МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА КАРТОГРАФІЧНОГО ВЕБСЕРВІСУ ВИЗНАЧЕННЯ ОПТИМАЛЬНИХ МАРШРУТІВ»**

Виконав: студент 4 курсу, групи 6.1260
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)
освітньої програми Інформаційні системи та штучний інтелект
(назва освітньої програми)
М.Ю. Назаров
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Мильцев О.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 126 інформаційні системи та технології
(шифр і назва)

Освітня програма інформаційні системи та штучний інтелект

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Назарову Микиті Юрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка картографічного вебсервісу визначення оптимальних маршрутів

керівник роботи Мильцев Олександр Михайлович, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка картографічного вебдодатку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.01.2023	
2.	Збір вихідних даних.	16.01.2024	
3.	Обробка методичних та теоретичних	09.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	25.03.2024	
5.	Розробка третього розділу.	24.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	22.06.2024	

Студент _____ М.Ю. Назаров
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.М. Мильцев
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка картографічного вебсервісу визначення оптимальних маршрутів»: 59 с., 32 рис., 10 джерел.

АВТОЗАПОВНЕННЯ МІСЦЬ, ГЕОКОДУВАННЯ, КАРТОГРАФІЧНИЙ ВЕБСЕРВІС, МАРШРУТИЗАЦІЯ, ОПТИМІЗАЦІЯ МАРШРУТУ.

Об’єкт дослідження: процес розробки та оптимізації картографічного вебсервісу для визначення оптимальних маршрутів.

Мета роботи: розробити та впровадити картографічний вебсервіс для визначення оптимальних маршрутів з використанням сучасних вебтехнологій та API, який забезпечить зручне та ефективне планування маршрутів.

Метод дослідження: аналіз літературних джерел, проєктування архітектури системи, розробка та інтеграція з зовнішніми API, тестування функціональності та оптимізації маршрутів.

Дана кваліфікаційна робота присвячена розробці картографічного вебсервісу для визначення оптимальних маршрутів. Метою роботи є створення системи, яка дозволяє користувачам планувати маршрути з урахуванням параметрів найкоротшого та найшвидшого шляху. В рамках роботи було використано такі технології, як NestJS для бекенду, React для фронтенду, а також сучасні інструменти для роботи з даними та інтерфейсом, такі як TypeScript, Vite, Axios, Tailwind CSS, Immer та Framer Motion.

Аналіз результатів показав, що система успішно оптимізує маршрути для найкоротшого та найшвидшого шляхів, забезпечуючи зручне та ефективне планування подорожей. Робота підтвердила ефективність обраних технологій та підходів, що відкриває можливості для подальшого розвитку та вдосконалення системи.

SUMMARY

Bachelor’s qualifying paper «Development of a Cartographic Web Service for Determining Optimal Routes»: 59 pages, 32 figures, 10 references.

AUTOCOMPLETE PLACES, GEOCODING, CARTOGRAPHIC WEB SERVICE, ROUTING, ROUTE OPTIMIZATION.

The object of the study is the process of developing and optimizing a cartographic web service for determining optimal routes.

The aim of the study is to develop and implement a cartographic web service for determining optimal routes using modern web technologies and APIs, which will provide convenient and efficient route planning.

The method of research is analysis of literature sources, system architecture design, development and integration with external APIs, testing of functionality and route optimization.

This bachelor’s thesis is dedicated to the development of a cartographic web service for determining optimal routes. The objective of the work is to create a system that allows users to plan routes considering the parameters of the shortest and fastest paths. The technologies used in this work include NestJS for the backend, React for the frontend, as well as modern tools for working with data and interface, such as TypeScript, Vite, Axios, Tailwind CSS, Immer, and Framer Motion.

The analysis of the results showed that the system successfully optimizes routes for the shortest and fastest paths, providing convenient and efficient travel planning. The work confirmed the effectiveness of the chosen technologies and approaches, which opens up possibilities for further development and improvement of the system.

ЗМІСТ

ВСТУП

У сучасному світі картографічні вебсервіси відіграють важливу роль у багатьох сферах діяльності, забезпечуючи користувачів зручними та ефективними інструментами для навігації та планування маршрутів. Однією з ключових функцій таких сервісів є визначення оптимальних маршрутів, що дозволяє заощадити час та ресурси. Ця дипломна робота присвячена розробці картографічного вебсервісу для визначення оптимальних маршрутів, який використовує сучасні технології та інструменти для створення надійного та високопродуктивного продукту.

Розробка цього вебсервісу базується на застосуванні наступних технологій та інструментів.

