Зазвичай евристичні алгоритми використовуються тоді, коли не відомі точні алгоритми, що працюють достатньо швидко.

У своєму базовому вигляді задача комівояжера полягає у пошуку найкоротшого можливого маршруту, який проходить через кожне місто рівно один раз і завершується у вихідному місті. Ця задача є NP-складною, що робить її складною для вирішення у випадку великої кількості міст.

Одним із підходів до розв'язання задачі комівояжера є використання алгоритмів генетичного пошуку. Генетичні алгоритми моделюють еволюційний процес шляхом використання понять генетики, таких як селекція, схрещування та мутація. Ці алгоритми можуть ефективно працювати для вирішення задачі комівояжера, шукаючи оптимальні рішення у просторі можливих маршрутів.

Інший підхід включає використання динамічного програмування та методів гілок та меж. Динамічне програмування дозволяє ефективно знаходити оптимальні рішення, розглядаючи проміжні підзадачі та забезпечуючи оптимальність локальних виборів. Метод гілок та меж також розкладає задачу на менші підзадачі, але здатний відсікати надто обширні гілки пошуку, зменшуючи тим самим обсяг обчислень.

Задача комівояжера стає ще складнішою, коли враховуються обмеження часу, ваги ребер чи інші умови. У таких випадках необхідно розглядати різні модифікації алгоритмів та враховувати більше факторів при визначенні оптимального маршруту.

Застосування задачі комівояжера в реальних областях включає планування маршрутів для доставки товарів, визначення послідовності операцій для обслуговування об'єктів та оптимізацію маршрутів для поліцейських чи екстрених служб. Ефективне розв'язання цієї задачі має велике значення для вдосконалення логістичних та транспортних процесів у різних галузях.

Гамільтонів шлях є важливою концепцією в графовій теорії та комбінаторній математиці. Цей математичний об'єкт виник у контексті задачі пошуку шляху, який пройде через кожен вузол графа рівно один раз.

У математичному термінологічному визначенні, Гамільтонів шлях – це такий шлях у графі, який проходить через кожен вузол графа рівно один раз [8]. Цей шлях може бути або може не бути циклом, тобто завершуватися в початковому вузлі. Якщо такий цикл існує, то його називають Гамільтоновим циклом. Гамільтонів граф – це граф, який містить цикл Гамільтона. Півгамільтонів граф – це граф, який містить шлях Гамільтона. Проблема знаходження шляху та циклу Гамільтона є NP-повною.

Якщо для кожної пари вершин графу G існує шлях Гамільтона, що з'єднує ці вершини, то граф G називається гамільтоново зв'язним. Прикладами є повні графи K_3 , K_4 , K_5 .

Якщо граф має цикл Гамільтона, то він також має шлях Гамільтона [9]. Достатньо видалити одне ребро з циклу.

Задача про існування Гамільтонового шляху в графі є важкою проблемою. У класичних умовах вона може бути NP-повною, що означає, що немає відомого ефективного методу для знаходження Гамільтонового шляху в загальному випадку. Багато досліджень спрямовані на знаходження умов, за яких такий шлях існує, або розробка евристичних методів для його швидшого знаходження у певних класах графів.

Однією з важливих характеристик Гамільтонового шляху є його застосування в комівояжерській проблемі, де метою є знайти найкоротший маршрут, який відвідує кожне місто лише один раз і завершується в початковому місті. Випадковим чином, задача комівояжера може бути розглянута як частковий випадок Гамільтонівського шляху.

Дослідження в області Гамільтонових шляхів також спрямовані на вивчення їх застосувань у різних галузях, таких як мережеве проєктування, транспортна логістика та оптимізація. Математики та інформатики розвивають нові методи для вирішення цих завдань, аби забезпечити оптимальні рішення в реальних сценаріях.

Таким чином ми визначили, що для вирішення проблеми пошуку найшвидшого шляху між усіма визначними місцями, з початковою точкою і відсутністю повернень у неї, використання Гамільтонового шляху є особливо обґрунтованим. Гамільтонів шлях визначає такий маршрут в графі, який проходить через кожен вузол рівно один раз, не утворюючи циклу. У нашому випадку, визначні місця можуть бути розглянуті як вершини графа, а шлях, що пов'язує їх у Гамільтонів шлях, відображає оптимальний маршрут. Враховуючи, що Гамільтонів шлях не передбачає повторення вершин, його застосування дозволяє ефективно знаходити найшвидший та оптимальний маршрут між визначними місцями, враховуючи їхню унікальність та відсутність необхідності у поверненні до початкової точки.

Загалом, використання методів визначення найоптимальніших маршрутів є ключовим елементом для створення вебзастосунків та сервісів, які надають користувачам можливість знаходити оптимальні маршрути для подорожей та навігації. Вони допомагають користувачам ефективно планувати свої подорожі та заощаджувати час і зусилля на переміщеннях.

1.2. Аналіз аналогів системи

Протягом останніх років спостерігається стрімкий ріст популярності цифрових карт та використання інструментів цифрового картографування. Хоча Google залишається провідним у цьому сегменті, спостерігається збільшення

кількості компаній, що діють у сфері картографічних технологій. Величезна зацікавленість цифровими картами не є випадковою; вони можуть виявитися корисними, інформативними, естетично приємними та можуть полегшувати користування продуктом.

Сьогодні майже 2 мільйони вебсайтів у всьому світі використовують API Карт Google; Google Maps займає 61,73% ринку; тільки в США понад 660 000 вебсайтів мають вбудовані карти Google, а Google Місця на карті можна знайти на 146 125 вебсайтах [10].

Зараз можна знайти необмежену кількість красиво оформлених інтерактивних та статичних цифрових карт. Але варто зазначити, що після проведення пошуку, продуктів з вище вказаними функціями, не було виявлено.

Далі розглянемо подібні існуючі рішення.

Google Карти – це вебкартографічна платформа та споживчий додаток, запропонований Google (рис. 1.5). Вони пропонують супутникові зображення, карти вулиць, панорамні види вулиць на 360° (перегляд вулиць), умови дорожнього руху в режимі реального часу (Google Traffic) і планування маршрутів для подорожей пішки, автомобілем, велосипедом (у бета-версії) або громадським транспортом [11].

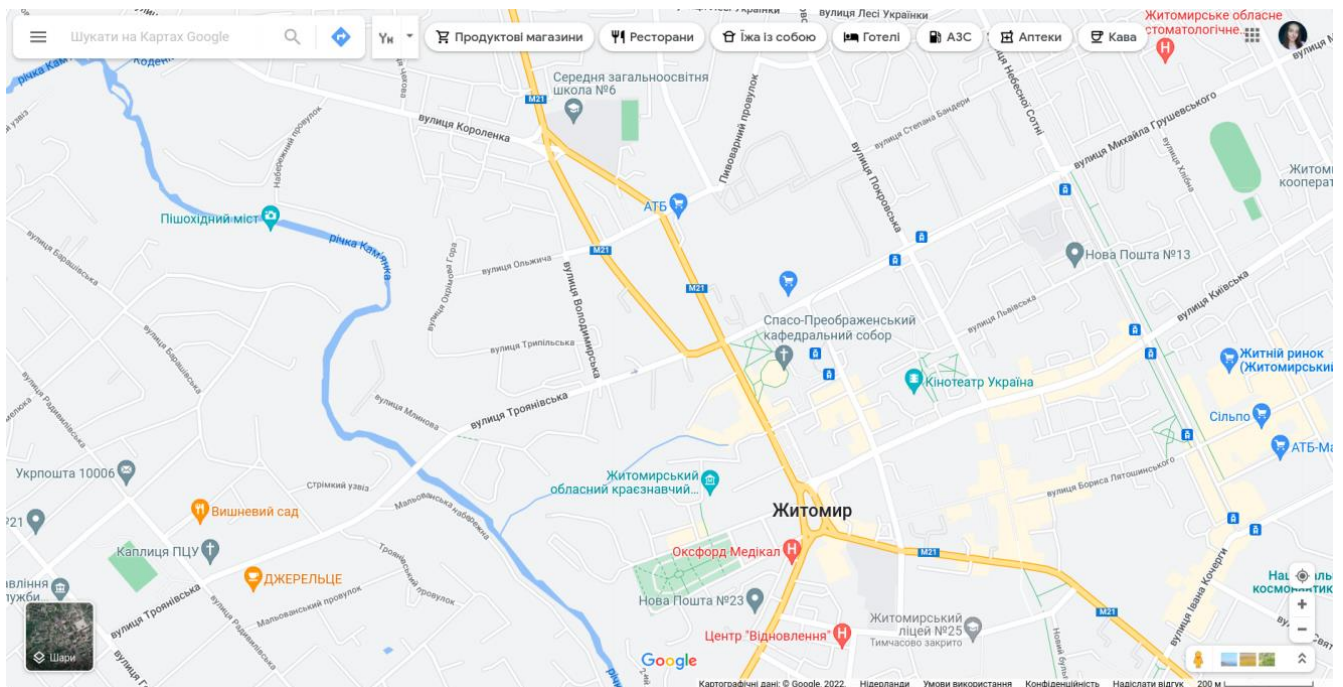


Рис. 1.5 - Google Maps

Коли карта масштабується або змінюється зона видимості, то квадрати сітки завантажуються з сервера та вставляються на сторінку. Коли користувач шукає компанію, результати завантажуються у фоновому режимі для вставки на бічну панель і карту; сторінка не перезавантажується. Використовується прихований `iframe` з поданням форми, оскільки він зберігає історію браузера. Як і багато інших вебпрограм Google, Google Maps широко використовує JavaScript. Сайт також використовує JSON для передачі даних, а не XML, з міркувань продуктивності. Google Карти є гарним сервісом, але містить рекламу та забагато елементів на карті.

Зокрема Google Карти також мають інформацію про місця, але іноді їх опис недостатній, або узагалі відсутній. Якщо говорити про відтворення інформації в аудіоформаті, то такий функціонал відсутній. А прокладання маршруту зі списку вибраних місць не є можливим, потрібно буде відзначати бажані точки кожного разу.

izi.Travel – відкрита і безкоштовна платформа для розповіді людей, щоб публікувати історії, і додаток для смартфонів, щоб споживачі могли почути їх (рис. 1.6) [12]. Загалом це аудіопутівники створені іншими людьми, де завчасно задані точки маршруту. Мінус цього сервісу в тому, що немає можливості шукати об'єкти окремо, не всі користувачі надають достатню кількість інформації про місце.

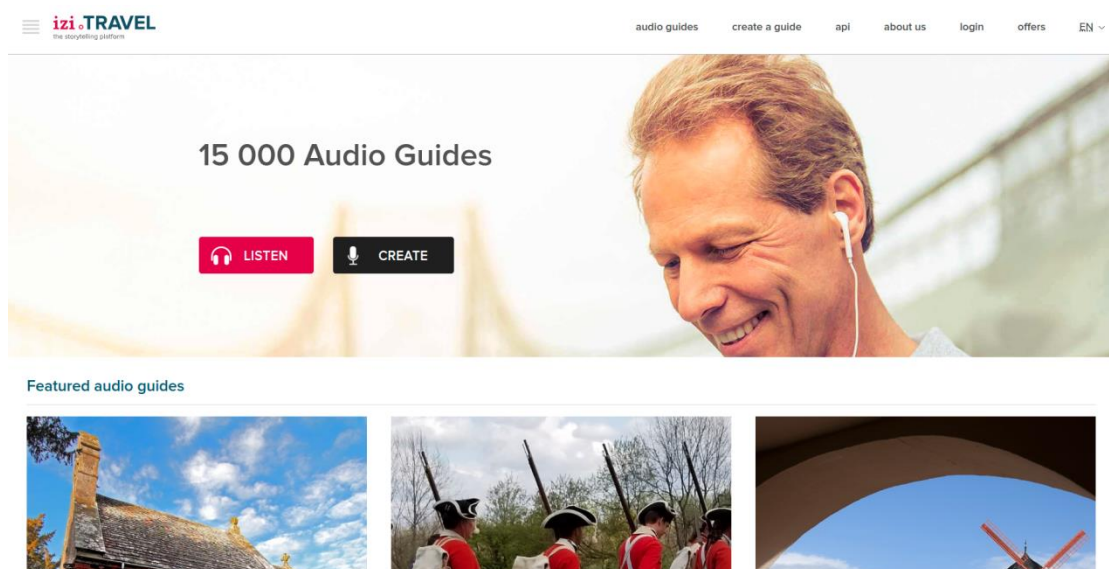


Рис. 1.6 - izi.TRAVEL

AllTrails – це популярний мобільний додаток та вебсайт, який спеціалізується на зборі та наданні інформації про маршрути для активного відпочинку на природі, такі як походи, велосипедні маршрути, біг і багато іншого (рис. 1.7). Він надає користувачам доступ до великої бази даних маршрутів по всьому світу.

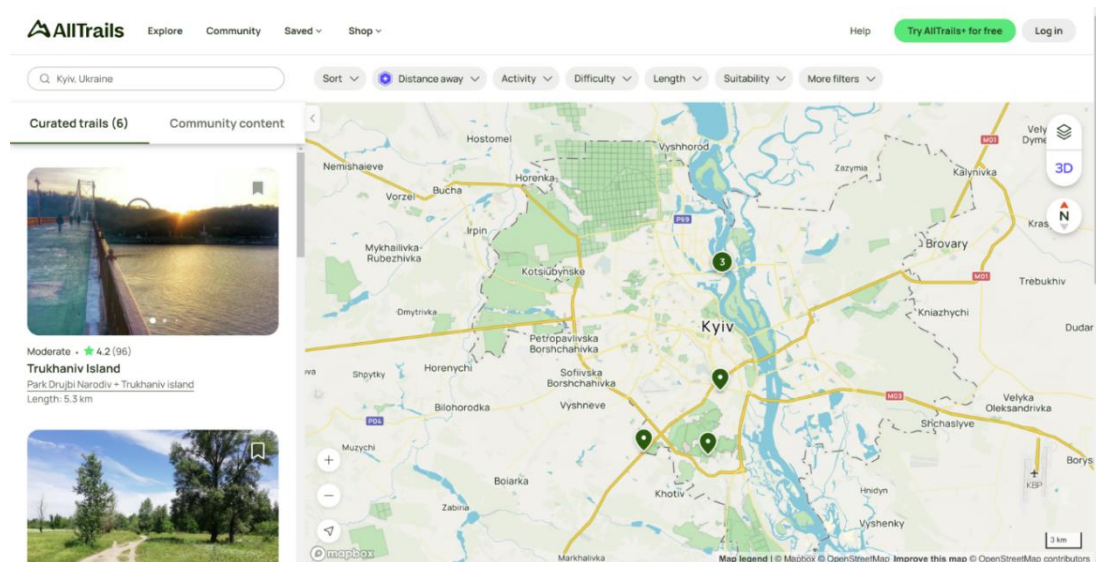


Рис. 1.7 - AllTrails

Сервіс дозволяє користувачам отримувати доступ до бази даних карт маршрутів, яка включає краудсорсингові відгуки та зображення. Залежно від статусу підписки користувача, ці ресурси можна використовувати онлайн і офлайн. Основні риси AllTrails включають:

Каталог маршрутів. Додаток містить багато тисяч маршрутів, які можна переглядати за типом активності, рейтингом, рівнем складності та місцем розташування. На поточний момент було успішно відібрано більше ніж 400 тисяч маршрутів [13].

Інформація про місце. Деякі маршрути включають докладну інформацію про визначні місця та природні атракції вздовж маршруту.

Спільнота користувачів. AllTrails має активну спільноту користувачів, які додають свої відгуки, фотографії та рейтинги маршрутів. На даний момент AllTrails налічує 60 мільйонів дослідників.

Збережені маршрути та подорожі. Користувачі можуть робити закладки та створювати подорожі, об'єднуючи декілька маршрутів в одну подорож.

Незважаючи на ці переваги, є деякі недоліки цього застосунку. По-перше, AllTrails спеціалізується переважно на активних відпочинкових маршрутах, таких як походи і велосипедні подорожі, що може бути недоліком для тих, хто шукає інформацію про інші види подорожей. Крім того, деякі додаткові функції і карти доступні лише для користувачів з платним підпискою, що може бути обмеженням для безкоштовних користувачів.

Komoot – це ще один додаток та платформа для створення та відстеження маршрутів, спрямованих на активний відпочинок, але з основним фокусом на велосипедних подорожах (рис. 1.8).

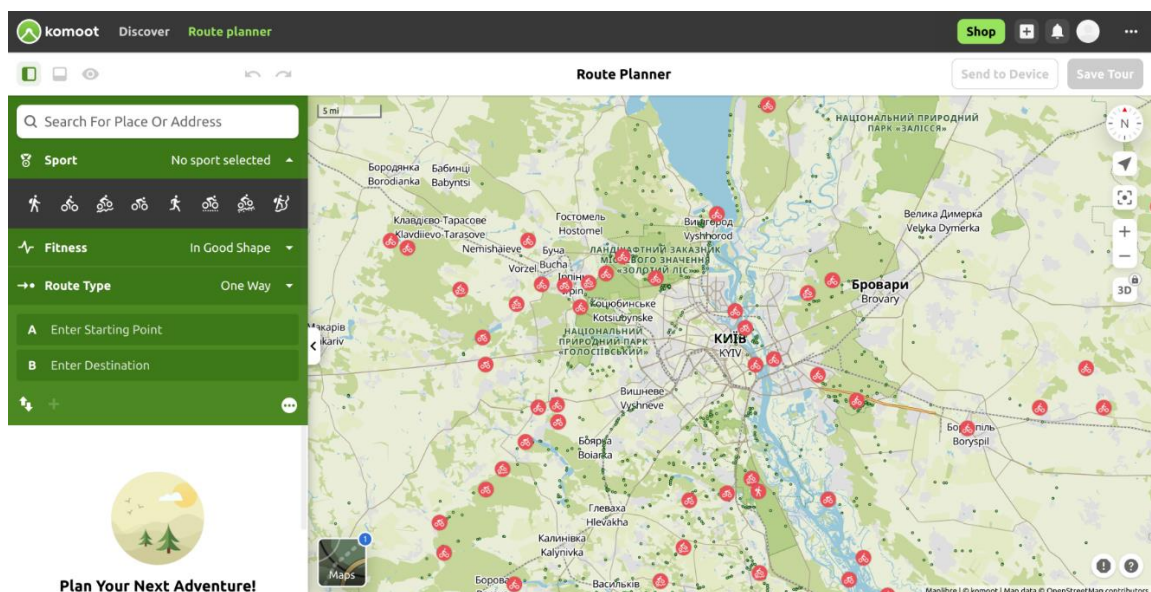


Рис. 1.8 - Komoot

Основні риси Komoot включають:

Планування маршрутів для велосипедів і пішоходів: Komoot дозволяє користувачам планувати маршрути для велосипедів і пішоходів, враховуючи типи доріг, терен та інші параметри.

GPS-навігація: додаток надає GPS-навігацію для точного відстеження маршруту під час подорожей.

Картографічні дані: Komoot використовує деталізовані картографічні дані для відображення маршрутів та інших природних об'єктів.

Загальні ідеї для подорожей: додаток надає ідеї для подорожей та маршрутів, що можна використовувати як вихідну точку для подорожей.

Недоліки:

1. Акцент на спорті: хоча Komoot також підходить для пішоходів, проте він має більший фокус на велосипедних маршрутах, туризму, що може обмежувати аудиторію.

2. Акцент на проходженні шляху: ви можете використовувати лише створений кимось маршрут, або пройти власний і лише тоді опублікувати. Ви можете вказати, який шлях хочете подолати і після цього система покаже, які потенційні місця ви зустрінете.

3. Обмежена безкоштовна версія: деякі функції і карти доступні лише за плату, і безкоштовна версія може бути обмеженою.

TripAdvisor – це популярний вебсайт та додаток, спеціалізований на відгуках та рекомендаціях для подорожуючих. Він охоплює широкий спектр інформації про подорожі, включаючи готелі, ресторани, визначні місця та активності (рис. 1.9).