Frontend:

- React – бібліотека для створення користувацьких інтерфейсів, яка забезпечує високу продуктивність та гнучкість у розробці;
- TypeScript (TS) – мова програмування, яка додає типізацію до JavaScript, що дозволяє виявляти помилки на етапі розробки та підвищує надійність коду;
- Vite – швидкий збірник для сучасних вебпроєктів, який значно скорочує час збірки та розробки;
- Axios – бібліотека для виконання HTTP-запитів, яка спрощує взаємодію з API;
- Tailwind CSS – утилітарний CSS-фреймворк, який дозволяє швидко створювати стильні та адаптивні інтерфейси.

Backend:

- NestJS – фреймворк для створення серверних застосунків на Node.js, який забезпечує масштабованість та модульність архітектури;
- TypeORM – ORM (Object-Relational Mapping) для TypeScript та JavaScript, що спрощує роботу з базами даних;

– JWT (JSON Web Token) – стандарт для безпечної передачі інформації між сторонами у вигляді JSON-об’єктів.

API:

– Google Place – API для отримання даних про місця, що дозволяє знайти різноманітні локації за запитом користувача;
– Google Geocode – API для перетворення адрес у географічні координати та навпаки;
– Google OAuth – система аутентифікації, що дозволяє користувачам входити в систему за допомогою облікових записів Google;
– OpenRouteService – сервіс, який надає маршрутизацію та інші картографічні послуги;
– Mapbox GL – API для відображення та взаємодії з інтерактивними картами, що надає широкий спектр можливостей для налаштування та використання карти в вебдодатках.

Основною метою цієї роботи є створення вебсервісу, який забезпечить швидкий та точний розрахунок оптимальних маршрутів з використанням вищезазначених технологій. Реалізація цього проєкту дозволить користувачам ефективніше планувати свої маршрути, що є особливо важливим у великих містах та для підприємств, що займаються логістикою.

У наступних розділах буде детально розглянуто процес розробки вебсервісу, включаючи архітектуру системи, вибір та інтеграцію API, а також тестування.

1 АНАЛІЗ ВИМОГИ ТА ПРОЄКТУВАННЯ СИСТЕМИ

1.1 Вимоги до системи

Щоб краще зрозуміти функціональні вимоги до вебдодатку для визначення оптимальних маршрутів, розглянемо кілька прикладів вимог (див. рис. 1.1).

Вхід користувача.

Поведінка: система дозволить користувачеві увійти в систему за допомогою облікового запису Google, забезпечуючи безпечний доступ до функціоналу додатку.

Вхідні дані: облікові дані користувача (електронна пошта та пароль) через обліковий запис Google.

Процес:

- користувач натискає на кнопку "Увійти за допомогою Google";
- система перенаправляє користувача до сторінки аутентифікації Google;
- користувач вводить свої облікові дані Google та підтверджує вхід;
- google OAuth обробляє аутентифікацію та повертає системі токен доступу;
- система використовує токен доступу для отримання інформації про користувача (ім’я, електронну пошту) від Google;
- система створює новий обліковий запис (якщо користувач реєструється вперше) або оновлює дані існуючого облікового запису;
- користувач отримує доступ до функціоналу додатку.

Вихідні дані:

- доступ до особистого кабінету та інших функцій додатку;
- токен доступу для подальшої аутентифікації під час сесії.

Ця функціональна вимога описує процес входу користувача в систему за допомогою облікового запису Google. Вона включає перенаправлення до сторінки аутентифікації Google, обробку токену доступу та надання доступу до функціоналу додатку після успішної аутентифікації. Такий підхід забезпечує безпечний та зручний спосіб входу для користувачів, використовуючи стандарти безпеки Google OAuth (див. рис. 1.1).

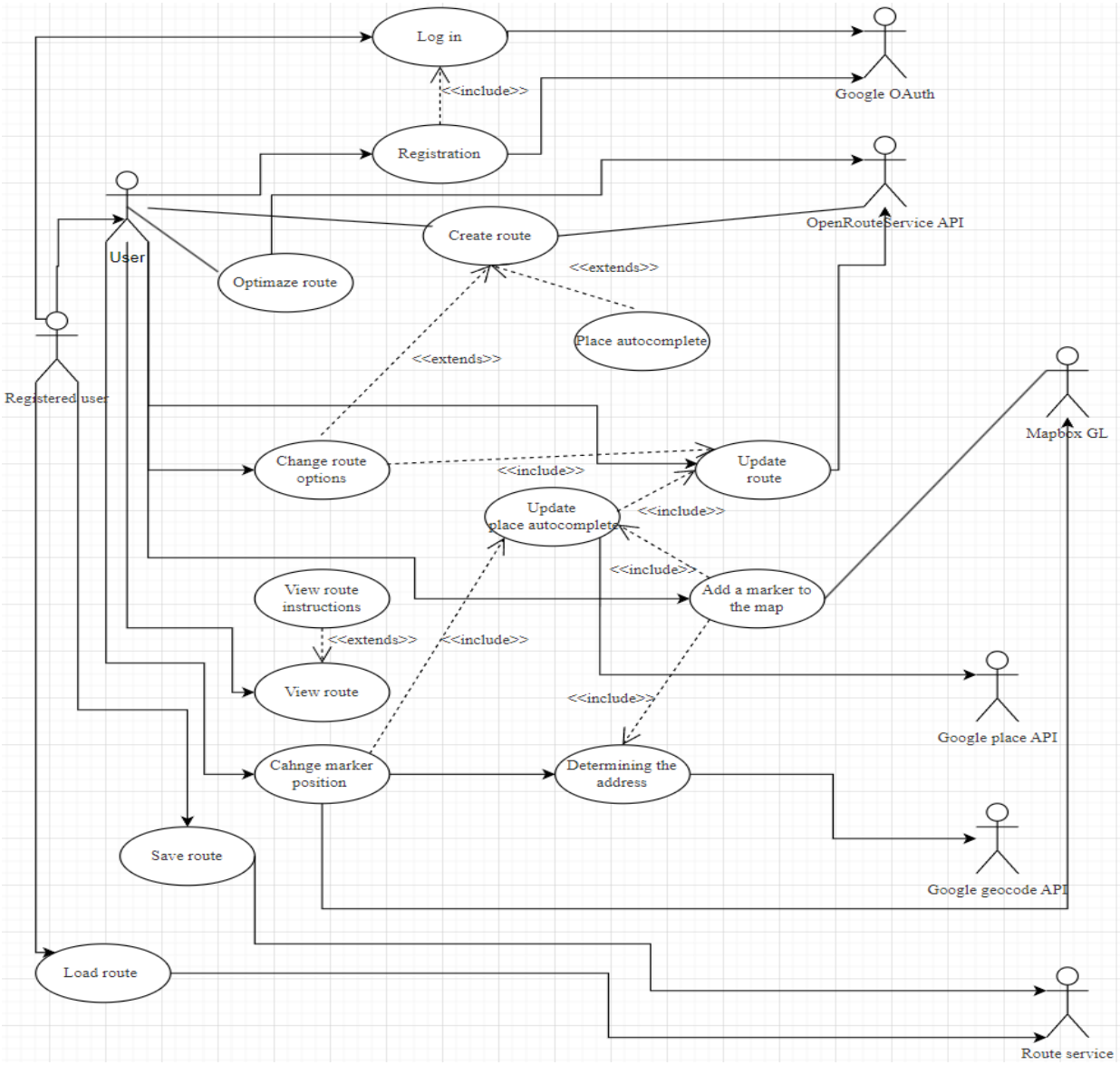

Рисунок 1.1 – Use case diagram

Реєстрація користувача.

Поведінка: система дозволить користувачеві зареєструватися за допомогою облікового запису Google, забезпечуючи безпечний і швидкий

процес створення нового облікового запису.

Вхідні дані: облікові дані користувача (електронна пошта та пароль) через обліковий запис Google.

Процес:

– користувач натискає на кнопку «Зареєструватися за допомогою Google»;
– система перенаправляє користувача до сторінки аутентифікації Google;
– користувач вводить свої облікові дані Google та підтверджує реєстрацію;
– google OAuth обробляє аутентифікацію та повертає системі токен доступу;
– система використовує токен доступу для отримання інформації про користувача (ім’я, електронну пошту) від Google;
– система перевіряє, чи існує користувач у базі даних;
– якщо користувач не існує, система створює новий обліковий запис;
– якщо користувач вже існує, система повідомляє про це користувача;
– після успішної реєстрації користувач отримує доступ до функціоналу додатку.

Вихідні дані:

– підтвердження про успішну реєстрацію;
– доступ до особистого кабінету та інших функцій додатку;
– токен доступу для подальшої аутентифікації під час сесії.

Ця функціональна вимога описує процес реєстрації користувача в системі за допомогою облікового запису Google. Вона включає перенаправлення до сторінки аутентифікації Google, обробку токену доступу, перевірку існування користувача у базі даних та створення нового облікового запису, якщо користувач ще не зареєстрований. Це забезпечує безпечний та зручний спосіб реєстрації для нових користувачів, використовуючи стандарти безпеки Google OAuth (див. рис. 1.1).