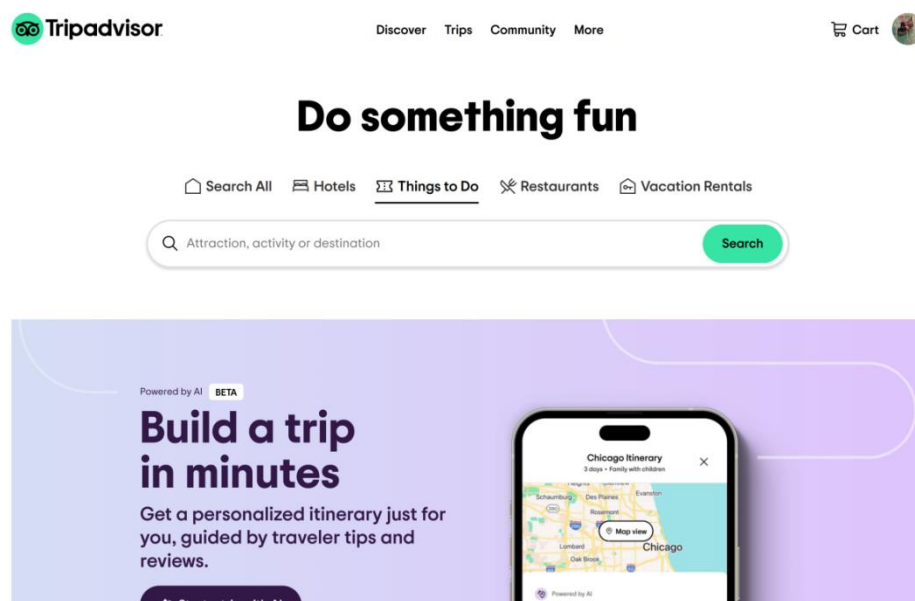


Рис. 1.9 - Tripadvisor

Основні риси TripAdvisor включають:

- відгуки та рейтинги. Користувачі можуть залишати відгуки і надавати рейтинги для готелів, ресторанів і місць, які вони відвідали;
- пошук за місцем та видом діяльності. TripAdvisor дозволяє користувачам шукати об'єкти за місцем розташування і типом діяльності;

- фотографії та інформація. Сайт містить фотографії та детальну інформацію про різні об'єкти;
- бронювання готелів та ресторанів. Користувачі можуть робити бронювання готелів і ресторанів безпосередньо через сайт;
- поради та рекомендації від подорожуючих. TripAdvisor надає поради та рекомендації від інших подорожуючих, що допомагає користувачам знайти найкращі місця для відвідування.

Незважаючи на переваги, існують певні недоліки. Немає змоги прокласти маршрут між визначними місцями. Деякі додаткові функції та доступ до певних інформаційних ресурсів може залежати від платної підписки, що може викликати деяке незадоволення серед користувачів.

Також важливо провести детальне порівняння цих вебсервісів, що можна знайти у табл. 1.1.

Таблиця 1.1.

Порівняння вебсервісів

Параметр	Google Карти	izi.Travel	AllTrails	Komoot	TripAdvisor
Карти	Google Maps	Google Maps	OpenStreetMap	OpenStreetMap	Google Maps
Бізнес-реклама	Так	Ні	Ні	Ні	Так
Офіційно підтримувані веббраузери	IE7+, Firefox 2.0.0.8+, Safari 3+, Mozilla 1.7+, Opera 8.02+, Google Chrome 1	Невідомо	Chrome, Firefox, Safari, Edge, IE	Chrome, Firefox, Safari, Edge, IE	Chrome, Firefox, Safari, Edge
API для розробників	Так	Так	Ні	Так	Так
Вартість платної версії	Безкоштовний	Покупка окремих турів	Від \$35.99/рік	Від €3.99/місяць	Від \$29.99/рік
Збереження маршрутів та об'єктів	Так	Ні	Так	Так	Так
Інтеграція зі спортивними додатками	Ні	Ні	Ні	Так	Так
Рейтинги та відгуки	Так	Так	Так	Так	Так
Оцінка локальності	Так	Так	Так	Ні	Так

Табл. 1.2 детально розглядає функціональні можливості кожного сервісу, включаючи пошук та фільтрацію маршрутів, доступні види активного відпочинку, наявність інформації про маршрути та об'єкти, можливість додавати місця та інше.

Таблиця 1.2.

Функціональні можливості

Функціональність	Google Карти	izi.Travel	AllTrails	Komoot	TripAdvisor
Пошук маршрутів	Так	Ні	Так	Так	Так
Фільтрація маршрутів	Так	Ні	Так	Так	Так
Активні відпочинкові маршрути	Пішоходи, велосипед, авто, тощо	Інформація відсутня	Пішоходи, велосипед, лижі, біг тощо	Велосипед, пішоходи, біг тощо	Всі види відпочинку
Інформація про маршрути	Довідник, оцінки, відгуки, фото	Довідник, оцінки, відгуки, фото	Довідник, оцінки, відгуки, фото	Довідник, оцінки, фото	Оцінки, відгуки, фото
Робота з закладками	Так	Ні	Так	Так	Так
Платна версія	Немає	Покупка окремих турів	Є, з додатковими функціями	Є, з додатковими функціями	Є, з додатковими функціями
Можливість додавати місця	Так	Ні	Ні	Так	Так
Аудіовідтворення інформації	Ні	Так	Ні	Так	Ні
Детальна інформація про об'єкт	Так	Ні	Так	Так	Так
Прокладання маршруту між точками	Так	Ні	Так	Так	Ні

Ці п'ять сервісів відрізняються за своєю спеціалізацією та функціональністю, проте всі вони спрямовані на полегшення подорожей та надання корисної інформації для подорожуючих.

З урахуванням аналізу конкурентів, наш вебзастосунок повинен поєднувати переваги цих продуктів та покращити їхні недоліки. Основна мета полягає у забезпеченні користувачам можливості прокладати оптимальні маршрути між визначними місцями, які користувач додав у закладки, надаючи докладну інформацію про об'єкти, а також можливість аудіовідтворення інформації для

збагачення вражень під час подорожей та для полегшення користування продуктом.

Отже, метою є впровадження функціоналу для прокладання найоптимальнішого шляху між визначними місцями, які користувач додасть собі в список закладок. Він доповнить основні характеристики цього вебзастосунку, такі як наявність мапи, можливість пошуку та фільтрації об'єктів, достатня кількість інформації про об'єкт, аудіовідтворення опису визначного місця, система аутентифікації користувачів, а також можливість пропонування нових місць. Це дозволить користувачам ефективно планувати свої маршрути та отримувати найкращі шляхи між їхніми обраними визначними місцями.

1.3. Постановка задачі дослідження

Головною метою системи визначення оптимального маршруту у вебзастосунку визначних місць «Vikha» є надання можливості користувачам самостійно створювати путівники зі списку визначних місць, який вони створили, та прокладання оптимального маршруту між цими місцями. Дане впровадження у вебзастосунок буде корисним для широкого кола користувачів, які подорожують, відвідують нові місця або проявляють інтерес до історії та культури рідної місцевості. Об'єднання цього корисного функціоналу в одній системі надасть перевагу «Vikha» перед конкуруючими аналогами. Крім того, у майбутньому передбачається можливість розширення та вдосконалення функціоналу цього вебзастосунку.

Модель розроблюваного програмного комплексу базується на задачі прокладання оптимального маршруту для об'єктів на мапі. Мапа представляється у вигляді графа, де вершини відповідають об'єктам, які необхідно відвідати, а ребра – відстаням або часу проходження між об'єктами. Метою є знаходження найкоротшого або найшвидшого маршруту, який охоплює всі місця та не повертається в початкову точку.

У межах даної роботи передбачено розширення функціоналу вебзастосунку «Vikha», розробленого для випускної кваліфікаційної роботи бакалавра [14], а

саме – додавання нового інноваційного компонента, який дозволить автоматично прокладати найоптимальніші маршрути між визначними місцями на мапі. Цей новий функціонал покликаний полегшити користувачам процес переміщення та забезпечити їм можливість ефективно обирати шляхи між цікавими локаціями, що робить використання вебзастосунку ще більш зручним та цінним для подорожуючих та людей, які цікавляться визначними об'єктами.

Основною задачею кваліфікаційної роботи буде розробити алгоритм знаходження оптимального маршруту між визначними місцями. Цей етап буде включати наступні кроки:

1. Розробити метод формалізації та структурування вхідних даних, що представляють мапу та її об'єкти, включаючи координати, ідентифікатори, час та відстані між об'єктами.

2. Отримання інформації про час та відстань між парами точок, а також геометрію.

3. Вирішити окремий випадок задачі комівояжера, а саме – проблему Гамільтонового шляху для прокладання оптимального маршруту на основі формалізованих даних.

4. Формалізація вихідних даних: сформулювати результати з інформацією про оптимальний маршрут.

5. Впровадження додаткових оптимізаційних методи для вдосконалення прокладання маршруту, такі як: варіанти вибору точки старту, вибір методу проходження шляху, прокладання найшвидшого чи найкоротшого.

6. Провести тестування розробленого програмного комплексу з використанням реальних тестових наборів даних для перевірки коректності та ефективності роботи алгоритмів.

Оскільки вже описано етапи реалізації алгоритму знаходження оптимального маршруту, опишемо наступні кроки, які потрібно буде виконати.

Проектування дизайну. Представляє собою процес створення вебзастосунку, що забезпечує зручний інтерфейс користувача, відповідаючи основним потребам за допомогою чіткого та привабливого інтерфейсу.

Розробка лицевіці. Охоплює створення компонентів, які взаємодіють із користувачами, таких як інтерфейс користувача, кнопки, введені дані, вебсайти та функції взаємодії. Зовнішній інтерфейс орієнтований на задоволення потреб користувача та надання позитивного користувацького досвіду.

Розробка тиловиці. Представляє собою процес створення компонентів, що відповідають за виконання основної логіки. Ці компоненти включають у себе засоби зберігання даних, інфраструктури, інтеграції з зовнішніми системами та код, написаний на одній або кількох мовах програмування. Користувачам недоступний прямий доступ до серверної частини.

Основними етапами для реалізації цього завдання є:

1. Розробити дизайн користувацького інтерфейсу.
 - 1.1. Створити загальну концепцію дизайну користувацького інтерфейсу.
 - 1.2. За допомогою графічного редактора GIMP створити графічні зображення основних елементів керування (кнопок, панелей, іконок та ін.).
 - 1.3. Зверстати та застосувати елементи інтерфейсу на основі Vue.
 - 1.3.1. Створення путівника.
 - 1.3.2. Список місць путівника.
 - 1.3.3. Деталі путівника.
 - 1.3.4. Маркер вибору місця.
 - 1.3.5. Нумерований маркер місця.
 - 1.3.6. Маршрут на мапі.
2. Проєктування та розробка основних частин вебзастосунку.
 - 2.1. Сценарій для синхронізації об'єктів.
3. Створення кінцевих точок API.
 - 3.1. Пошук найближчого місця за розташуванням у списку закладок.
 - 3.2. Створення путівника.
4. Розробка лицевіці вебзастосунку.
 - 4.1. Відстеження розташування користувача.
 - 4.2. Створення путівника.
 - 4.3. Відображення путівника.

- 4.4. Функціонал автовідтворення аудіо визначного місця при наближенні користувача.
- 4.5. Створення компонентів для відображення елементів на мапі.
- 4.6. Налаштування спілкування з API тиловиці.
- 5. Створення інструментів прокладання шляху між двома точками.
 - 5.1. Використання стороннього API або налаштування власного серверу.
- 6. Тестування вебзастосунку.
 - 6.1. Тестування API.
 - 6.2. Тестування лицевіці.

Висновки до розділу 1

У цьому розділі було розглянуто ключові аспекти, що стосуються тематики дослідженням даної кваліфікаційної роботи. На цій основі визначено напрямки подальших досліджень для знаходження ефективних рішень проблеми.

Також проведено аналіз існуючих рішень, що дозволив виділити ключові аспекти майбутньої системи.

РОЗДІЛ 2. РОЗРОБКА СИСТЕМИ ВИЗНАЧЕННЯ ОПТИМАЛЬНОГО МАРШРУТУ

2.1. Визначення архітектурних рішень системи

Карта на вебсайті – найкращий спосіб зробити географічні дані легко доступними для користувачів, оскільки вона просто представляє інформацію, що стосується певної географічної області, і фактично використовується багатьма онлайн-сервісами.

У даній роботі створений вебзастосунок за допомогою вебфреймворку Django на базі Python, використовуючи його модуль GeoDjango, зберігаючи географічні дані у локальній базі даних, на якій можна виконувати геопросторові запити. Також розроблено API, за допомогою якого можна буде робити запити на тиловицю вебзастосунку за допомогою лицевиді, що у свою чергу буде написана на фреймворку Vue.js.

Під час проєктування системи було вирішено зробити саме вебзастосунок. Таким чином буде забезпечено більше гнучкості в розробці, тим що розділимо логіку лицевиді та тиловиді, що вони зможуть працювати окремо.

Вебзастосунок являється клієнт-серверним, коли клієнтом виступає браузер, а сервером – вебсервер. Логіка вебзастосунку розподілена між сервером і клієнтом, зберігання даних здійснюється переважно на сервері, обмін інформацією відбувається по мережі (рис. 2.1). Серед переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому вебзастосунки є міжплатформеними сервісами.

Як тільки користувач відкриває вебсторінку, сервер надсилає браузеру певні дані у відповідь на запит користувача. Якщо бути точним, вебклієнт (або агент користувача) може запитувати вебресурси або більш відомі вебдокументи через вебсервер. Вже на наступному етапі на екрані відобразиться сторінка з необхідною інформацією. Після цього починається взаємодія між користувачем із вебсайтом. Звичайно, всі ці дії повинні виконуватися за лічені секунди. Тут важливий код, який був розібраний браузером. Цей код може мати або не мати

конкретних інструкцій, які вказують браузеру, як реагувати на широкий діапазон введених даних. У результаті архітектура вебзастосунків включає всі підкомпоненти та зовнішні додатки, що обмінюються, для цілого програмного додатка.

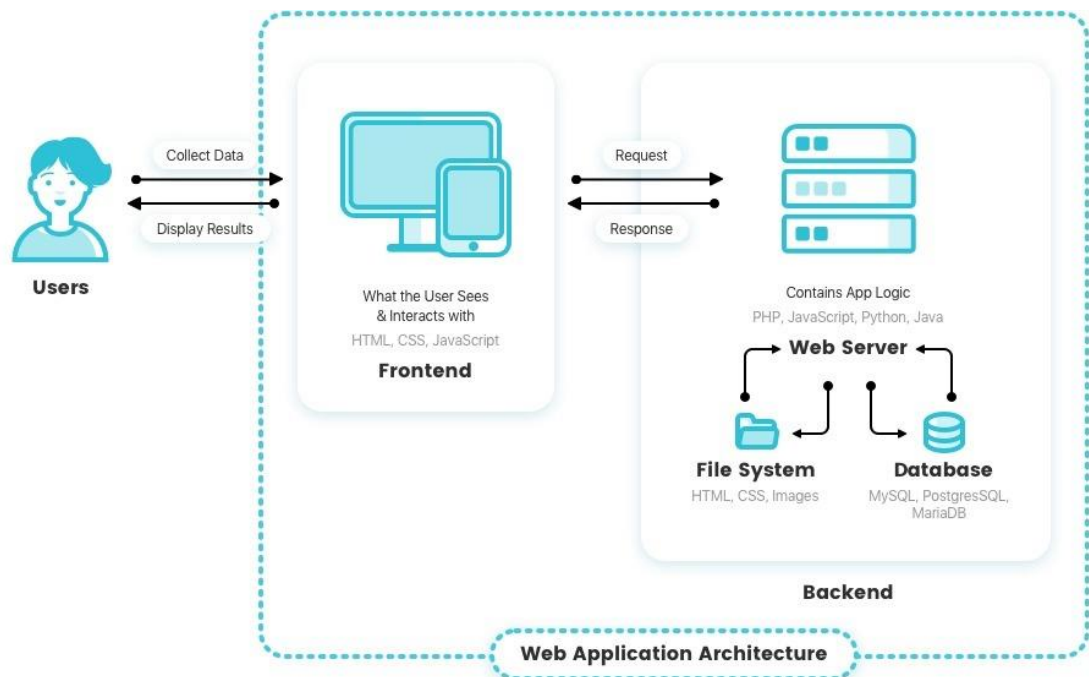


Рис. 2.1 - Схема архітектури вебзастосунка

Головні переваги вебзастосунків [15]:

- у порівнянні з настільними програмами, вебзастосунки легше обслуговувати, оскільки вони використовують той самий код у всій програмі. Проблем із сумісністю немає;
- на відміну від мобільних додатків, для вебзастосунків не потрібно схвалення жодних платформ, щоб випустити свою програму;
- вебзастосунки можна використовувати на будь-якій платформі (Linux, Mac, Windows), оскільки всі вони підтримують сучасні браузери;
- вебзастосунок простіше програмувати, оскільки він не містить багато роботи з елементами ПК (ядро, процесор, відеокарта);
- вебзастосунки – це більш економний варіант для будь-якого підприємства, адже вони не вимагають підписки або покупки ліцензій, а можуть використовуватися як SaaS-сервіс, що значно дешевше.

Як вже зрозуміло вебзастосунок – це те, що запускається в браузері, проте він містить більшу кількість елементів з якими можна взаємодіяти. Вебзастосунок та вебсайт, на перший погляд два ідентичних поняття, проте варто зауважити, що в них різні функції, а також компоненти розробки. Розглянемо більш детальну характеристику, чим відрізняється вебзастосунок і вебсайт, у табл. 2.1.

Таблиця 2.1.

Різниця між вебзастосунком і вебсайтом

Параметр	Вебзастосунок	Вебсайт
Створено для	Вебзастосунок призначений для взаємодії з кінцевим користувачем.	Вебсайт переважно складається зі статичного вмісту. Він є загальнодоступним для всіх відвідувачів.
Взаємодія з користувачем	У вебзастосунку користувач не тільки читає вміст сторінки, але й маніпулює обмеженими даними.	Вебсайт надає візуальний і текстовий вміст, який користувач може переглядати та читати, але не змінювати його.
Аутентифікація	Вебзастосунки потребують аутентифікації, оскільки вони пропонують набагато ширший спектр можливостей, ніж вебсайти.	Для інформаційних вебсайтів автентифікація не є обов'язковою. Користувача можна попросити зареєструватися, щоб отримувати регулярні оновлення або отримати доступ до додаткових опцій. Ця функція недоступна для незареєстрованих відвідувачів вебсайту.
Завдання та складність	Функції вебзастосунків є досить вищими та складними порівняно з вебсайтом.	Вебсайт відображає статичні дані та інформацію на певній сторінці.
Зміни в проєкті	Щоб внести будь-які зміни в проєкт, потрібно робити перегляд усього коду і лише опісля вносити зміни.	Зміни вносяться легко, достатньо лише зробити декілька змін у HTML коді сторінки.

У сучасних вебзастосунках широко використовується підхід зі створення API, як серверної частини зі збереженням та обробкою даних, та користувацької частини з реалізації інтерфейсу для використання даного API.

Інтерфейс API (Application Programming Interface) – це формальний спосіб опису двох комп'ютерів, що взаємодіють один з одним. І хоча існує безліч способів створення API, веб-API, які дозволяють передавати дані через всесвітнє

павутиння – в переважній більшості випадків структуровані у вигляді шаблону RESTful (REpresentational State Transfer) [16].

Передача Репрезентативного Стану (REST) – це підхід до архітектури мережових протоколів, вперше запропонований у 2000 році Роєм Філдіном у його дисертації. Це підхід до створення API поверх інтернету, що означає поверх протоколу HTTP.

Будь-який RESTful API повинен слідувати, щонайменше, наступним п'ятьом принципам:

- модель клієнт-сервер;
- відсутність стану;
- однорідний інтерфейс (Uniform interface) для ресурсів;
- шари абстракції;
- можливості кешування відповіді сервера.

Забезпечення стандарту є ключовим, оскільки воно гарантує консистентний підхід до розробки та використання веб-API.

Ідеальне поєднання Django і Django Rest Framework ефективно вирішує значну частину складнощів, пов'язаної із веб-API. Як наслідок, буде створено веб-API у вигляді набору кінцевих точок, які надають конкретні елементи вихідної бази даних (рис. 2.2).

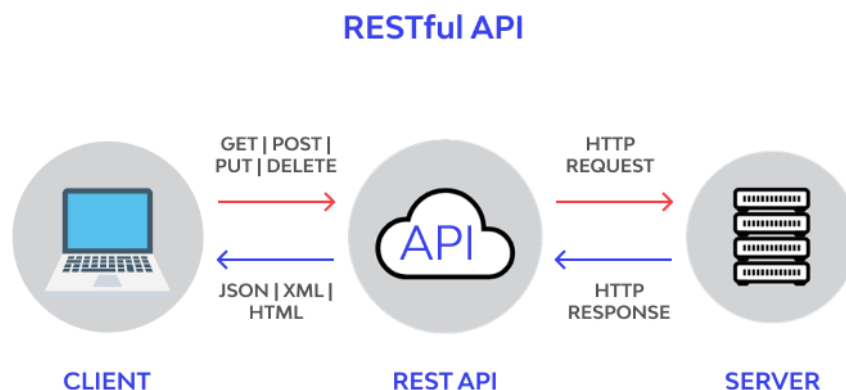


Рис. 2.2 - Схема роботи REST API

З інтродукцією асинхронних методів програмування в JavaScript-фреймворках, потреба в повному оновленні вебсторінки при кожному запиті відпала. Часткове оновлення представлення дозволяє сайтам і додаткам швидше

реагувати, хоча, з певною мірою, може виникати дублювання коду, оскільки логіка представлення часто дублює бізнес-логіку.

MVVM (Model - View- ViewModel - «модель - представлення - модель представлення») представляє наступний етап у розвитку архітектури MVC. Виділяються його особливості, такі як модель представлення (ViewModel, VM) і механізм зв'язування даних. MVVM служить основою для створення клієнтських додатків, що роблять взаємодію з користувачем та зворотний зв'язок більш гнучкими, уникненням зайвого дублювання коду у всій архітектурі. Цей шаблон спрощує модульне тестування [17].

MVVM відкриває можливості для створення вебзастосунків, які надають миттєвий відгук на дії користувача та дозволяють легко переміщатися між завданнями. Модель представлення також може виконувати декілька ролей (рис. 2.3). Ця консолідація відповідальності має ключовий наслідок для представлень додатка: будь-які зміни в даних у VM автоматично відображаються в усіх представленнях, пов'язаних з цією моделлю. Механізм зв'язування даних робить дані доступними та гарантує, що будь-які зміни в них будуть відображені в представленні [18].



Рис. 2.3 - Елементи шаблону проєктування MVVM

Розглянуті моделі проєктування MVC та MVP не забезпечують необхідної гнучкості в контексті відображення даних. Отже, для розробки інтерфейсу

вебзастосунку було обрано шаблон MVVM. Це обумовлено бажанням забезпечити високий рівень інтерактивності та зручності використання нашої системи.

Отже, можна зробити висновок, що дана архітектурна концепція надає всі необхідні технологічні можливості для розробки гнучкого додатку. Цей підхід гарантує успішне втілення проєкту, а також його подальшу адаптацію та вдосконалення. Крім того, такий підхід забезпечить ефективну взаємодію лицевіці з нашим API, яке відповідає стандартам RESTful.

2.2. Розробка бази даних системи

Спроектована та розроблена база даних включає 10 ключових таблиць, з яких 5 націлені на забезпечення логіки зберігання адрес, а інші 5 безпосередньо пов'язані із визначними місцями. Ураховуючи всі умови та вимоги до розроблюваної системи, було складено детальну схему бази даних системи (див. рис. 2.4). Уся схема бази даних представлена на рис. А.1.

Таблиця «places_place» – таблиця, яка описує визначне місце та включає наступні поля:

- uuid – поле для зберігання універсального унікального ідентифікатора;
- external_id – ідентифікатор для синхронізації зі стороннім сервісом;
- name – назва визначного місця;
- phone_number – контактний номер;
- description – опис визначного місця;
- location – спеціальне GeoDjango поле для збереження точки;
- created_at – час створення запису визначного місця;
- updated_at – час оновлення запису визначного місця;
- email – електронна пошта для контакту;
- website – посилання на вебсайт визначного місця;
- is_active – булеве поле, що визначає, чи місце активне, і чи його потрібно відображати;

- `is_paid_entrance` – булеве поле, що вказує, чи вхід у дане визначне місце є платним;
- `audio_file` – аудіофайл з описом визначного місця;
- `status` – поле статусу, що визначає, чи запис даного визначного місця, наприклад, знаходиться на розгляді, чи є опублікований;
- `address_id` – ідентифікатор на запис пов’язаної адреси з місцем;
- `place_type_id` – ідентифікатор на запис пов’язаного типу з місцем;
- `user_id` – ідентифікатор на запис користувача, що додав запис;
- `id` – ідентифікатор місця.

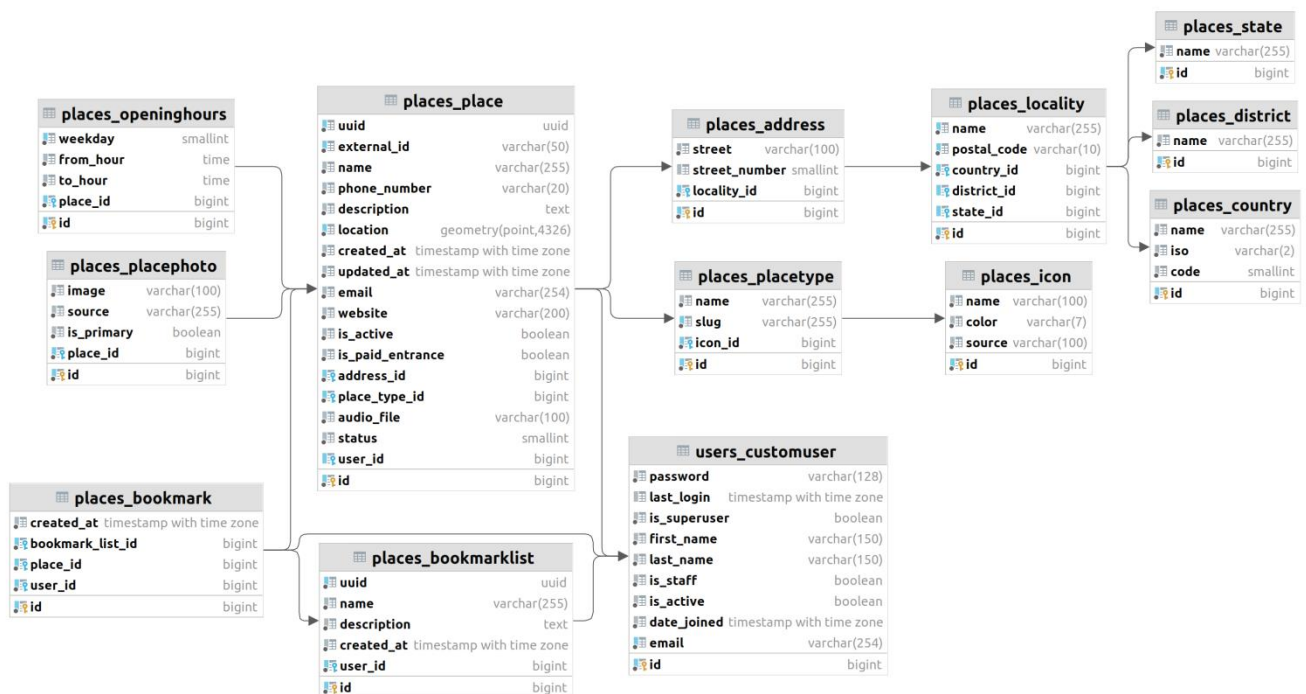


Рис. 2.4 - Схема таблиць бази даних, що відповідають за збереження місць

Важливо відзначити, що для зберігання інформації про розташування використовується поле `PointField`, яке є специфічним геометричним полем `GeoDjango`. Це поле призначене для зберігання об'єкта `GEOS Point`, який визначається парою координат довготи та широти.

Крім того, існують інші таблиці, що мають пряме відношення до основної таблиці з місцями. Серед них: `places_placetype`, `places_placephoto`, `places_address`, `places_state`, `places_country`, `places_locality`, `places_bookmark` та `places_bookmarklist`.

Також значну роль відіграють самі користувачі, які мають можливість створювати власні списки закладок і додавати визначні місця до цих списків. У зв'язку з цим необхідно зберігати інформацію про зареєстрованих користувачів у базі даних для можливості обробки їхніх даних та регулювання доступу до певних ресурсів. На рис. 2.5 представлена схема таблиць бази даних, які пов'язані із користувачем.

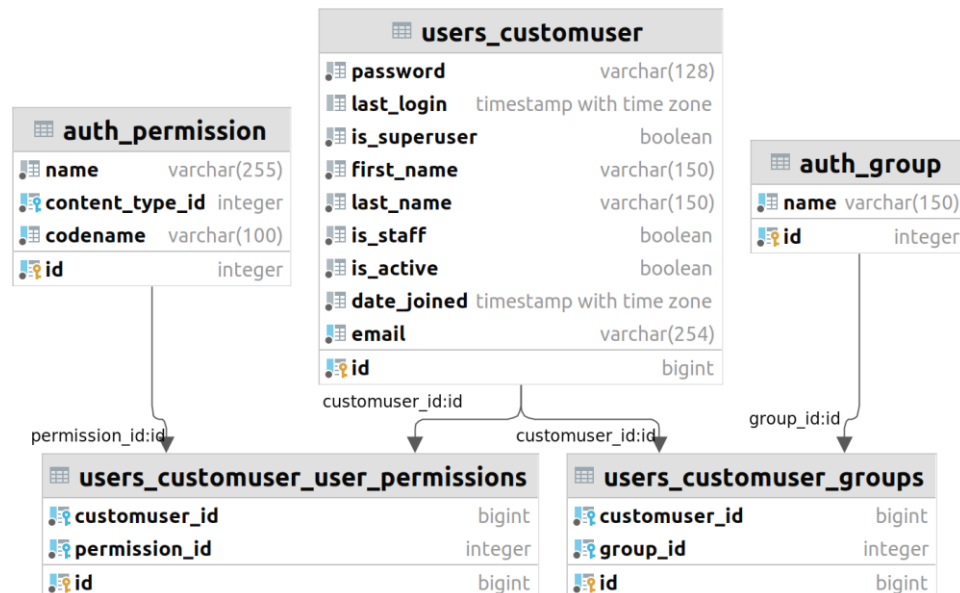


Рис. 2.5 - Схема таблиць бази даних, що відповідають за збереження користувача

Таблиця «users_customuser» – таблиця з інформацією про користувачів, що зареєструвалися у вебзастосунку. Містить наступні поля:

- password – зашифрований пароль користувача;
- last_login – час останньої авторизації користувача;
- is_superuser – булеве поле, що вказує, чи даний користувач є суперкористувачем та має всі права доступу до адміністративної панелі;
- first_name – ім'я користувача;
- last_name – прізвище користувача;
- is_staff – булеве поле, що вказує, чи користувач є персоналом;
- is_active – булеве поле, що вказує, чи користувач є активним;
- date_joined – час створення користувача;
- email – електронна адреса користувача;
- id – ідентифікатор користувача.

Стратегічне проєктування архітектури бази даних на початковому етапі відіграє ключову роль у розвитку проєкту. Це визначає зручність доступу до даних, ефективність виконання запитів та оптимальне використання ресурсів обраної бази даних.

2.3. Розробка алгоритму створення путівника з прокладанням оптимального маршруту

Найважливішою частиною нашої системи є алгоритм пошуку найкоротшого Гамільтонового шляху. Задача полягає у знаходженні такого шляху з мінімальною сумою ваг у ребрах. Варто навести основні методи, що будуть використовуватися в даному алгоритмі, у табл. 2.2. Повна реалізація алгоритму, а саме його програмний код наведено в додатку Б.

Алгоритм використовує рекурсивний підхід та перестановки вершин для перебору різних можливих шляхів, при цьому вибирається той, що має найменшу вагу. Часова складність нашого алгоритму складає $O(n!)$. Діаграма роботи алгоритму) представлена на рис. В.1.

Таблиця 2.2.

Опис функцій алгоритму

Функція	Опис
is_valid	Перевіряє, чи може вершина v бути додана до поточного шляху. Вона перевіряє, чи існує ребро між останньою вершиною у шляху та поточною вершиною v , і чи вершина v вже була відвідана. Це допомагає у визначенні можливості розширення поточного шляху.
calculate_path_weight	Обчислює вагу шляху, використовуючи зважену матрицю суміжності графа. Вона сумує ваги ребер шляху між послідовними вершинами.
hamiltonian_path_weight_util	Є рекурсивною і використовується для конструювання Гамільтонового шляху. Вона перебирає різні вершини як кандидатів для наступної вершини у шляху, перевіряє їх валідність та рекурсивно продовжує конструювання шляху. Якщо шлях знайдено, функція повертає True.
shortest_hamiltonian_path	Ініціалізує початкову вершину та викликає рекурсивну функцію для пошуку Гамільтонового шляху. Якщо шлях не знайдено, виводиться повідомлення про його відсутність. В іншому випадку, вона перебирає всі можливі перестановки вершин та знаходить ту, яка має найменшу вагу.

Результат алгоритму містить інформацію про найменшу вагу шляху та сам шлях у вигляді кортежу. Візуальне представлення алгоритму зображено діаграмою послідовності на рис. 2.6.

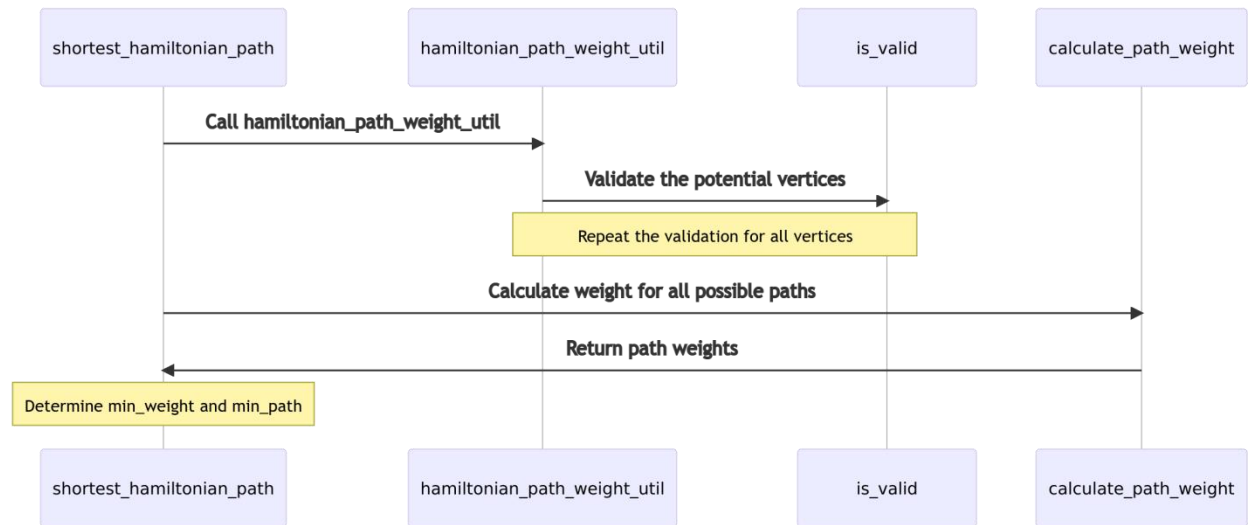


Рис. 2.6 - Діаграма послідовності роботи алгоритму

Кожна функція в даному коді має конкретний вхід та вихід, а отже, приймає певні параметри і повертає результати обчислень (див. табл. 2.3).

Таблиця 2.3.

Опис аргументів функцій алгоритму

Функція	Параметри входу	Повернення
1	2	3
is_valid	v: поточна вершина, яка розглядається для додавання до шляху. pos: поточна позиція у шляху. path: список, що представляє поточний шлях. graph: зважена матриця суміжності графа.	True, якщо вершина v може бути додана до шляху, інакше False
calculate_path_weight	path: список, що представляє шлях. graph: зважена матриця суміжності графа.	Суму ваг усіх ребер шляху
hamiltonian_path_weight_util	graph: зважена матриця суміжності графа. path: список, що представляє поточний шлях. pos: поточна позиція у шляху. start_vertex: початкова вершина шляху.	True, якщо Гамільтоновий шлях знайдено, інакше False

1	2	3
shortest_hamiltonian_path	graph: зважена матриця суміжності графа. start_vertex: початкова вершина для пошуку Гамільтонового шляху (типове значення 0).	Кортеж, що містить інформацію про найменшу вагу гамільтонового шляху та сам шлях. У випадку відсутності Гамільтонового шляху виводить повідомлення про це та повертає None

Основна логіка реалізації нашої системи зосереджена в представленні `MapView` кінцевої точки (рис. 2.7). Варто розібрати основні кроки, що будуть виконуватися. Повна реалізація представлення, а саме його програмний код наведено в додатку Г.

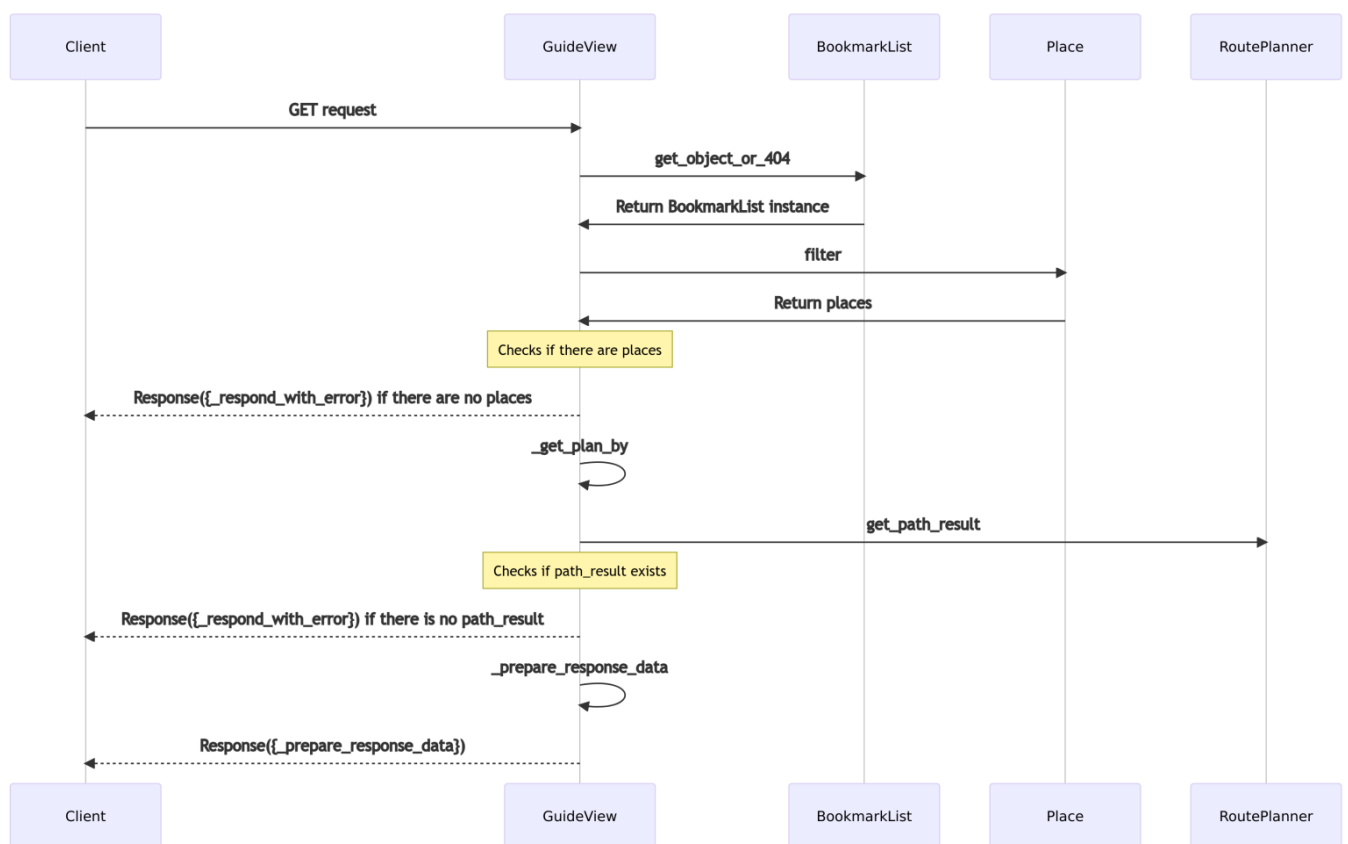


Рис. 2.7 - Діаграма послідовності для виконання запиту до представлення `MapView`

На лицевіці нашого вебзастосунку спочатку ми визначаємо місцезнаходження користувача. Після цього робимо запит на кінцеву точку, де передаємо широту та довготу з місцезнаходженням користувача, а також UUID списку закладок. Натомість API нам поверне найближче визначне місце до переданого розташування серед списку закладок.

Коли користувач зробив запит на створення путівника, ми на лицевіці передаємо UUID вибраного початкового місця, метод проходження шляху та параметр визначення оптимальності маршруту. Далі це запит надходить у наше представлення, де ми за допомогою Django ORM дістаємо в моделі Place місця, що містяться у вказаному списку закладок.

Як вже було попередньо досліджено та описано в розділі 1 параграфі 1 - нам потрібно скласти матрицю суміжності з даними для нашого алгоритму і для цього ми використаємо систему OSRM, що ми налаштуємо для визначення шляху між точками за допомогою Багаторівневого алгоритму Дейкстри.