Створення маршруту.

Поведінка: система дозволить користувачеві створити новий маршрут, включаючи автозаповнення місць, оптимізацію маршруту, зміну опцій, додавання маркерів та визначення адрес.

Вхідні дані:

– початкова та кінцева точки маршруту;
– проміжні точки (за необхідності);
– параметри оптимізації (наприклад, уникнення платних доріг, вибір типу транспорту тощо);
– параметри часу (час початку або прибуття).

Процес:

– користувач вводить початкову, кінцеву та проміжні точки маршруту;
– використовується Google Place API для автозаповнення місць при введенні адреси [3];
– система визначає адреси введених координат за допомогою Google Geocode API;
– користувач може додавати маркери на карту для позначення проміжних точок маршруту;
– користувач може перетягувати маркери для зміни їх положення на карті;
– система використовує Google Geocode [2] API для оновлення адрес після зміни положення маркерів;
– використовується OpenRouteService [1] API для оптимізації маршруту на основі введених даних;
– користувач може змінювати опції маршруту (тип транспорту, уникнення платних доріг тощо);
– система створює оптимізований маршрут на основі введених даних та налаштувань.

Вихідні дані:

– оптимізований маршрут, який включає найкоротший або

найшвидший шлях, залежно від налаштувань;

– відображення маршруту на інтерактивній карті (Mapbox GL);
– текстові інструкції для маршруту (за необхідності).

Ця функціональна вимога описує процес створення маршруту в системі. Вона включає введення даних користувачем, обробку запитів через Google Place API, Google Geocode API та OpenRouteService API, а також відображення оптимізованого маршруту на інтерактивній карті за допомогою Mapbox GL (див. рис. 1.1).

Оптимізація маршруту.

Поведінка: система дозволить користувачеві оптимізувати маршрут для найкоротшого або найшвидшого шляху з урахуванням заданих параметрів.

Вхідні дані:

– початкова та кінцева точки маршруту;
– проміжні точки (за необхідності);
– опції оптимізації (наприклад, уникнення платних доріг, вибір типу транспорту тощо).

Процес:

– користувач вводить початкову, кінцеву та проміжні точки маршруту;
– користувач налаштовує опції оптимізації маршруту;
– система збирає всі введені дані та надсилає запит до OpenRouteService API для обробки;
– openRouteService API обробляє запит і повертає оптимізований маршрут на основі заданих параметрів.

Вихідні дані:

– оптимізований маршрут, який включає найкоротший або найшвидший шлях, залежно від налаштувань;
– відображення маршруту на інтерактивній карті (Mapbox GL);
– текстові інструкції для маршруту (за необхідності).

Ця функціональна вимога описує процес оптимізації маршруту в системі. Вона включає введення даних користувачем, обробку запиту через

OpenRouteService API та відображення оптимізованого маршруту на інтерактивній карті. Такий підхід забезпечує користувачів точними та ефективними маршрутами відповідно до їхніх потреб та налаштувань (див. рис. 1.1).

Оновлення маршруту.

Поведінка: система дозволить користувачеві оновити існуючий маршрут, включаючи оптимізацію, зміну параметрів та маркерів маршруту;

Вхідні дані:

– ідентифікатор існуючого маршруту;
– початкова та кінцева точки маршруту;
– проміжні точки (за необхідності);
– опції оптимізації (наприклад, уникнення платних доріг, вибір типу транспорту тощо);
– параметри часу (час початку або прибуття);
– позиції маркерів на карті.

Процес:

– користувач вибирає маршрут для оновлення;
– користувач змінює початкову, кінцеву та проміжні точки маршруту;
– користувач налаштовує опції оптимізації маршруту;
– користувач може додавати нові маркери на карту, змінювати їх положення або видаляти;
– система збирає всі введені дані та надсилає запит до OpenRouteService API для обробки;
– openRouteService API обробляє запит і повертає оновлений маршрут на основі заданих параметрів.

Вихідні дані:

– оновлений маршрут, який включає найкоротший або найшвидший шлях, залежно від налаштувань;
– відображення оновленого маршруту на інтерактивній карті (Mapbox GL);

– текстові інструкції для маршруту (за необхідності).

Ця функціональна вимога описує процес оновлення маршруту в системі. Вона включає вибір існуючого маршруту, внесення змін користувачем, обробку запиту через OpenRouteService API та відображення оновленого маршруту на інтерактивній карті за допомогою Mapbox GL [6] (див. рис. 1.1).

Перегляд маршруту.