Тож ми в циклі проходимося по місцях і робимо запит до OSRM для поточного місця і його сусідів, та запитуємо геодані між цими двома точками. Формуємо для цих даних словник, що слугуватиме для нас графом.

```
class RouteInfoRetriever(object):
    TRAVEL_MODE_URL_MAPPING = {
        TRAVEL_MODE_WALKING: settings.OSRM_FOOT_DOMAIN,
        TRAVEL_MODE_DRIVING: settings.OSRM_CAR_DOMAIN,
        TRAVEL_MODE_CYCLING: settings.OSRM_BICYCLE_DOMAIN,
    }
    HTTP_OK_STATUS = 200

    def get_route_info_between_points(self, lat1, lon1, lat2, lon2, travel_mode=TRAVEL_MODE_WALKING):
        request_url = self.build_request_url(lon1, lat1, lon2, lat2, travel_mode)
        response = requests.get(request_url)

        if response.status_code == self.HTTP_OK_STATUS:
            data = response.json()
            route_info = self.extract_route_info(data, ["distance", "duration", "geometry"])
            return route_info
        else:
            raise Exception(f"Failed to get route info. Status code: {response.status_code}")

    def extract_route_info(self, data, fields):
        first_route = data.get("routes", [{}])[0]
        return {field: first_route.get(field) for field in fields}

    def build_request_url(self, lon1, lat1, lon2, lat2, travel_mode):
        domain = self._get_domain_for_travel_mode(travel_mode)
        coordinates = f"{lon1},{lat1};{lon2},{lat2}"
        url = f"{domain}/route/v1/{travel_mode}/{coordinates}"
        return url

    def _get_domain_for_travel_mode(self, travel_mode):
        return self.TRAVEL_MODE_URL_MAPPING.get(travel_mode, settings.OSRM_FOOT_DOMAIN)
```

Алгоритм для знаходження найкоротшого Гамільтонового шляху потребуватиме індексу початкової вершини. Якщо індекс початкової точки вершини не буде передано, то воно міститиме типові значення 0. Для знаходження в словнику індексу початкової вершини, ми скористаємося функцією, яку наведено нижче, та віднайдемо індекс, що відповідатиме ID місцю початкової вершини, що ми попередньо визначили.

```
def find_index_by_id(dictionary, target_id):
    for index, key in enumerate(dictionary):
        if key == target_id:
            return index
    return None
```

У наш алгоритм ми маємо передати матрицю суміжності, де значеннями буде тривалість у секундах, або відстань у метрах, за яку користувач подолає шлях між двома точками. Для створення матриці суміжності ми скористаємося функцією `graph_data_to_adjacency_matrix`, де аргументами будуть словник, що містить дані про відстань між місцями, та стрічка, за якою ми шукатимемо значення в словнику.

```
def graph_data_to_adjacency_matrix(graph_data, value_key):
    node_ids = sorted(set(graph_data.keys()))

    adjacency_matrix = [[0] * len(node_ids) for _ in range(len(node_ids))]

    for i, node_id in enumerate(node_ids):
        if node_id in graph_data:
            for j, other_node_id in enumerate(node_ids):
                if other_node_id in graph_data[node_id]:
                    adjacency_matrix[i][j] = graph_data[node_id][other_node_id][value_key]

    return adjacency_matrix
```

На рисунку 2.8 продемонстровано матрицю суміжності для даних потенційного графа. Ми бачимо, що в більшості випадків матриця симетрична. Цей набір даних представлено для 7 точок, початок шляху відбувається в першій вершині (0-вий індекс), профіль в OSRM задано для проходження пішки, дані представлені в часовому форматі секунд.

0.0	363.0	135.8	465.4	999.9	2325.8	0.0
363.0	0.0	396.1	342.9	797.3	2013.3	0.0
135.8	396.1	0.0	571.4	864.1	2325.8	0.0
465.4	342.9	571.4	0.0	1112.1	2162.9	0.0
999.9	797.3	864.1	1112.1	0.0	2250.8	0.0
2325.8	2013.3	2325.8	2162.9	2250.8	0.0	0.0
940.9	758.1	1046.9	475.5	1363.9	1856.0	0.0

Рис. 2.8 - Матриця суміжності

Після роботи нашого алгоритму на знаходження найкоротшого Гамільтонового шляху, ми отримаємо кортеж, що міститиме послідовність індексів місць, які ми маємо пізніше представити у вигляді маршруту. Якщо зобразити отриманий результат, то отримаємо сильний граф, на якому червоними стрілками відмічений найкоротший Гамільтонів шлях (рис. 2.9).

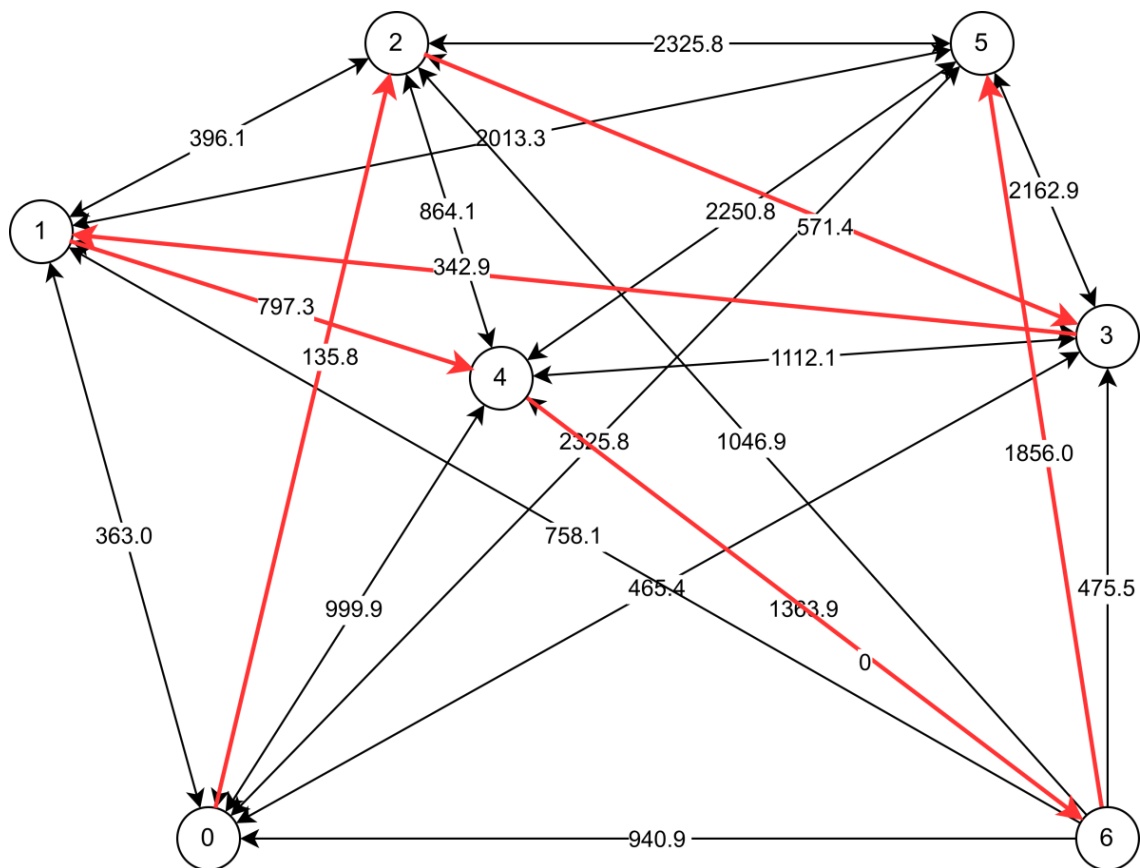


Рис. 2.9 - Граф з оптимальним шляхом

Аби відшукати відповідники індексів місць до їх ID ми використаємо функцію `get_path_details`, та отримаємо словник зі списками інформації, що містить час, відстань та список закодованої геометрії для нашої послідовності. Для зручності ми отримуємо одразу всі дані, а потім за ключем у словнику отримуватимемо бажане значення потрібного нам параметру.

```
def get_path_details(graph_data: Dict, path_sequence: List[int]) -> Dict[str, List[Union[int, str]]]:
    path_details = {}
    keys = list(graph_data.keys())

    for i in range(len(path_sequence) - 1):
        current_node_id, next_node_id = keys[path_sequence[i]], keys[path_sequence[i + 1]]

        if current_node_id in graph_data and next_node_id in graph_data[current_node_id]:
            for property_key, property_value in graph_data[current_node_id][next_node_id].items():
                if property_key not in path_details:
                    path_details[property_key] = []

                path_details[property_key].append(property_value)

    return path_details
```

Геометрія, яку ми отримали від OSRM сервісу, є закодованою, тому нам потрібно її декодувати та представити у вигляді списку, де будуть розміщені списки точок для маршруту. Це потрібно для зручності промальовування шляху на лицевіці нашого вебзастосунку. Тому, як наведено нижче в лістингу коду, у функції `decode_polyline` ми викликаємо функцію `decode` попередньо встановленого пакунку `polyline`. У свою чергу функція `flatten_coordinates` поверне нам список декодованої геометрії, яку ми безпосередньо передамо у відповідь на запит до кінцевої точки.

```
def decode_polyline(encoded_polyline):
    decoded_coordinates = polyline.decode(encoded_polyline)
    return decoded_coordinates

def flatten_coordinates(encoded_polyline_list):
    flattened_coordinates = []

    for encoded_polyline in encoded_polyline_list:
        decoded_coordinates = decode_polyline(encoded_polyline)
        flattened_coordinates.extend(list(coord) for coord in decoded_coordinates)

    return flattened_coordinates
```

Також у нашій системі присутня функція відтворення інформації про визначне місце. Таким чином користувач не повинен читати опис самотійно, а відбуватиметься його награвання, що покращить досвід користування нашим продуктом.

Якщо користувач надав доступ до свого місцезнаходження, то він може обрати параметр автоматичного аудіовідтворення опису про визначне місце. Тоді, при наближенні до визначного місця на путівнику, відобразиться компонента, що є бічною панеллю обраного визначного місця з інформацією про нього. У даній компоненті знаходиться рядок з елементом під назвою «Прослухати». Далі автоматично відбувається перевірка, чи наявний аудіофайл у даного визначного місця. Якщо він є, то відображається елемент audio та автоматично відтворюється аудіоінформація. Якщо ж такий файл відсутній, то відображається компонента text-to-speech, якій передається текст з описом місця. Після створення компоненти формуються дані та виконується запит до Google Cloud Text-to-Speech API, для проведення синтезації. Якщо дані перетворено успішно, то повертається аудіоконтент у текстовому форматі, який вставляється в HTML елемент audio, та також автоматично відтворюємо. Якщо запит завершився помилкою, то користувачу відображається сповіщення про помилку (рис. 2.10).

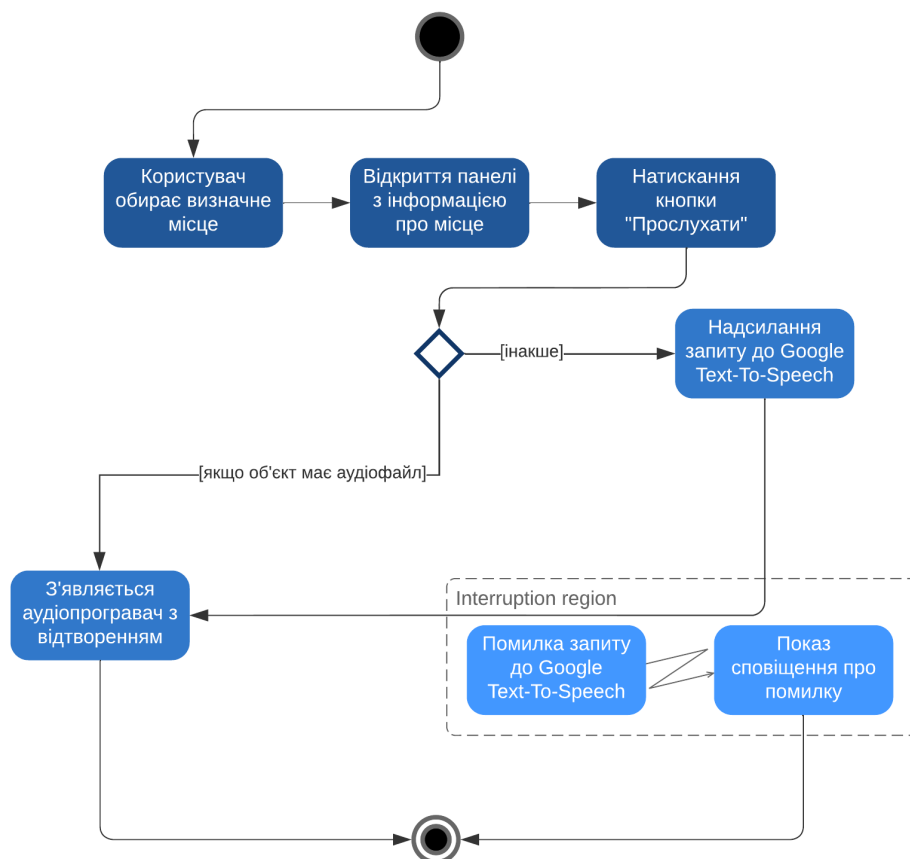


Рис. 2.10 - Діаграма діяльності аудіовідтворення опису

Лістинг коду компоненти text-to-speech наведено в додатку Г.

Таким чином, було розглянуто алгоритми основних процесів у нашій системі визначення оптимального маршруту.

2.4. Реалізація відображення та передачі даних

Серверна складова вебзастосунку представлена RESTful API, реалізованою на мові програмування Python з використанням вебфреймворку Django та інструменту створення API Django REST Framework. Вона включає основний модуль, що прямо пов'язаний з визначними місцями, а також окремий модуль для зберігання інформації про користувачів. Оскільки лицевия вебсайту реалізована за допомогою мови програмування JavaScript, дані будуть представлені у форматі JSON для більшої зручності.

Необхідно забезпечити веб-API можливістю серіалізувати та десеріалізувати екземпляри місць у формат JSON. Використовуючи Django REST Framework, нам просто потрібно створити клас серіалізатора для ефективного керування цим процесом, забезпечуючи якісну серіалізацію у JSON та його відновлення при десеріалізації [19].

Однією з найпотужніших і найкорисніших функцій Django є його Об'єктно-реляційне відображення. ORM дозволяє відображати дані між моделями додатків і базою даних без необхідності писати будь-які запити SQL. Об'єктно-реляційне відображення відображатиме атрибути об'єктів, визначені в класах моделі Django, у відповідні поля в базі даних і взаємодіяти з даними, якщо це необхідно, з заданої бази даних. Це дозволяє швидко створювати програми, усуваючи потребу писати окремі запити SQL для транзакцій бази даних.

Однією з найвпливовіших та корисних можливостей Django є його використання Об'єктно-реляційного відображення (ORM). ORM дозволяє взаємодіяти з даними між моделями додатків та базою даних, уникнувши необхідності вручну створювати SQL-запити. Цей механізм відображення автоматично перетворює атрибути об'єктів, описаних в класах моделі Django, у відповідні поля в базі даних, спрощуючи взаємодію з даними та усуваючи необхідність писати окремі SQL-запити для операцій з базою даних.

GeoDjango відмінно інтегрується з Django ORM та має в своєму арсеналі набір геометричних полів, визначених Open Geospatial Consortium (OGS), які служать для представлення різних типів геометрії в геопросторових базах даних. Основним класом для всіх геометричних полів у GeoDjango є GeometryField. Також використовується PointField для ефективного зберігання об'єктів GEOS Point.

Центральним елементом нашої бази даних є таблиця Place. Вона служить основною сховищем для ключової інформації, необхідної для детального опису визначного місця. Нижче наведено клас Place, який успадковує від класу ContactsMixin та django.contrib.gis.db.models.Model.

```
class Place(ContactsMixin, models.Model):
    class Statuses(models.IntegerChoices):
        DRAFT = 0, _("Draft")
        PUBLISH = 1, _("Publish")
        PENDING = 2, _("Pending")
        TRASH = 3, _("Trash")

    uuid = models.UUIDField(unique=True, editable=False, default=uuid.uuid4)
    external_id = models.CharField(verbose_name=_("External ID"), unique=True, max_length=50, blank=True)
    name = models.CharField(verbose_name=_("Name"), max_length=255)
    description = models.TextField(verbose_name=_("Description"), blank=True)
    location = models.PointField(verbose_name=_("Location"), blank=True)
    audio_file = models.FileField(verbose_name=_("Audio file"), upload_to="places/audio",
                                validators=[FileExtensionValidator(settings.AUDIO_TYPES), validate_audio_file_size], blank=True)
    status = models.PositiveSmallIntegerField(verbose_name=_("Status"), choices=Statuses.choices, default=Statuses.PENDING)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, verbose_name=_("User"), related_name="places",
                            on_delete=models.SET_NULL, null=True, blank=True)
    address = models.OneToOneField("places.Address", verbose_name=_("Address"), on_delete=models.PROTECT)
    place_type = models.ForeignKey("PlaceType", verbose_name=_("Type"), related_name="places", on_delete=models.DO_NOTHING)
    created_at = models.DateTimeField(verbose_name=_("Created at"), auto_now_add=True)
    updated_at = models.DateTimeField(verbose_name=_("Updated at"), auto_now=True)
    is_active = models.BooleanField(verbose_name=_("Is active"), default=True)
    is_paid_entrance = models.BooleanField(verbose_name=_("Is paid entrance"), default=False)

    class Meta:
        verbose_name = _("Place")
        verbose_name_plural = _("Places")
        ordering = ["name"]

    def __str__(self):
        return self.name
```

Django REST Framework використовує двофазний процес для здійснення серіалізації. Серіалізатори виступають посередниками між екземплярами моделі та примітивами Python, у той час як парсери та інструменти візуалізації виконують роль посередників між примітивами Python і HTTP-запитами та відповідями. Для налаштування цього посередника між екземплярами моделі Place та примітивами Python ми створюємо підклас класу

`rest_framework.serializers.Serializer`, у якому визначаємо необхідні поля та методи для ефективного управління процесом серіалізації та десеріалізації.

```
class PlaceDetailSerializer(serializers.ModelSerializer):
    location = serializers.SerializerMethodField()
    place_type = PlaceTypeSerializer(many=False, read_only=True)
    photos = PlacePhotoSerializer(many=True, read_only=True)
    full_address = serializers.CharField(source="address.get_full_address")

    class Meta:
        model = Place
        fields = ("uuid", "name", "description", "location", "full_address", "place_type", "photos",
                  "email", "phone_number", "website", "is_paid_entrance")
        read_only_fields = fields

    def get_location(self, obj):
        return [obj.location.y, obj.location.x]
```

Далі в процесі розробки виникає необхідність створення відображення для нашого серіалізатора. Зазвичай в цьому відображенні міститься ключова логіка кінцевої точки. Тут здійснюється фільтрація даних та визначається доступ до кінцевої точки. Таким чином, дане відображення представляє собою Python-клас, що успадковується від `rest_framework.generics.RetrieveAPIView`. Цей клас залишається досить компактним, оскільки `RetrieveAPIView` вже включає основну логіку та призначений для надання даних екземпляру моделі місця.

```
class PlaceDetail(RetrieveAPIView):
    serializer_class = PlaceDetailSerializer
    queryset = Place.objects.filter(is_active=True, status=Place.Statuses.PUBLISH)
    permission_classes = (permissions.AllowAny,)
    lookup_field = "uuid"
```

Для створення URL-адрес програми необхідно створити Python-модуль з назвою `urls.py`. У цьому модулі визначаються відповідності між шаблонами URL-адрес та функціями зворотного виклику Python. Як тільки один із шаблонів URL збігається, Django викликає задане представлення, яке є функцією Python. Кожному представленню передається об'єкт запиту, який містить метадані запиту, і значення, зафіксовані в шаблоні [20].

У ході роботи було додано дві ключові кінцеві точки для логіки створення путівника. А саме: кінцева точка для знаходження найближчого місця в списку

закладок користувача за геоданими та основна кінцева точка, що слугує для створення путівника за переданими параметрами.

```
bookmark_router = SimpleRouter()
bookmark_router.register( prefix: "bookmarks", BookmarkListViewSet, basename="bookmark_list")

auth_urlpatterns = [
    path("token/", jwt_views.TokenObtainPairView.as_view(), name="token_obtain_pair"),
    path("token/refresh/", jwt_views.TokenRefreshView.as_view(), name="token_refresh"),
    path("token/verify/", jwt_views.TokenVerifyView.as_view(), name="token_verify"),

    path("signup/", SignupView.as_view(), name="signup"),
    path("email/confirm/", EmailConfirmView.as_view(), name="email_confirm"),
    path("password/reset/", PasswordResetView.as_view(), name="password_reset"),
    path("password/reset/confirm/", PasswordResetConfirmView.as_view(), name="password_reset_confirm"),
]

place_urlpatterns = [
    path("places/", PlaceList.as_view(), name="place_list"),
    path("places/add", PlaceCreate.as_view(), name="place_create"),
    path("place-types/", PlaceTypeList.as_view(), name="place_type_list"),
    path("places/<uuid:uuid>/", PlaceDetail.as_view(), name="place_detail"),
]

bookmark_urlpatterns = [
    path("", include(bookmark_router.urls)),
    path("bookmarks/<uuid:uuid>/nearest-place/", NearestBookmarkedPlaceView.as_view(), name="nearest_bookmarked_place"),
    path("bookmark-item/", BookmarkViewSet.as_view({"post": "create", "delete": "destroy"}), name="bookmark_item"),
]

guide_urlpatterns = [
    path("guide/<uuid:uuid>/", GuideView.as_view(), name="guide"),
]
```

Наш API розташовано за адресою <https://<domain>/api/v1/>, де v1 позначає першу версію API. Усі кінцеві точки надалі розташовуються відносно цієї адреси. У табл. 2.4 представлені основні кінцеві точки API, HTTP-дії, які можна виконувати з ними та їх семантика.

Таблиця 2.4.

Огляд кінцевих точок API

Кінцева точка	HTTP дія	Семантика
1	2	3
auth/signup/	POST	Реєстрація користувача.
auth/email/confirm/	POST	Підтвердження пошти користувачем та активація користувача в разі успіху.
auth/password/reset/	POST	Скидання паролю користувачем.
auth/password/reset/confirm/	POST	Підтвердження скидання паролю користувачем. Надсилення нового паролю.
auth/token/	POST	Приймає облікові дані та віддає токен допуску та оновлення.
auth/token/refresh/	POST	Приймає токен оновлення та віддає оновлений токен доступу.
auth/token/verify/	POST	Приймає токен доступу та перевіряє його валідність.

Продовження таблиці 2.4

1	2	3
places/	GET	Повертає список усіх активних визначних місць.
places/?search=	GET	Повертає список усіх активних визначних місць, що задовільняють параметр пошуку.
places/?place_type=	GET	Повертає список усіх активних визначних місць, що належать до певного типу.
place-types/	GET	Список типів місць.
places/<uuid>/	GET	Екземпляр визначного місця з детальною інформацією.
bookmarks/	GET	Список списків закладок для конкретного користувача. Вимагається авторизація.
bookmarks/<uuid>/	GET	Екземпляр окремого списку закладки для конкретного користувача. Вимагається авторизація.
bookmarks/	POST	Створення списку закладок.
bookmarks/<uuid>/	DELETE	Видалення екземпляру списку закладок.
bookmarks/<uuid>/nearest-place/	GET	Повертає найближче місце серед списку закладок до вказаної геолокації.
bookmark-item/	POST	Створення екземпляру закладки.
	DELETE	Видалення екземпляру закладки. Вимагається авторизація.
guide/<uuid>/	GET	Формує маршрут відповідно до списку закладок користувача і параметрів запиту, повертаючи деталі маршруту. Вимагається авторизація.
profile/	GET	Інформація про конкретного користувача. Вимагається авторизація.
	PATCH	Оновлення даних конкретного користувача. Вимагається авторизація.
	POST	Призупинення облікового запису конкретного користувача. Вимагається авторизація.
	DELETE	Видалення облікового запису конкретного користувача. Вимагається авторизація.

Для отримання переліку визначних місць виконується запит до відповідної кінцевої точки API. Наприклад, за адресою <https://<domain>/api/v1/places/>, де domain – іменованний адрес сервера, на якому встановлене програмне забезпечення, api/v1/ – адреса нашого API за першою версією, можна отримати потрібні дані про перелік визначних місць (рис. 2.11). Дана кінцева точка вміє віддавати список місць, а також конкретний екземпляр місця, якщо передано UUID місця. Також можуть на вхід прийматися параметри для фільтрації списку місць, такі як: search та place-types. За першим параметром відбуватиметься пошук у списку місць, а за другим відбуватиметься фільтрація за типом місця, тобто за категорією.

```

{
  "count": 3,
  "next": null,
  "previous": null,
  "results": [
    {
      "uuid": "2687b956-8a5c-4849-8aaa-03dald77d48a",
      "name": "Житомирський обласний український музично-драматичний театр імені Івана Кочерги",
      "description": "Академічний український музично-драматичний театр у місті Житомирі, головна театральна сцена Житомирщини.",
      "location": [
        50.25374067937409,
        28.659247713656217
      ],
      "address": "Соборна площа, 6",
      "place_type": {
        "id": 6,
        "slug": "performing-arts-theater",
        "name": "Драматичний театр",
        "icon": {
          "image": null,
          "name": "bxs-mask",
          "color": "#1AC478"
        }
      },
      "is_paid_entrance": true,
      "photos": [
        {
          "image": "http://127.0.0.1:8000/media/places/photos/Zhytomyr_Theatre.JPG",
          "is_primary": true
        }
      ]
    }
  ]
}

```

Рис. 2.11 - Структура вихідних даних пошукового запиту

Далі важливо розглянути, як працює користувацька частина. Зазначимо, які компоненти є необхідними для лицевіці вебзастосунку, це представлено в табл. 2.5.

Таблиця 2.5.

Залежності лицевіці проєкту

Назва 1	Опис 2
Axios	HTTP-клієнт на основі Promise для браузера та Node.js
Vuex	Це бібліотека для керування станом, яка створює централізоване сховище, доступне для будь-яких компонентів програми.
Vue-router	Є офіційним маршрутизатором для Vue.js.
Leaflet	JavaScript бібліотека з відкритим кодом для інтерактивних карт, зручних для мобільних пристроїв.
Vue-leaflet	Обгортка для Leaflet.
Vue-i18n	Плагін інтернаціоналізації.
Vite	Інструмент для створення інтерфейсу.
Vuelidate	Засіб перевірки Vue на основі моделі.
mdb-vue-ui-kit	Використання матеріального дизайну.

Важливою концепцією в мові Vue є система компонентів. Ця концепція є абстракцією, що надає можливість розробки великомасштабних програм, побудованих з невеликих та автономних компонентів, які можна часто

використовувати в різних частинах проєкту. Практично будь-який тип інтерфейсу програми можна розглядати як дерево компонентів. У табл. 2.6 наведено основні компоненти, які використовуються у нашому проєкті, разом із їхніми представленнями [21].

Таблиця 2.6.

Опис основних компонент проєкту

Назва 1	Опис та призначення 2
the-map	Основна мапа на якій відображаються місця, а також присутні різноманітні панелі керування мапою.
the-marker	Застосовується аби відображати маркер на мапі разом з іконкою та підписом.
select-marker	Застосовується аби відображати маркер на мапі при виборі початкової точки для путівника.
numbered-marker	Застосовується аби відображати нумерований маркер на мапі для послідовності місць.
place-hovercard	Компонента з карточкою, що містить стислу інформацію про екземпляр місця.
guide-create	Слугує для створення нового путівника, обираючи режими подорожі, тип маршруту, початкове місце та закладки, і закінчуючи плануванням маршруту.
the-guide	Перегляд деталей вибраного путівника, включаючи список закладок, елементи маршруту, їх відстань і час, а також параметри для автоматичного відтворення аудіоопису.
guide-details	Відображення деталей про місце у путівнику. Користувач також має можливість навігації до попереднього або наступного місця в путівнику.
polilyne-path	Відображає полілінію на карті, використовуючи набір координат шляху.
user-location	Відображає маркер на мапі, що позначає географічне положення користувача.
map-filter	Спадне меню з елементами для фільтрації мапи визначних місць.
search-box	Пошукова стрічка з елементом відкриття бічної панелі головного меню.
place-detail	Бічна панель для відображення детальної інформації про конкретне місце. Тут знаходиться функціонал перегляду зображень, основні дії з місцем, а також опис та контакти.
sidebar-menu	Головне меню (бічна панель), що включає в себе елементи навігації, кнопку входу/виходу в обліковий запис та кнопку додавання нового місця.

1	2
text-to-speech	Компонента з аудіоелементом, що перетворює текст у мову.
user-profile	Містить форму з основною інформацією про поточного користувача, а також з можливістю її редагування.
settings	Містить основні налаштування.
bookmark-list	Бічна панель для відображення списків закладок користувача.
add-place	Модальне вікно з формою, що слугує для додавання користувачем нового місця.
login	Призначена для авторизації користувача.
signup	Дає змогу провести реєстрацію користувача. Користувач отримує лист на пошту з посиланням для підтвердження електронної адреси.
email-confirm	Компонента, що дозволяє підтвердити електронну адресу після реєстрації.
password-reset	Компонента з формою для скидання паролю у випадку його забуття. Користувач отримує лист на пошту з посиланням для відновлення.
new-password	Компонента для відновлення паролю при переході за посиланням.

За допомогою вищезгаданих компонентів нам вдалося сформувати ієрархію сторінок вебзастосунку, яка відображена на рис. 2.12. Можемо спостерігати, що доступ до основних компонент, які беруть участь у створенні путівника, проходить через навігацію до сторінки з закладками.

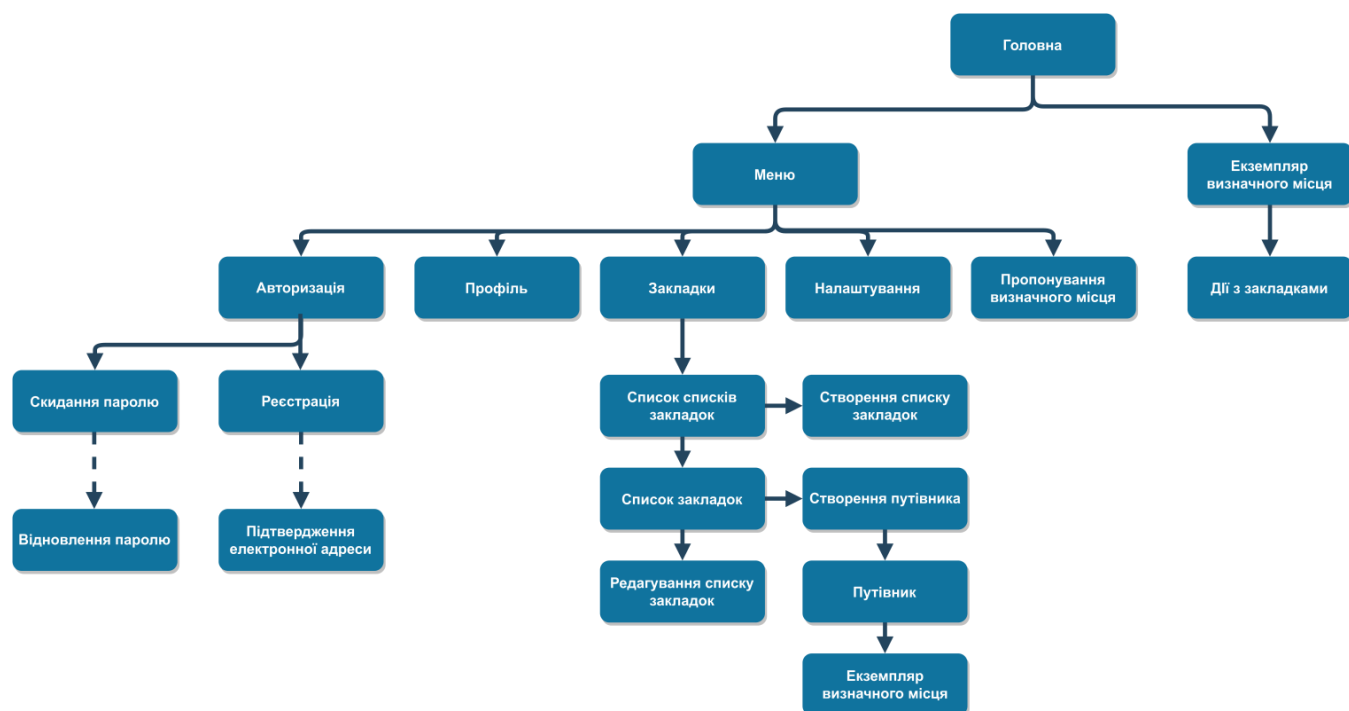


Рис. 2.12. Схема сторінок

Нижче наведено приклад виконання GET-запиту за допомогою бібліотеки Axios. У змінній APIClient вже міститься попередньо налаштована система виконання запитів до нашої API.

```
export const actions = {
  SET_PLACE_LIST({ commit }, params) {
    return APIClient
      .get(
        'places/',
        { params }
      )
      .then((response) => {
        commit('setPlaceList', response.data.results);
        return response;
      })
      .catch((error) => error);
  },
  SET_PLACE({ commit }, payload) {
    return APIClient
      .get(
        `places/${payload}`,
      )
      .then((response) => {
        commit('setPlace', response.data);
        return response;
      })
      .catch((error) => error);
  },
}
```

Під час завантаження необхідної сторінки відбуваються запити до API, результатом яких є отримання відповідних даних. Важливо відзначити, що наш застосунок підтримує динамічну обробку сторінок на стороні клієнта. Таким чином, з тиловиці на лицевицю передаються об'єкти, що містять не HTML-сторінки, а відповідну інформацію.

На рисунку 2.13 проілюстрована взаємодія лицевиці з API. У нашому проєкті використовується локалізоване сховище, створене з використанням Vuex. При переході користувача на певну сторінку Vue, застосовуючи дію (action), ініціює запит до нашого API через Axios. Після успішного отримання даних вони зберігаються або оновлюються в сховищі за допомогою мутацій (mutations). Ця мутація, у свою чергу, обробляє дані заданим способом та оновлює стан (state) [22]. Надалі ми в компоненті за допомогою getters можемо обирати потрібні нам значення сховища.

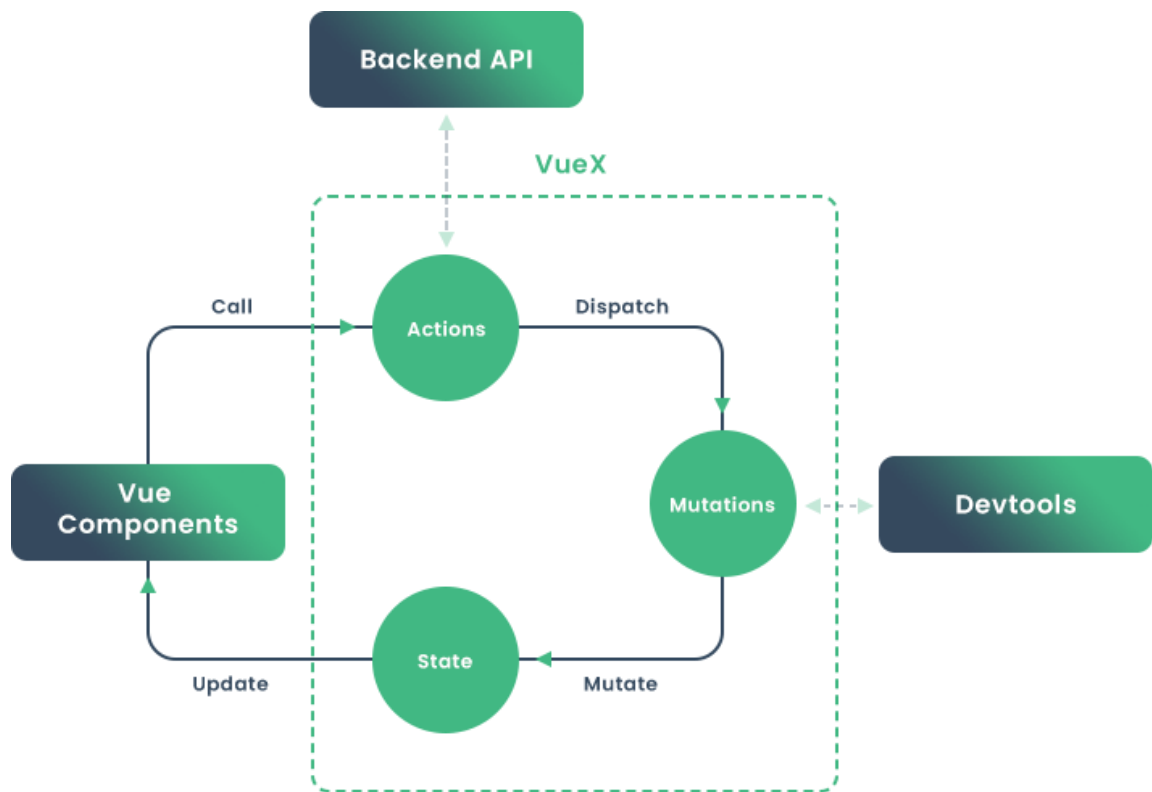


Рис. 2.13 - Спілкування лицевіці з API

Висновки до розділу 2

Було проведено розбір архітектури системи. Описано роботу алгоритму знаходження найкоротшого Гамільтонового шляху. Було наведено опис детальної роботи основного представлення API. А написання діаграм алгоритмів основних процесів системи допомогло в подальшій розробці.

РОЗДІЛ 3. ПОРЯДОК РОБОТИ, ІНТЕРФЕЙС ТА ТЕСТУВАННЯ СИСТЕМИ

3.1. Порядок встановлення та налаштування системи

Як вже згадувалося в розділі 2.1, у нашому вебзастосунку лицевия та тиловиця можуть розміщуватися на сервері окремо одна від одної. Початкове розгортання проєкту проводиться на сервері, що відповідає наступним мінімальним технічним вимогам:

- ОС «Linux». Linux рекомендовано для робочих середовищ. Docker вимагає 64-розрядної версії ОС і версії 3.10 або новішої версії ядра Linux;
- щонайменше двоядерний процесор з тактовою частотою 2.4 ГГц або вище;
- 2-4 ГБ оперативної пам'яті;
- 20 ГБ вільного місця на SSD або більше.

Для того, аби розгорнути систему на сервері потрібно виконати ряд вимог:

- 1) Треба мати встановлений Git та клонувати проєкт у папку з відповідними правами користувача. Для цього потрібно виконати команду `git clone`.
- 2) Потрібно переконаватися, що файл `.env` містить усі необхідні змінні середовища.
- 3) Потрібно встановити Docker. Рекомендовано Docker 19.03.0+.
- 4) Потрібно встановити `docker-compose`. `docker-compose`: це інструмент для визначення та запуску багатоконтейнерних програм Docker. Рекомендується версія 1.27.0 або новіша.
- 5) З кореневого каталогу проєкту запустити команду `docker-compose up -d --build`.
- 6) Запустити скрипти міграції, після побудови контейнерів для заповнення бази даних.

Після виконання вищенаведених дій, вебзастосунок повинен бути розгорнутим та готовим до використання.

Діаграма, що зазначена на рис. 3.1 відображає структуру Docker додатку, включаючи всі контейнери та їх взаємозв'язки. Кожен компонент виконується в

окремому Docker-контейнері. Контейнери мають вхідні та вихідні порти для мережових з'єднань та декілька об'ємів для зберігання даних.

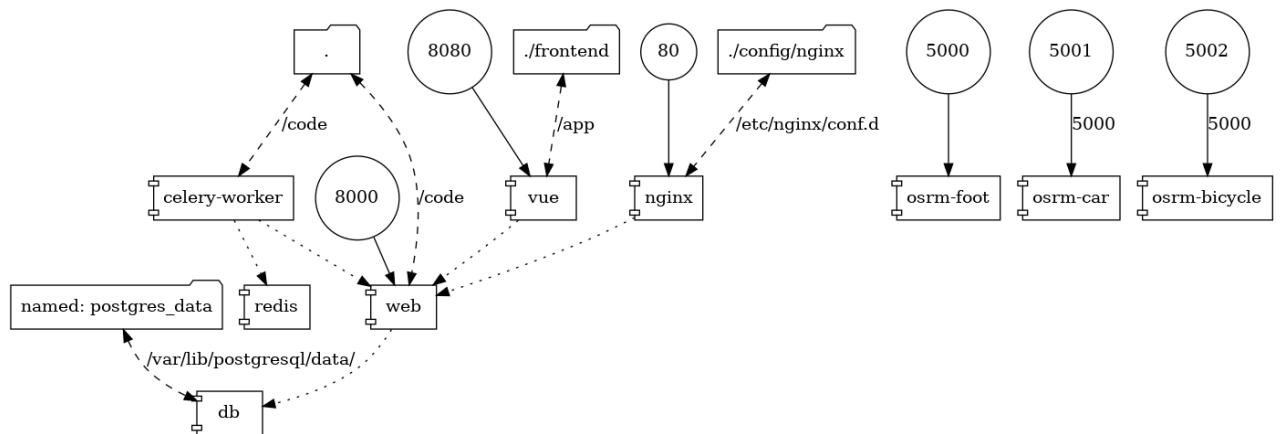


Рис. 3.1 - Діаграма розгортання системи

3.2. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для розробки вебзастосунку, що спеціалізується на обробці геопросторових даних, велике значення має вибір відповідних інструментів, які сприятимуть успішній реалізації проєкту. Ця вимога дозволяє скоротити список можливих інструментів та зосередитися на основних засобах розробки.

Багато операцій та запитів до даних у веб-ГІС відбуваються на серверному рівні. Тому вибір технології для серверної частини є ключовим аспектом розробки. Python є однією з найбільш популярних мов програмування у геопросторовій галузі. Python підтримує безліч потужних геопросторових бібліотек, таких як GeoPandas, Shapely, PyProj, а також GDAL. Ця обширна підтримка бібліотек робить Python ідеальним вибором для створення серверної частини вебзастосунку ГІС. Мова Python надає можливість легко та ефективно використовувати ці бібліотеки у серверних компонентах додатка.

Мова програмування Python – це високорівнева інтерпретована мова зі строгою динамічною типізацією. Python підтримує різні парадигми програмування, включаючи об'єктно-орієнтоване програмування (ООП) та

функціональне програмування [23]. Однією з ключових переваг цієї мови програмування є наявність великої кількості готових бібліотек, включаючи ті, які використовуються для мережевого програмування та роботи з базами даних через ORM (Object-Relational Mapping). Python відомий своєю продуктивністю та ефективністю, оскільки він дозволяє реалізовувати ідеї за допомогою меншої кількості коду порівняно з іншими мовами програмування. Ще однією сильною стороною Python є його легкість читання та зрозумілості коду, який майже аналогічний звичайному англійському тексту.

Однією з галузей, де Python проявив себе на високому рівні, є веброзробка. У світі веброзробки Python пропонує безліч фреймворків, серед яких можна виділити Flask, CherryPy, bottle.py, Pyramid, Django та web2py. Ці фреймворки використовувалися для розробки деяких з найпопулярніших вебсайтів у світі, таких як Spotify, Mozilla, Reddit, The Washington Post та Yelp.