Поведінка: система дозволить користувачеві переглянути створений або завантажений маршрут на інтерактивній карті, надаючи повну візуалізацію маршруту та можливість взаємодії з ним.

Вхідні дані: ідентифікатор маршруту для перегляду.

Процес:

– користувач вибирає маршрут для перегляду зі списку створених або завантажених маршрутів;
– система завантажує дані маршруту з бази даних або іншого сховища;
– система відображає маршрут на інтерактивній карті, використовуючи Mapbox GL;
– маршрут відображається з усіма точками (початкова, кінцева, проміжні точки) та оптимізованими сегментами;
– користувач може взаємодіяти з картою (масштабування, переміщення, перегляд деталей маршруту).

Вихідні дані:

– відображений маршрут на інтерактивній карті;
– інформація про маршрут, така як відстань, час у дорозі, текстові інструкції тощо (за необхідності).

Ця функціональна вимога описує процес перегляду маршруту в системі. Вона включає вибір маршруту користувачем, завантаження даних маршруту, відображення його на інтерактивній карті за допомогою Mapbox GL та надання користувачеві можливості взаємодії з картою. Вимога наведена на рисунку 1.1.

Додавання маркера на карту та визначення адреси.

Поведінка: система дозволить користувачеві додати маркер на карту

натисканням, автоматично визначаючи та відображаючи адресу для нового положення маркера.

Вхідні дані: координати місця, на яке користувач натиснув на карті.

Процес:

– користувач натискає на карту для додавання маркера;
– система визначає координати точки, де було здійснено натискання;
– система надсилає запит до Google Geocode API для отримання адреси за цими координатами;
– google Geocode API повертає адресу для вказаних координат;
– система додає маркер на карту у визначене місце;
– система відображає визначену адресу у відповідному полі.

Вихідні дані:

– доданий маркер на інтерактивній карті (Mapbox GL);
– визначена адреса для місця додавання маркера.

Ця функціональна вимога описує процес додавання маркера на карту та визначення адреси в системі. Вона включає натискання користувачем на карту для додавання маркера, визначення координат, отримання адреси за допомогою Google Geocode API та відображення маркера разом з адресою на інтерактивній карті за допомогою Mapbox GL (див. рис. 1.1).

Зміна положення маркера.

Поведінка: система дозволить користувачеві змінити положення маркера на карті шляхом перетягування, з автоматичним оновленням адреси для нового положення маркера.

Вхідні дані:

– поточне положення маркера (координати);
– нова позиція маркера (координати після перетягування).

Процес:

– користувач натискає на маркер і перетягує його на нове місце на карті;
– після відпускання маркера система визначає нові координати;

– система надсилає запит до Google Geocode API для отримання адреси нової позиції маркера;
– система оновлює положення маркера на карті та відображає нову адресу.

Вихідні дані:

– оновлене положення маркера на інтерактивній карті (Mapbox GL);
– нова адреса, визначена за новими координатами маркера.

Ця функціональна вимога описує процес зміни положення маркера на карті. Вона включає перетягування маркера користувачем, визначення нових координат та автоматичне оновлення адреси за допомогою Google Geocode API. Це забезпечує користувачів зручним та інтерактивним способом редагування маршрутів на карті, з миттєвим оновленням даних (див. рис. 1.1).

Збереження маршруту.

Поведінка: система дозволить автентифікованому користувачеві зберегти створений або оновлений маршрут для подальшого використання.

Процес:

– користувач натискає на кнопку збереження маршруту;
– система зберігає маршрут в базі даних через Route Service;
– користувач отримує підтвердження про успішне збереження маршруту.

Завантаження маршруту.

Поведінка: система дозволить автентифікованому користувачеві завантажити раніше збережений маршрут для перегляду або редагування.

Процес:

– користувач обирає зі списку збережених маршрутів;
– система завантажує вибраний маршрут з бази даних через Route Service;
– маршрут відображається на карті для перегляду або редагування.

Ці функціональні вимоги конкретно визначають, що повинна робити система. Вони описують основні функції та поведінку системи. Вимоги наведені на рисунку (див. рис. 1.1).

1.2 Концептуальна модель БД

Нижче наведено детальний опис ключових елементів діаграми (див. рис. 1.2).

Концептуальна модель бази даних (БД) – це абстрактне представлення структури бази даних, яке відображає основні сутності, їх атрибути та відношення між ними, але не враховує конкретних технічних деталей реалізації. Основна мета концептуальної моделі – забезпечити чітке та зрозуміле уявлення про дані, що допомагає у їх плануванні, проєктуванні та управлінні.