Серед найбільш популярних фреймворків, які використовуються для розробки вебзастосунків з геопросторовими можливостями, виділяється Django. Він є одним з найширше використовуваних завдяки своїй доступності на платформі Python та великій спільноті підтримки. Django також дозволяє безпроблемно інтегрувати згадані вище геопросторові бібліотеки.

Django REST Framework (DRF) – це платформа для створення повнофункціональних API, розроблена для реалізації RESTful API з використанням фреймворку Django. В основі DRF лежать ключові елементи Django, такі як моделі, представлення та URL-адреси, що дозволяє створювати RESTful HTTP-ресурси зручно та ефективно.

Django REST Framework є вже добре вдосконаленим і повністю функціональним рішенням, яке пройшло вичерпне налаштування та тестування, а також має високий рівень документації. Тому багато завдань вже розглядаються та можуть бути легко вирішені з використанням цього інструменту. Пошук підтримки для задачі стає зручним та доступним завдяки багаторічному досвіду та розвинутій документації.

GeoDjango – це модуль для Django, призначений для роботи з геопросторовими даними. Він дозволяє взаємодіяти з різними геопросторовими базами даних, такими як PostGIS і GeoServer, зберігати та опрацьовувати геопросторові дані, виконувати запити, створювати геопросторові пошуки, серіалізувати об’єкти у форматі GeoJSON, розробляти геопросторові форми та навіть надає геопросторовий інтерфейс адміністратора.

GeoDjango також інтегровано з API GEOS і GDAL. Необхідно відзначити, що GeoDjango постачається разом із надбудовою під назвою Django REST Framework, яка надає можливість створювати API для відправки геопросторових даних у форматі GeoJSON. Цей формат широко використовується та підтримується у багатьох бібліотеках для картографії [24].

Оскільки геопросторові дані мають бути збережені у спеціалізованих геопросторових базах даних, таких як PostgreSQL, MongoDB, MySQL Spatial і Oracle Spatial, вибір PostGIS, що є розширенням PostgreSQL, є найбільш обґрунтованим. PostGIS має можливість зберігати та працювати з просторовими даними та виконувати просторові операції. Він додає підтримку для географічних об’єктів, що дозволяє виконувати запити про місцезнаходження в SQL [25].

Django офіційно підтримує різні системи управління базами даних, включаючи PostgreSQL, MariaDB, MySQL, Oracle і SQLite. Однак, наразі GeoDjango працює оптимально тільки з PostgreSQL, MySQL, Oracle і Spatialite. Після проведеного аналізу стало очевидним, що PostgreSQL, зокрема у поєднанні з розширенням PostGIS, що робить його спеціалізованим для ГІС, є найкращим вибором. Ця комбінація підтримує пошук за всіма типами геопросторових даних і має повний спектр функцій бази даних та агрегатних функцій, недоступних в інших системах. Порівняльна характеристика баз даних подана у табл. 3.1.

Таблиця 3.1.

Максимально можливий обсяг збережених даних для кожної БД

Назва БД	Розмір БД	Розмір таблиці	Розмір рядка
1	2	3	4
PostgreSQL	Необмежений	32ТБ	1.6ТБ

Продовження таблиці 3.1

1	2	3	4
MySQL	Необмежений	256ТБ	64КБ
Oracle	2ПБ-8ЕБ	4ГБ * розмір блоку	8КБ
SQLite	128ТБ	Обмежується розміром файлу	Обмежується розміром файлу

Отже, було прийняте рішення використовувати базу даних PostgreSQL, спільно з розширенням PostGIS. Це дозволить користуватися всіма доступними функціями та перевагами для вирішення поставлених завдань.

Для відповіді на потреби соціальних мереж та розвитку користувацького контенту, розробники почали створювати бібліотеки та фреймворки, спрямовані на спрощення розробки інтерактивних вебсайтів. Початково ці проєкти були великими, монолітними системами, які нав'язували своє бачення розробникам. Однак тепер спостерігається зсув у напрямку компактних і легших бібліотек, які можна легко використовувати в будь-якому проєкті.

Vue є одним із найбільш продуктивних фреймворків JavaScript для створення користувацьких інтерфейсів. Він легко впорається з більшістю варіантів використання вебзастосунків без необхідності ручної оптимізації. Однією з основних переваг Vue є його прогресивність, що дозволяє додавати підтримку вже існуючим проєктам, розширюючи їх функціональність. Крім того, цей фреймворк легко інтегрується з різними додатковими бібліотеками.

У випробуваннях стрес-тестування Vue проявляє вражаючу ефективність, випереджаючи React і Angular з великим відривом у відомому тесті js-framework-benchmark. Крім того, Vue відзначається своєю відмінною продуктивністю серед фреймворків виробничого рівня, які не використовують віртуальний DOM.

Завданням першочергового значення для Vue є створення та підтримка зв'язку між створеними представленнями та даними в моделі представлення, щоб забезпечити ефективну взаємодію з користувачем [26].

Необхідно також вибрати інструмент для збірки проєкту. Для розробки на Vue рекомендовані інструменти для збірки – Vue CLI та Vite. У табл. 3.2 нижче

наведені порівняльні характеристики зовсім нового проєкту, розробленого з використанням Vue CLI, і аналогічного проєкту, створеного за допомогою Vite.

Таблиця 3.2.

Порівняння контрольних показників

Назва інструменту	Час початку	Hot Module Replacement
Vue CLI	2591мс	171мс
Vite	259мс	~10мс

Важливо зрозуміти, що це лише вихідний шаблон, і з часом, зі зростанням проєкту, швидкість роботи версії Vue CLI буде поступово сповільнюватися, в той час як Vite гарантує однакову продуктивність, незалежно від обсягу проєкту. Саме тому було обрано Vite як інструмент для збирання проєкту.

Vite.js дозволяє розробникам налаштовувати середовище розробки для різних фреймворків, включаючи Vue і React, а також для програм із чистим JavaScript з можливістю використання вбудованого сервера розробки [27]. Середовище розробки Vite працює дуже швидко. Оскільки воно використовує вбудовану підтримку браузера для модулів JavaScript, час запуску сервера майже миттєвий. Цей підхід також гарантує, що заміна модулів «на льоту» залишається ефективною, незалежно від обсягу програми, оскільки немає потреби в повному перезбиранні всього пакету.

Leaflet – це провідна бібліотека JavaScript з відкритим вихідним кодом, яка дозволяє створювати легкі та інтерактивні карти, які оптимізовані для використання на мобільних пристроях. Створена Володимиром Агафонкіним та підтримується великою кількістю волонтерів.

Ця бібліотека також підтримує широкий спектр плагінів, включаючи постачальників базових карт, функції пошуку, спливаючі вікна та елементи управління шарами. Вона значно спрощує завдання розробників JavaScript у створенні вебкартографічних та вебгеоінформаційних застосунків.

Бібліотека Leaflet вражає своєю легкістю, займаючи всього близько 33 КБ JavaScript коду, при цьому маючи всі необхідні функції для більшості розробників. Варто відзначити, що серед користувачів Leaflet знаходяться такі відомі компанії як FourSquare, Pinterest і Flickr [28].

Google Cloud Speech API є невід’ємною складовою Google Cloud і забезпечує можливість зміни людської мови в текстовий формат. Це програмний інтерфейс, який охоплює більше 110 мов світу. Система дозволяє налаштовувати розпізнавання за допомогою списку можливих слів. Google Cloud Speech API може ефективно опрацьовувати як пакетні завдання, так і надходження даних у реальному часі. Він надійно працює в умовах шуму в аудіозаписах та здатний відфільтровувати недоречні слова для певних мов. Цей інструмент використовує глибокі нейронні мережі, і його можливості з часом можуть бути розширені та вдосконалені [29].

Оскільки Google Cloud Speech API має численні переваги, такі як висока точність розпізнавання, підтримка різноманітних мов та встановлені позиції на ринку, було обрано цей конкретний інструмент для використання.

У розробці функціоналу прокладання найоптимальнішого шляху між визначними місцями нам потрібно буде отримувати інформацію про відстань і час між точками, а також геометрію для промальовування шляху. Для цього ми використаємо інструмент під назвою Open Source Routing Machine.

Open Source Routing Machine (OSRM) надає зручний інтерфейс для взаємодії з сервісом розрахунку маршрутів. Передбачена можливість з легкістю запитувати дані про відстань та час між точками, що дозволить ефективно використовувати геолокаційні можливості у вебзастосунку. Запитуючи ці дані, можна забезпечити користувачів точною інформацією про різні маршрути та їх характеристики [30].

Використовуючи отримані дані від OSRM, розробник може легко запитувати точки для прокладання маршруту. OSRM підтримує режими автомобіля, велосипеда, пішки, це легко налаштовується через профілі. Це дозволить побудувати оптимальний маршрут з урахуванням відстані та часу, і відображати його на карті для зручності користувачів. Таким чином, вебзастосунок зможе надавати актуальну інформацію про маршрути та навігацію у реальному часі, покращуючи взаємодію з користувачами та забезпечуючи зручну навігацію.

Аналізуючи обрані інструменти для створення вебзастосунка, важливо враховувати мінімальні вимоги до серверного середовища, на якому буде встановлено цю платформу. Зазначимо, що Django може виявити труднощі при продуктивному використанні на операційній системі «Windows» через свою специфічну залежність від файлової системи «Linux». Також для оптимальної роботи Django потребує, як мінімум, 4 ГБ оперативної пам'яті, пристрій SSD, двоядерний процесор з тактовою частотою не менше 2 ГГц. Однак важливо відзначити перевагу нашої архітектури додатку, оскільки лицевий та тильний сервери можуть знаходитися на різних серверах і не обов'язково бути об'єднаними на одній машині.

3.3. Опис інтерфейсу системи

Однією з основних частин розробленої системи є клієнтська частина. Вебзастосунок мапи визначних місць з системою визначення оптимального маршруту містить у собі наступні сторінки:

- Головна – основна мапа на якій відображаються об'єкти, стрічка пошуку, фільтрація об'єктів та меню сайту;
- Профіль користувача – відображення інформації про авторизованого користувача з можливістю редагування;
- Закладки – список створених списків, у яких містяться збережені користувачем визначні місця;
- Екземпляр закладки – список визначних місць окремої закладки.
- Редагування закладки – оновлення імені списку, а також видалення місць зі списку закладки;
- Створення путівника – задання параметрів для створення оптимального маршруту для конкретного списку закладок;
- Путівник – сторінка зі списком визначних місць, що відноситься до конкретного списку закладок та мапа з оптимальним маршрутом для проходження цих місць у конкретному порядку;
- Деталі путівника – розміщена детальна інформація про обране місце;

- Вхід – авторизація користувача;
- Скидання та відновлення паролю користувача;
- Реєстрація – реєстрація нового користувача;
- Підтвердження пошти – підтвердження електронної адреси користувача для активації акаунту.

Після завантаження вебзастосунку користувач потрапляє на головну сторінку, на якій розташовано основну мапу, стрічку пошуку, елемент фільтрації, відкривання основного меню (рис. 3.2). Засоби керування мапою, такі як: масштабування, зміна шарів мапи, визначення місцезнаходження, масштаб поточного центру екрана.

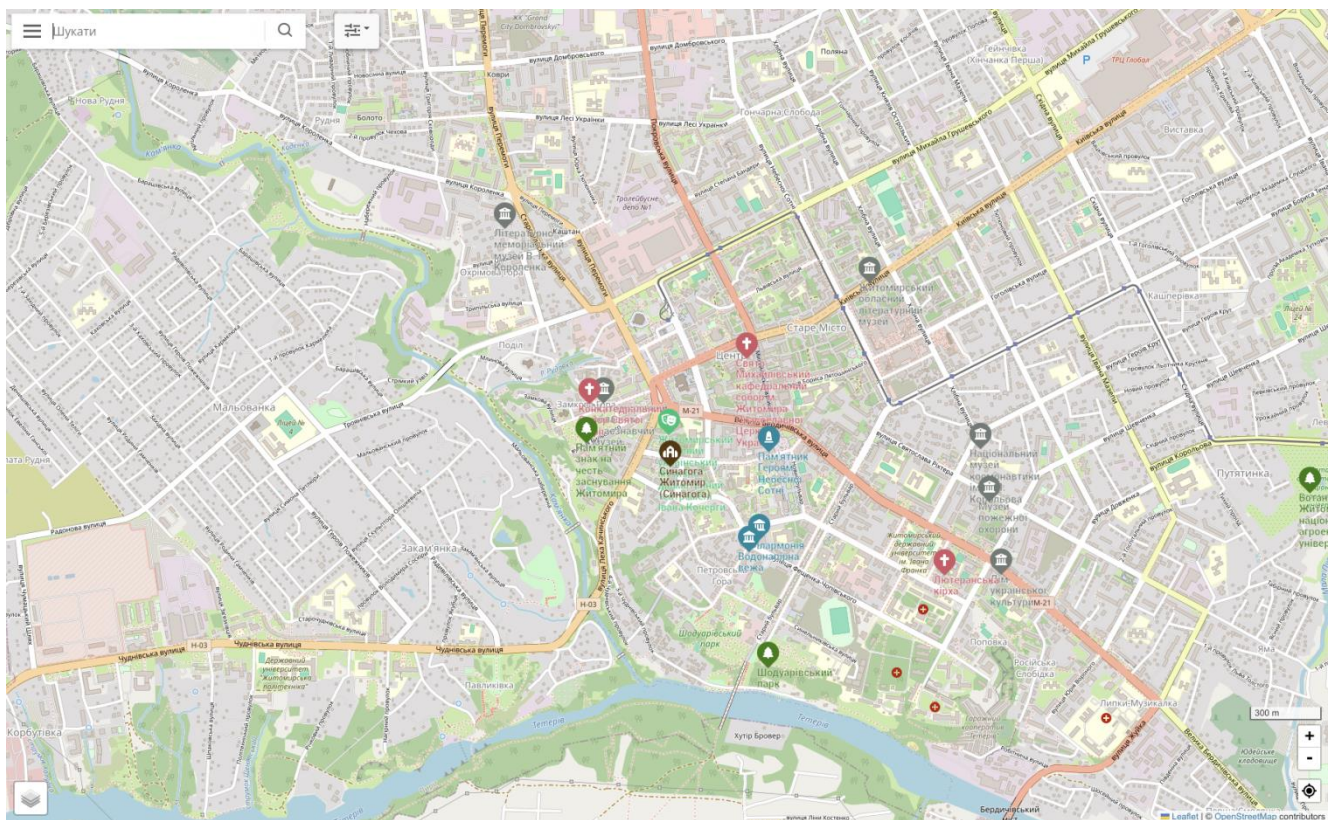


Рис. 3.2 – Головна сторінка вебзастосунку

Лише авторизовані користувачі мають право додавати місця в закладки та керувати закладками, тому попередньо потрібно виконати вхід у профіль, або зареєструватися.

Щоб створити маршрут, потрібно перейти до бажаного списку закладок з місцями. Для цього ми відкриваємо бічну панель з меню (рис. 3.3) та натискаємо на «Закладки».

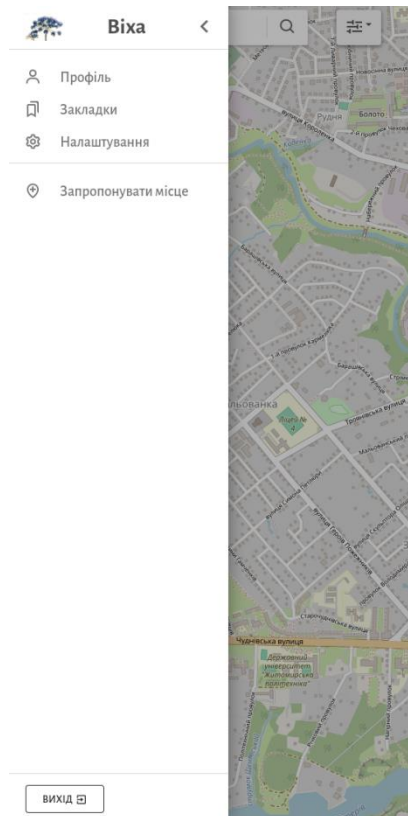


Рис. 3.3 - Бічна панель головного меню

При переході відкривається бічна панель, де розміщуються списки закладок для авторизованого користувача, що зображено на рис. 3.4.

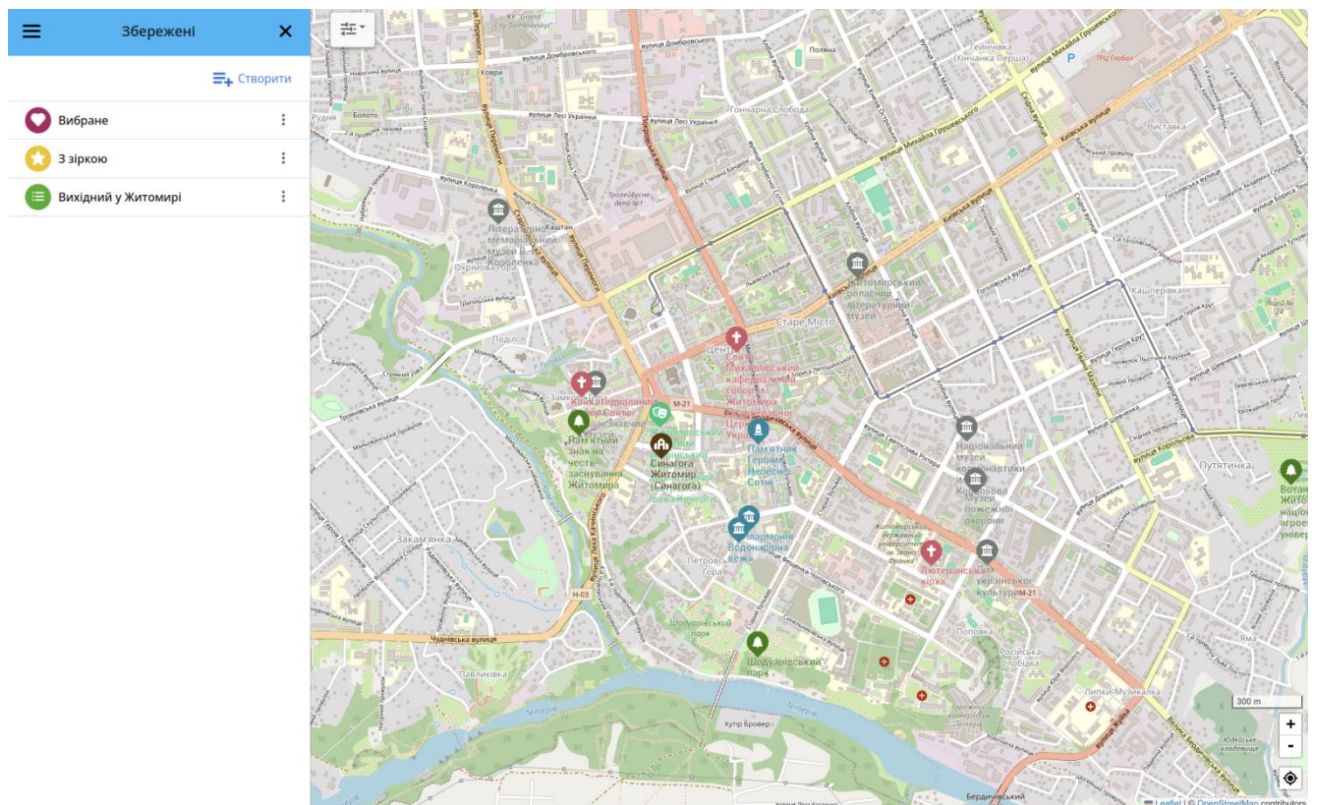


Рис. 3.4 - Сторінка закладок

Для того, аби відкрити певний список закладок, потрібно натиснути на бажаний список, тоді відкриється нова бічна панель. У цій панелі відображаються визначні місця, що додані до поточного списку закладок (рис. 3.5). Якщо список закладки є порожній, то буде відображено відповідний текст.

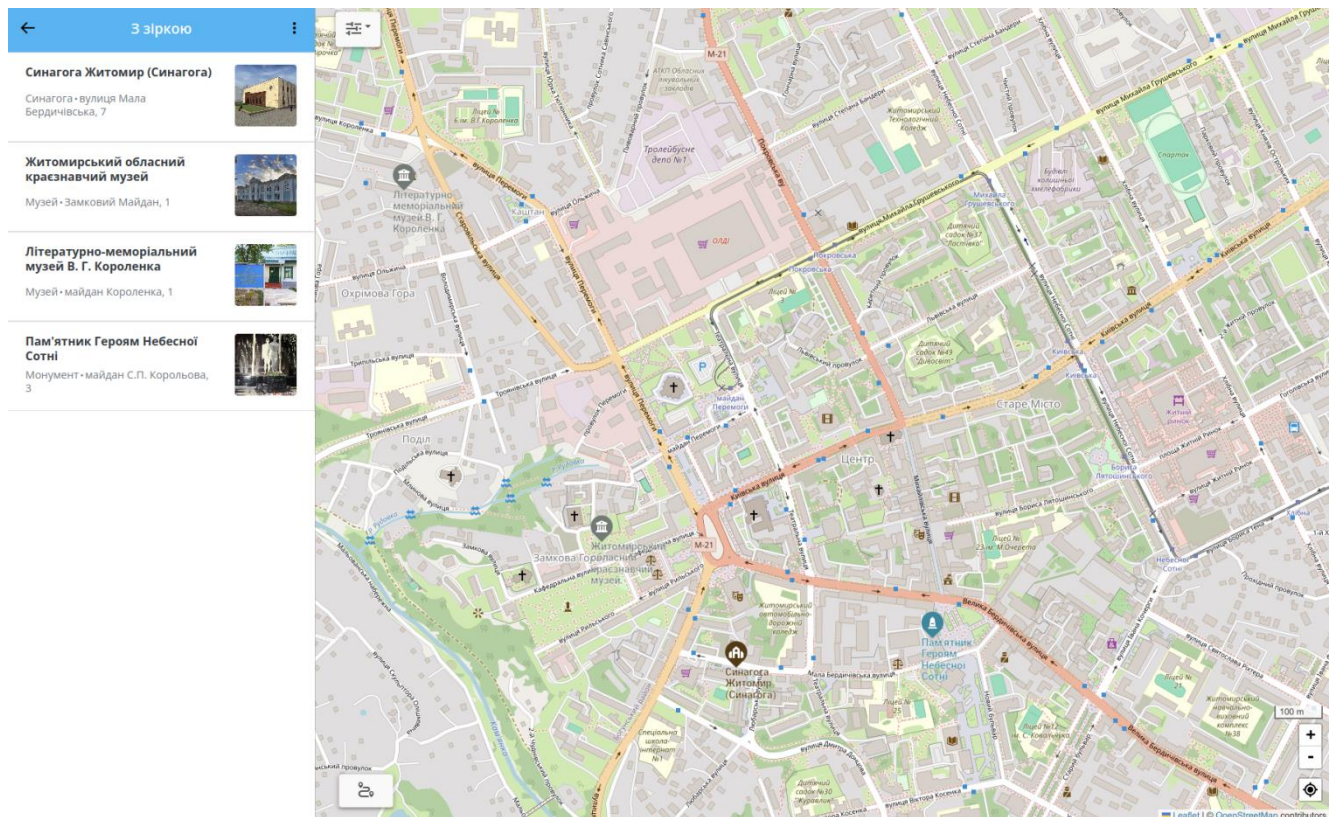


Рис. 3.5 - Список місць в закладці

Кожен елемент списку містить коротку важливу інформацію про дане місце: назва, тип місця, адреса та зображення.

У правому верхньому кутку даної бічної панелі розташований елемент для редагування поточного списку, а саме: перейменування списку, додавання опису та видалення місць зі списку.

Далі потрібно натиснути на кнопку створення маршруту, що перенаправить нас на сторінку для створення маршруту, як зображено на рис. 3.6. Варто відзначити, що якщо в списку закладок недостатня кількість місць, а саме менше 2, то користувачу буде відображено відповідне сповіщення та він залишиться на поточній сторінці. У такому випадку користувачу доведеться додати більше місць, аби отримати можливість створити путівник.

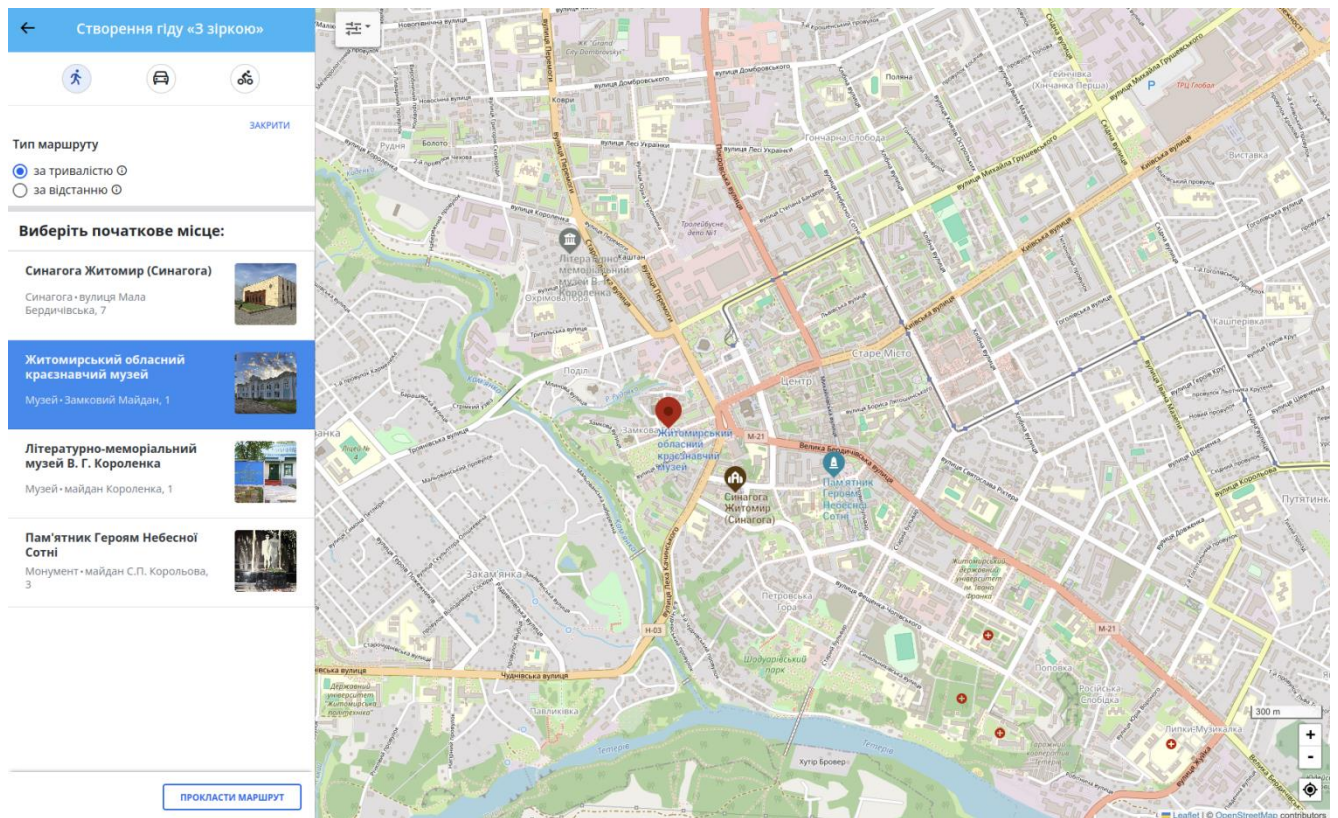


Рис. 3.6 - Сторінка створення путівника

На бічній панелі, у першому блоці ми бачимо три профілі: пішки, авто та велосипедом. Цей параметр дозволяє користувачу вибрати тип пересування під час подолання маршруту.

У параметрах нижче можна обрати чи потрібно прокласти маршрут за часом чи за відстанню. Відповідно прокладеться найшвидший або найкоротший маршрут.

Варто звернути увагу на те, що в цьому списку вже виділене місце, воно обралося автоматично, як найближче місце до місцезорештування користувача, якщо він надав дозвіл до геоданих. Також користувач може самостійно обрати стартове місце. Якщо користувач не обрав стартове місце, то він не зможе перейти до етапу створення путівника.

Після бажаних модифікацій натискаємо кнопку «Прокласти маршрут». Бачимо, що на мапі відображається маршрут та маркери з нумерацією в якому порядку варто відвідувати ці місця. При наведенні на маркер з'являється підказка з короткою інформацією про об'єкт: зображення, назва, тип визначного місця та опис (рис 3.7).

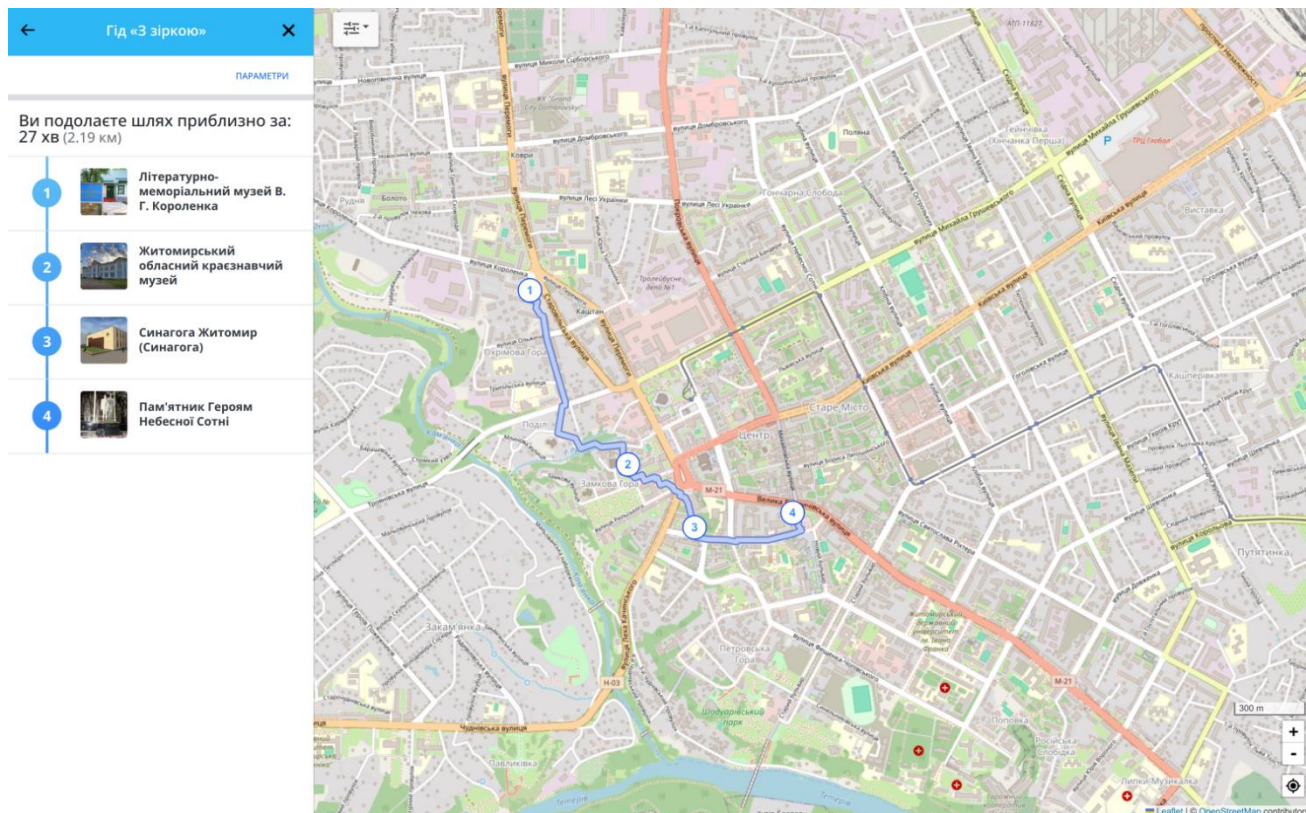


Рис. 3.7 - Сторінка путівника для списку закладок «З зіркою»

Якщо натиснути на маркер, що розташований на мапі, то відкриється панель з детальним описом визначного місця (рис. 3.8).

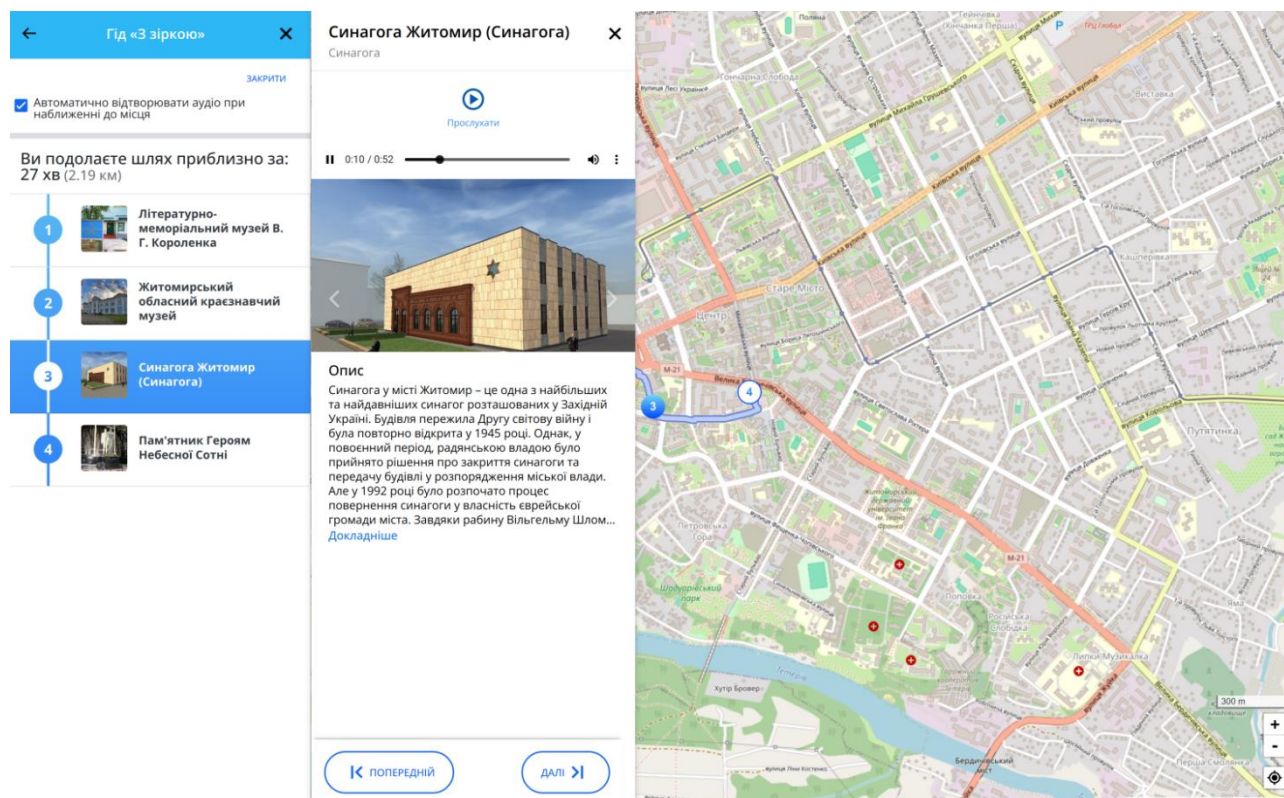


Рис. 3.8 - Відображення детальної інформації про визначне місце

На цій панелі можна прослухати опис даного визначного місця, переглянути фотографії, дізнатися детальну інформацію про це місце, як: адреса, вебсайт, контакт для зв'язку. Також присутня зручна навігація до попереднього та наступного компоненту деталей визначного місця. Є можливість встановити прапорець для ввімкнення функції автоматичного відтворення аудіо з описом про дане місце.

На рисунку 3.9 зображено, як виглядатиме сторінка путівника на малому екрані. Вміст панелі рівномірно розтягується по ширині екрану, а бічна панель з деталями путівника відображатиметься безпосередньо під елементом списку.

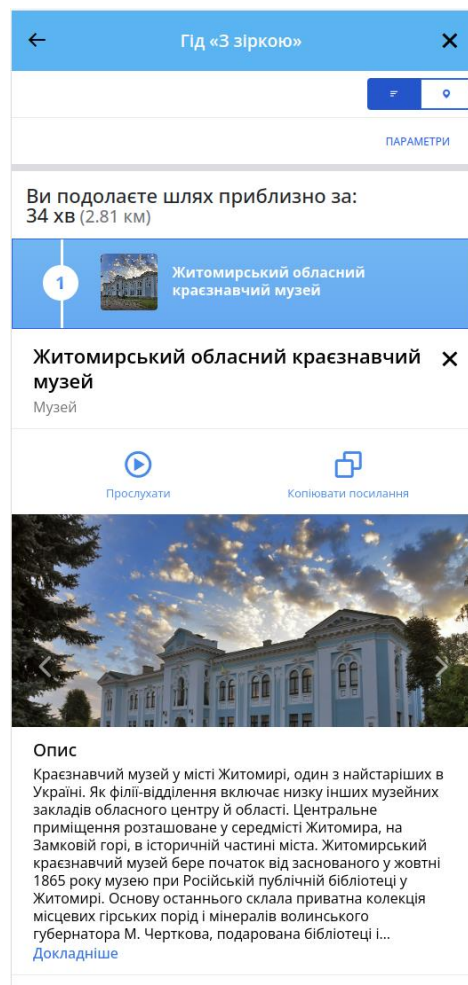


Рис. 3.9 - Відображення детальної інформації про визначне місце на малому екрані

У верхньому блоці розміщується перемикач між списком та мапою. Якщо натиснути на кнопку для представлення мапи, то основний вміст бічної панелі приховається і буде видно основну мапу (рис. 3.10).

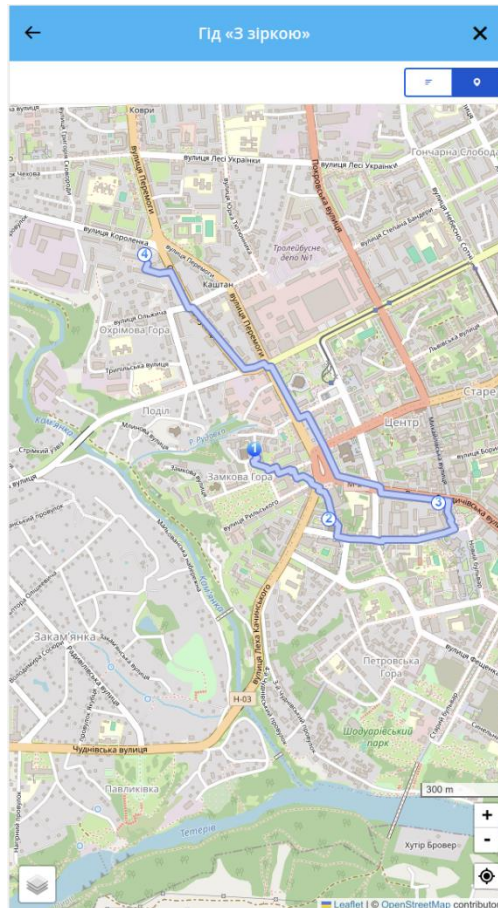


Рис. 3.10 - Відображення мапи на сторінці путівника на малому екрані

3.4. Тестування роботи системи

Після завершення розроблення вебзастосунку було проведено тестування на відповідність вимогам, поставленим у даній кваліфікаційній роботі. Тестування – це важлива частина життєвого циклу розробки програмного забезпечення, яка дозволяє забезпечити міцність коду, запобігти помилкам та поліпшити якість продукту.

Тестування функціональних можливостей та роботи елементів лицевіці проводилося за допомогою ручного тестування.

Також було створено набір модульних тестів на Python для перевірки функцій, пов'язаних з обчисленням Гамільтонівських шляхів у графі.

Базові функції, перевірені в цьому тестовому наборі, допомагають фіксувати і обчислювати ваги цих шляхів, перевіряти чи може вершина графа бути відвідана (`is_valid`), знаходити найкоротший Гамільтонівський шлях

(shortest_hamiltonian_path) і т.д. (рис. 3.11). Кожен тест створює різні сценарії, щоб переконатися, що функції правильно обробляють різні вхідні дані та відповідають на різні питання про стан графа.

: 12 total, 12 passed		1 ms
		Collapse Expand
test_shortest_hamiltonian_path		1 ms
HamiltonianPathTests		1 ms
test_calculated_weight_matches_expected_in_valid_graph	passed	1 ms
test_hamiltonian_path_does_not_exist_in_invalid_graph	passed	0 ms
test_hamiltonian_path_does_not_exist_in_self_edges_graph	passed	0 ms
test_hamiltonian_path_exists_in_valid_graph	passed	0 ms
test_no_hamiltonian_path_exists_in_small_graph	passed	0 ms
test_no_path_in_empty_graph	passed	0 ms
test_shortest_path_in_single_node_graph	passed	0 ms
test_shortest_path_matches_expected_in_valid_graph	passed	0 ms
test_single_node_path_exists	passed	0 ms
test_vertex_already_visited_in_valid_graph	passed	0 ms
test_vertex_can_be_visited_in_valid_graph	passed	0 ms
test_vertex_cannot_be_visited_in_empty_graph	passed	0 ms

Рис. 3.11 - Результат успішного виконання тестів

Перевага таких тестів полягає в тому, що вони дозволяють переконатися, що функції ведуть себе відповідно до очікувань в різних сценаріях, включаючи «крайні випадки» (наприклад, графи з єдиною вершиною або порожні графи).

Тестування API було проведено за допомогою програми Postman. Postman – це платформа API, на якій розробники можуть проєктувати, створювати, тестувати й повторювати свої API. Дана платформа пропонує такі інструменти як: клієнт API, дизайн API, документація API, тестування API, фіктивні сервери та виявлення API. Також Postman пропонує багаторівневу модель ціноутворення. Варіанти варіюються від безкоштовного плану для невеликих команд до корпоративних планів [31].