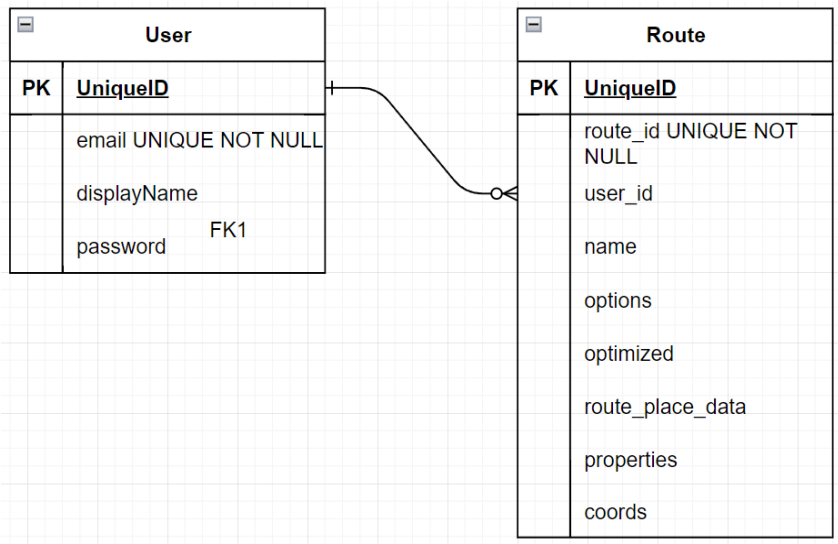

Рисунок 1.2 – ER-діаграма сутностей

Сутність USER. Атрибути:

- ідентифікатор користувача (user_id) – унікальний ідентифікатор для кожного користувача;
- ім’я користувача (displayName) – ім’я, яке використовує користувач;
- електронна пошта (email) – електронна адреса користувача;
- пароль (password) – захищений пароль для входу в систему;
- USER має один або кілька ROUTE. Один користувач може мати багато маршрутів, але кожен маршрут належить тільки одному користувачеві.

Сутність ROUTE. Атрибути:

- ідентифікатор маршруту (route_id) – унікальний ідентифікатор для кожного маршруту;
- назва маршруту (name) – назва маршруту;
- налаштування (options) – налаштування маршруту;
- оптимізація (optimized) – оптимізовано маршрут чи ні;
- адреси або місця (route_place_data) – адреси або місця де проходить маршрут;
- властивості (properties) – властивості маршруту (загальний час, дистанція, інструкції тощо).
- координати (coords) – координати маршруту.

Взаємозв'язки між сутностями.

USER – ROUTE:

- один користувач може створювати багато маршрутів;
- відношення: один до багатьох (один користувач може мати багато маршрутів).

USER є головною сутністю, яка взаємодіє з додатком. Користувач може створювати, оновлювати та видаляти маршрути. Кожен маршрут зберігається у системі та пов'язаний з конкретним користувачем за допомогою унікального ідентифікатора користувача.

ROUTE представляє шлях, який користувач створює у додатку. Маршрут містить початкову, кінцеву та, за необхідності, проміжні точки. Кожен маршрут зберігається з унікальним ідентифікатором, що дозволяє відслідковувати та керувати різними маршрутами.

Діаграма Entity – Relationship для вебдодатку з визначення оптимальних маршрутів включає три основні сутності: користувачі, маршрути та маркери. Користувачі можуть створювати маршрути, кожен з яких може містити багато маркерів для визначення точок на карті. Ця структура забезпечує гнучкість і зручність у створенні, оновленні та управлінні маршрутами, що робить додаток ефективним та корисним для користувачів.

1.3 Аналіз прецедентів

У сучасних вебдодатках, зокрема у тих, що займаються визначенням оптимальних маршрутів, важливою частиною є розробка та аналіз прецедентів. Sequence Diagram (діаграма послідовності) є одним з ключових інструментів моделювання, який дозволяє візуалізувати та деталізувати взаємодію між різними компонентами системи в конкретних сценаріях використання. Цей інструмент допомагає зрозуміти, як різні частини системи взаємодіють одна з одною у часі для досягнення певної мети або виконання завдання.

Діаграма послідовності дозволяє розробникам і аналітикам візуально уявити послідовність повідомлень і дій, які відбуваються між різними учасниками (акторами) системи (див. рис. 1.3).