Для кожної кінцевої точки було написано скрипт, у якому вказано очікуваний статус відповіді, а також результат. Якщо запустити тести для даної колекції, то в результаті буде відображено, що вони всі проходять успішно, що відображено на рис. 3.12.

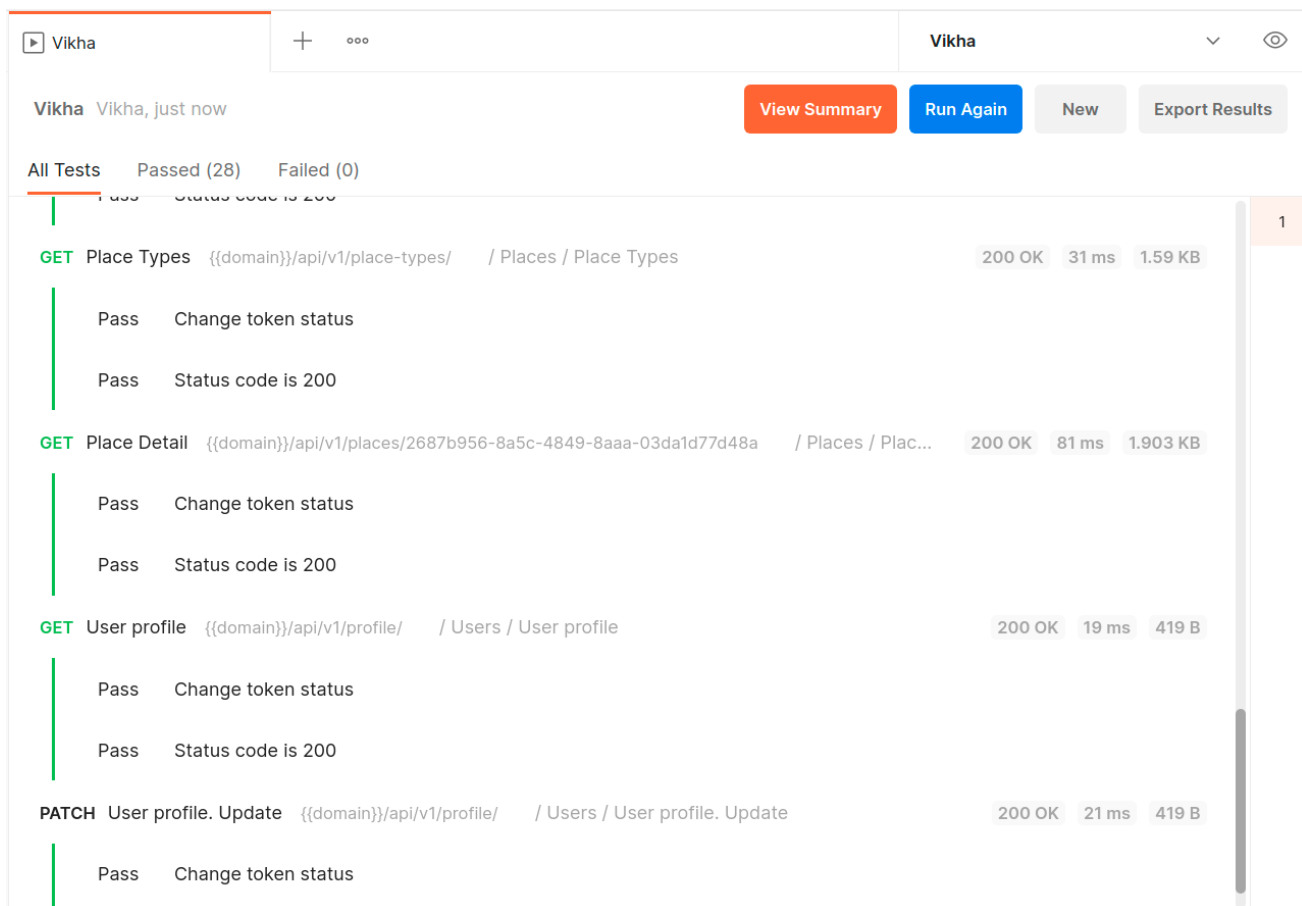


Рис. 3.12 - Результат проходження тестів у Postman

Висновки до розділу 3

Результатом опису та дотримання вимог до серверу, на якому встановлюється система, вдалося уникнути проблем під час розробки та використання системи.

Обрані інструменти розробки дозволили досягнути поставленої мети кваліфікаційної роботи.

У результаті проведення тестування розробленої системи було представлено коректну роботу її основних процесів.

ВИСНОВКИ

У даній кваліфікаційній магістерській роботі вирішено актуальне завдання з розробки системи, що дозволяє користувачам самостійного створювати путівники, з оптимальним проходженням усіх визначних місць та автоматичним аудіовідтворенням інформації про об'єкт.

Результатом кваліфікаційної магістерської роботи стало вирішення задачі Гамільтонового шляху, а саме знаходження найкоротшого Гамільтонового шляху. Для встановлення вагів ребер графу використовується OSRM з налаштуванням прокладання шляху за допомогою Багаторівневого алгоритму Дейкстри.

Основні застосовані технології включають Гамільтонів шлях, теорію графів, що використовується для планування маршруту, і різні вебтехнології, що використовуються для реалізації вебзастосунків, включаючи, але не обмежуючись, принцип вебзастосунків, використання карт і API.

У підсумку розроблено алгоритм створення путівника з прокладанням оптимального маршруту, що включало в себе реалізацію відображення та передачі даних. Розроблені параметри для налаштування прокладання маршруту, а саме: різні профілі для вибору типу пересування, оптимізація маршруту за часом чи відстанню.

Також, використовуючи геодані користувача, ми забезпечили автоматичне визначення стартового місця для путівника. Та, при відповідному налаштуванні, користувач має змогу ввімкнути автовідтворення аудіоопису про визначне місце при наближенні до нього.

Проведений докладний аналіз аспектів реалізації надав можливість отримати важливі висновки. Описані методи розробки тиловиці, зокрема API, разом із викладеними архітектурними шаблонами у вебзастосунку, дали розуміння та визначення результативних рішень у процесі виконання дослідження.

Практична цінність результатів полягає в можливості використання розробленої системи для створення путівників з автоматичним відтворенням аудіоописів визначних місць.

Напрямом подальшого наукового дослідження та вдосконалення програмного комплексу можуть стати врахування додаткових параметрів при прокладанні маршрутів, інтеграція з іншими вебсервісами та покращення інтерфейсу користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Перевалова А. Д., Чижмотря О. Г. Розробка вебзастосунку мапи визначних місць з аудіовідтворенням інформації. Матеріали III Всеукраїнської науково-практичної інтернет-конференції: «Сучасні комп'ютерні та інформаційні системи і технології» 12-19 грудня 2022 р., с. 338-340.
2. Ковтун В. В., Перевалова А. Д., Чижмотря О. В. Штучний інтелект в освіті. Тези доповідей VI Всеукраїнської науково-технічної конференції: «Комп'ютерні технології: інновації, проблеми, рішення» 29-30 листопада 2023 р.
3. Introduction to Algorithms / T.Cormen, C. Leiserson, R. Rivest, C. Stein., 2009. – 1292 с.
4. Kadry S. Social Network Analysis: An Introduction with an Extensive Implementation to a Large-Scale Online Network Using Pajek / S. Kadry, M. Z. Al-Taie., 2018. – 128 с.
5. Hamme J. Customizable Route Planning in External Memory / Hamme Janis – Karlsruhe, 2013. – 40 с.
6. Dijkstra E. W. A note on two problems in connexion with graphs. Numerische Mathematik. 1959. Vol. 1. P. 269–271.
7. Levitin A. Introduction to the Design and Analysis of Algorithms. Pearson, 2011. – 600 с.
8. Hamiltonian Circuits [Електронний ресурс] – Режим доступу до ресурсу: [https://math.libretexts.org/Bookshelves/Applied_Mathematics/Book%3A_College_Mathematics_for_Everyday_Life_\(Inigo_et_al\)/06%3A_Graph_Theory/6.04%3A_Hamiltonian_Circuits](https://math.libretexts.org/Bookshelves/Applied_Mathematics/Book%3A_College_Mathematics_for_Everyday_Life_(Inigo_et_al)/06%3A_Graph_Theory/6.04%3A_Hamiltonian_Circuits).
9. Pieńkosz K. Графи і мережі / К. Pieńkosz, J. Wojciechowski. – Варшава: Wydawnictwo Naukowe PWN, 2013.
10. Puzzo L. The Best Map Online and 12 Best Google Maps Alternatives in 2021 [Електронний ресурс] / Lynn Puzzo. – 2021. – Режим доступу до ресурсу: <https://www.mosaic51.com/community/alternatives-to-google-maps/>.

11. Google Maps [Електронний ресурс]. – 2013. – Режим доступу до ресурсу: <https://www.techtarget.com/whatis/definition/Google-Maps>.
12. About us [Електронний ресурс] – Режим доступу до ресурсу: <https://izi.travel/en/about-us/>.
13. About AllTrails [Електронний ресурс] – Режим доступу до ресурсу: <https://www.alltrails.com/about>.
14. Перевалова А. Д. Вебзастосунок «Vikha». Мапа визначних місць з аудіовідтворенням інформації / Перевалова Анастасія Дмитрівна – Житомир: Державний університет «Житомирська політехніка», 2022. – 78 с.
15. Difference between Website and Web Application (Web App) [Електронний ресурс] – Режим доступу до ресурсу: <https://www.guru99.com/difference-web-application-website.html>.
16. Django REST Framework документація [Електронний ресурс] – Режим доступу до ресурсу: <https://www.django-rest-framework.org/>.
17. Model-View-ViewModel (MVVM) [Електронний ресурс] – Режим доступу до ресурсу: <https://www.techtarget.com/whatis/definition/Model-View-ViewModel/>.
18. Hanchett E. Vue.js in Action / E. Hanchett, B. Listwon., 2018. – 375 с.
19. Hillar G. Building RESTful Python Web Services / Gaston C. Hillar., 2016. – 418 с.
20. Django at a glance [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.djangoproject.com/en/5.0/intro/overview/>.
21. Au-Yeung J. Vue.js 3 By Example: Blueprints to learn Vue web development, full-stack development, and cross-platform development quickly / John Au-Yeung., 2021. – 320с.
22. What is Vuex? [Електронний ресурс] – Режим доступу до ресурсу: <https://vuex.vuejs.org/>.
23. Ceder N. The Quick Python Book / Naomi Ceder., 2018. – 472 с.
24. Make a Location-Based Web App With Django and GeoDjango [Електронний ресурс] – Режим доступу до ресурсу: <https://realpython.com/location-based-app-with-geodjango-tutorial/>.

- 25.About PostGIS [Электронный ресурс] – Режим доступа до ресурсу:
<https://postgis.net/>.
- 26.Au-Yeung J. Vue.js 3 By Example: Blueprints to learn Vue web development, full-stack development, and cross-platform development quickly / John Au-Yeung., 2021. – 320с.
- 27.Vite JS - Next Generation Frontend Tool [Электронный ресурс] – Режим доступа до ресурсу: <https://radixweb.com/blog/vite-js-latest-front-end-development-tool>.
- 28.Minn M. Introduction to the Leaflet JavaScript Mapping Library [Электронный ресурс] / Michael Minn – Режим доступа до ресурсу:
<https://michaelminn.net/tutorials/leaflet/index.html>.
- 29.Speech-to-Text [Электронный ресурс] – Режим доступа до ресурсу:
<https://cloud.google.com/speech-to-text/>.
- 30.Lambert R. Mastering PostGIS and OpenStreetMap / Ryan Lambert.
- 31.About Postman [Электронный ресурс] – Режим доступа до ресурсу:
<https://www.postman.com/company/about-postman/>.

ДОДАТКИ

Схема бази даних проекту



Алгоритм знаходження найкоротшого Гамільтонового шляху

```
from itertools import permutations

def is_valid(v, pos, path, graph):
    # Check if the vertex v can be added to the path
    if graph[path[pos - 1]][v] == 0:
        return False

    # Check if the vertex has not already been visited
    if v in path:
        return False

    return True

def calculate_path_weight(path, graph):
    weight = 0
    for i in range(len(path) - 1):
        weight += graph[path[i]][path[i + 1]]
    return weight

def hamiltonian_path_weight_util(graph, path, pos, start_vertex):
    n = len(graph)

    # Base case: If all vertices are included in the path
    if pos == n:
        return True

    # Try different vertices as the next candidate in the path
    for v in range(n):
        if v != start_vertex and is_valid(v, pos, path, graph):
            path[pos] = v

            # Recur to construct the rest of the path
            if hamiltonian_path_weight_util(graph, path, pos + 1, start_vertex):
                return True

    # If adding the current vertex doesn't lead to a solution, backtrack
    path[pos] = -1

    return False

def shortest_hamiltonian_path(graph, start_vertex=0):
    n = len(graph)
```

```

path = [-1] * n

# Start from the first vertex
path[0] = start_vertex

if not hamiltonian_path_weight_util(graph, path, 1, start_vertex):
    return None

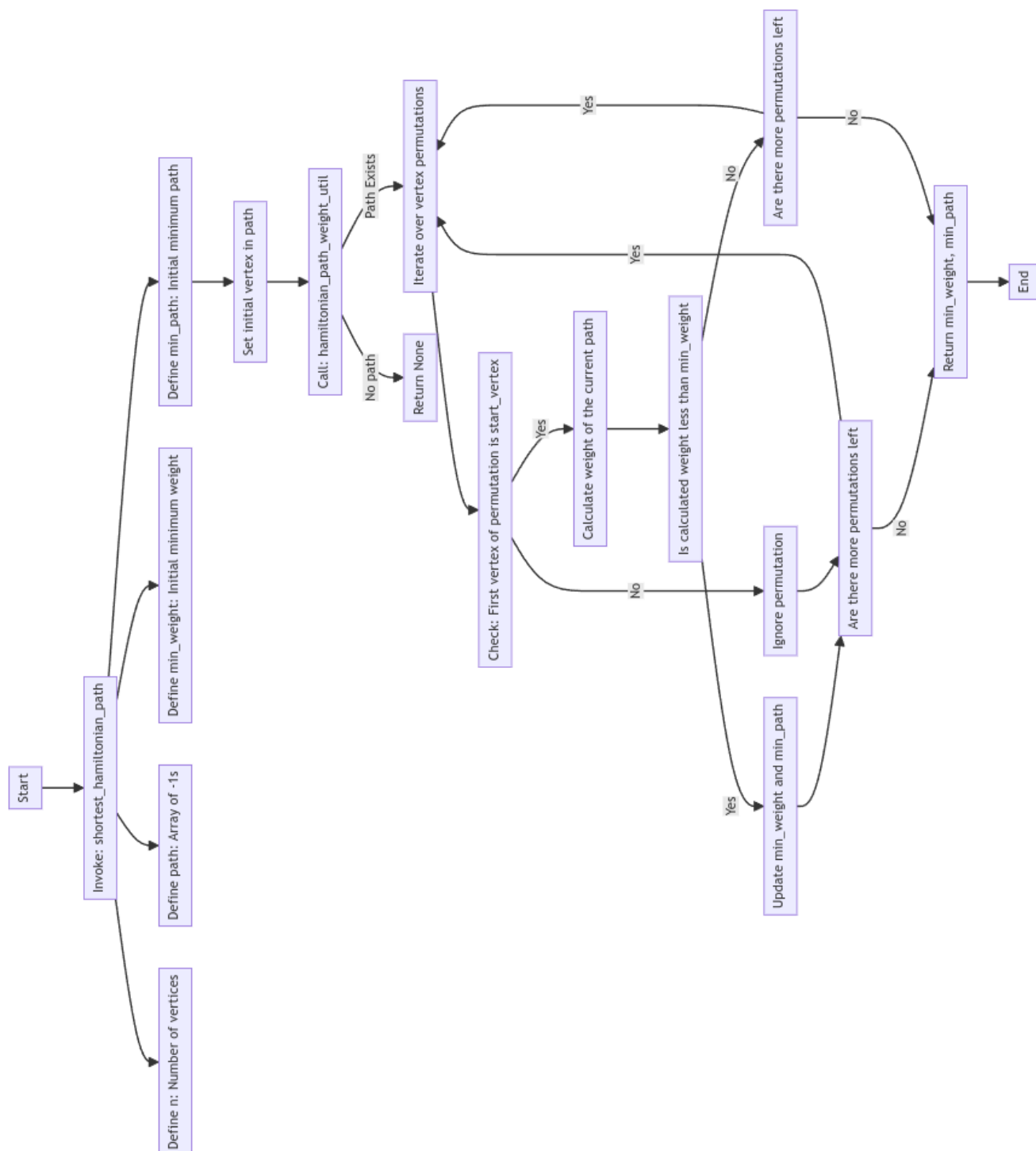
min_weight = float('inf')
min_path = None

# Find the Hamiltonian path with the minimum weight
for p in permutations(range(n)):
    if p[0] == start_vertex:
        weight = calculate_path_weight(p, graph)
        if weight < min_weight:
            min_weight = weight
            min_path = p

return min_weight, min_path

```

Діаграма роботи алгоритму знаходження найкоротшого Гамільтонового шляху



Представлення кінцевої точки GuideView

```

class GuideView(APIView):
    def get(self, request, uuid):
        bookmark_list = get_object_or_404(BookmarkList, uuid=uuid, user=request.user)
        places =
Place.objects.filter(bookmarks__bookmark_list=bookmark_list).order_by("id")

        if not places:
            return self._respond_with_error("No places found related to the provided id.",
status.HTTP_404_NOT_FOUND)

        route_planner = RoutePlanner(places)
        travel_mode = request.query_params.get("travel_mode")
        start_place_uuid = request.query_params.get("start_place_uuid")
        start_place_id = self.get_place_id(start_place_uuid)
        plan_by = self._get_plan_by(request)
        path_result, places_graph = route_planner.get_path_result(start_place_id,
travel_mode, plan_by)

        if not path_result:
            return self._respond_with_error("No Hamiltonian path exists.",
status.HTTP_404_NOT_FOUND)

        response_data = self._prepare_response_data(path_result, places, places_graph)
        return Response(response_data, status=status.HTTP_200_OK)

    @staticmethod
    def get_place_id(place_uuid):
        place = get_object_or_404(Place, uuid=place_uuid)
        return place.id

    @staticmethod
    def _respond_with_error(message: str, http_status: int) -> Response:
        return Response({"message": message}, status=http_status)

    @staticmethod
    def _get_plan_by(request):
        """Returns the query parameter plan_by or distance by default."""
        return request.query_params.get("plan_by", "distance")

    def _prepare_response_data(self, path_result, places, places_graph):

```

```
path_weight, path = path_result
path_details = get_path_details(places_graph, path)
places_order = {index + 1: places[path[index]].uuid for index in range(len(path))}
response_data = {
    "polylines": flatten_coordinates(path_details["geometry"]),
    "places_order": places_order,
    "total_distance": sum(path_details["distance"]),
    "total_duration": sum(path_details["duration"]),
}
return response_data
```

Компонента text-to-speech

```

<template>
  <div
    v-if="audioFile"
    class="row"
  >
    <audio
      autoplay
      controls
      :src="`data:audio/ogg;base64,{audioFile}`"
    >
      {{ $t('audioElementWarning.info') }}
      <code>{{ $t('audioElementWarning.audio') }}</code>
    {{ $t('audioElementWarning.element') }}
    </audio>
  </div>
</template>

<script src="./text-to-speech.js"></script>
<style src="./text-to-speech.scss" lang="scss" scoped></style>

import { mapActions } from 'vuex';
import { useToast } from 'vue-toastification';

export default {
  name: 'TextToSpeech',
  props: {
    inputText: {
      type: String,
      required: true
    },
    language: {
      type: String,
      default: 'uk-UA'
    },
    voiceName: {
      type: String,
      default: 'uk-UA-Standard-A'
    },
    voiceGender: {
      type: String,
      default: 'FEMALE'
    },
    speakingRate: {
      type: Number,
      default: 0.8
    },
  },
  setup () {

```

```

    const toast = useToast();
    return { toast };
  },
  data() {
    return {
      audioFile: null
    };
  },
  mounted() {
    // this.synthesizeText();
  },
  methods: {
    async synthesizeText() {
      const data = {
        'input': {
          'text': this.inputText
        },
        'voice': {
          'languageCode': this.language,
          'name': this.voiceName,
          'ssmlGender': this.voiceGender
        },
        'audioConfig': {
          'audioEncoding': 'OGG_OPUS',
          'speakingRate': this.speakingRate
        }
      };
      try {
        const result = await this.SYNTHESIZE_TEXT(data);
        this.audioFile = result.data.audioContent;
      } catch (e) {
        this.toast.error(this.$t('notifications.errorHasOccurred'));
      }
    },
    ...mapActions([
      'SYNTHESIZE_TEXT',
    ]),
  }
};

audio::-webkit-media-controls-panel {
  background-color: white;
}

```