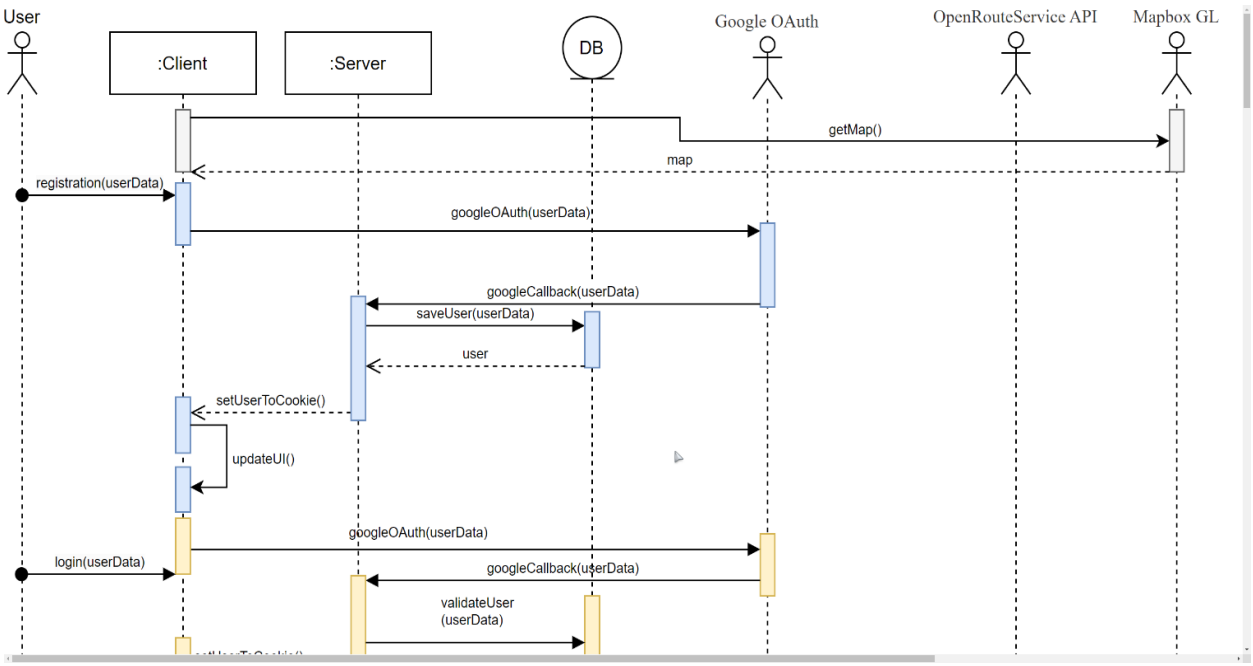

Рисунок 1.3 – Sequence Diagram – registration method

За допомогою Sequence Diagram можна детально проаналізувати взаємодію між користувачем і системою, а також між різними внутрішніми і зовнішніми компонентами системи, такими як API, бази даних та зовнішні сервіси. Діаграма послідовності допомагає виявити і уточнити функціональні

вимоги до системи, а також зрозуміти, як ці вимоги реалізуються на практиці. Візуалізація процесів дозволяє виявити можливі точки оптимізації і покращення взаємодії між компонентами системи.

Далі роздивмось методи, які подані у Sequence Diagram.

Метод **registration(userData)** викликає кілька важливих методів для забезпечення реєстрації користувача.

GoogleOAuth(userData) використовується для автентифікації користувача через Google OAuth. Цей метод ініціює процес автентифікації, перенаправляючи користувача на сторінку входу Google, де користувач вводить свої облікові дані. Після успішного входу Google надає токен доступу, який використовується для отримання даних користувача.

GoogleCallback(userData) обробляє зворотний виклик після автентифікації через Google OAuth. Після того, як користувач успішно автентифікується через Google, Google перенаправляє його назад до системи з токеном доступу. Метод перевіряє валідність токена та витягує необхідні дані користувача.

SaveUser(userData) зберігає дані користувача у базі даних. Після того як дані користувача були отримані та перевірені, цей метод забезпечує збереження їх у відповідній таблиці бази даних для подальшого використання.

SetUserToCookie() встановлює дані користувача у cookie на стороні клієнта. Це необхідно для того, щоб користувач залишався авторизованим у системі при подальших відвідуваннях.

UpdateUI() оновлює інтерфейс користувача після успішної реєстрації. Після того як користувач був успішно зареєстрований та авторизований, цей метод забезпечує оновлення інтерфейсу, щоб відобразити зміни, відображення додаткових функцій.

Метод **login(userData)** використовується для входу користувача в систему (див. рис. 1.4). Призначення цього методу полягає у перевірці облікових даних користувача та наданні доступу до системи. Основні кроки, які виконує цей метод:

– валідація облікових даних: метод перевіряє, чи були введені всі необхідні дані, такі як електронна пошта та пароль;
– виклик validateUser(userData): після початкової валідації, метод викликає validateUser(userData) для перевірки правильності облікових даних користувача у базі даних;
– створення сесії: якщо облікові дані правильні, метод створює сесію для користувача, встановлюючи відповідні дані у кукі або сесії на сервері;
– оновлення інтерфейсу: після успішного входу метод оновлює інтерфейс користувача, наприклад, відображаючи привітання або перенаправляючи на головну сторінку.

Метод **validateUser(userData)** використовується для перевірки правильності облікових даних користувача.

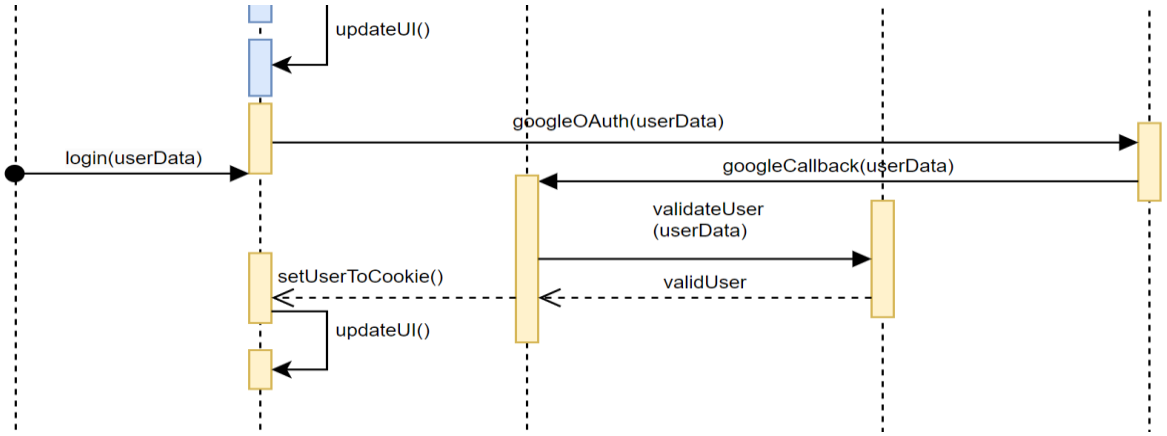

Рисунок 1.4 – Sequence Diagram – login method

Основна мета цього методу полягає у підтвердженні, що користувач існує у базі даних та що введений пароль відповідає збереженому хешованому паролю. Основні кроки, які виконує цей метод (див. рис. 1.4):

– отримання даних з бази: метод звертається до бази даних для отримання запису користувача за введеною електронною поштою;
– перевірка паролю: порівнює введений користувачем пароль з хешованим паролем, що зберігається у базі даних;

– повернення результату: метод повертає результат перевірки, що може бути успішним або неуспішним, залежно від того, чи були облікові дані правильними.

Методи **створення маршруту** (див. рис. 1.5).

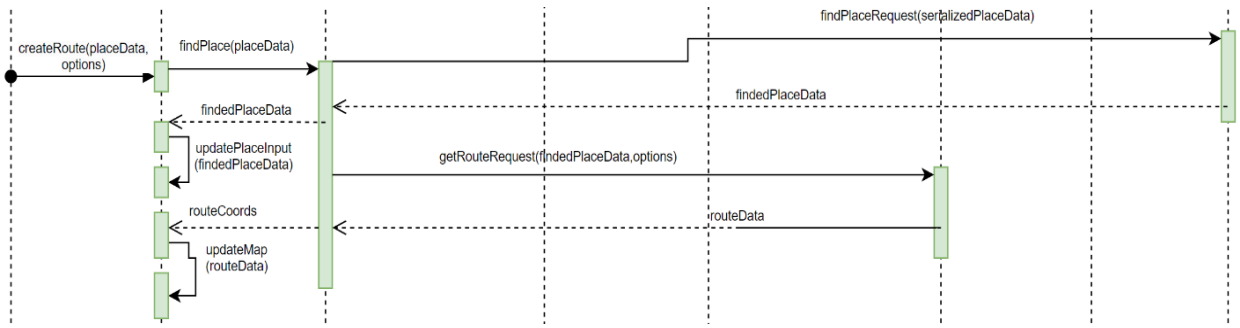

Рисунок 1.5 – Sequence Diagram – createRoute method

CreateRoute(placeData, options) створює маршрут на основі наданих даних про місце та опцій. Він приймає placeData, що містить інформацію про початкову та кінцеву точки маршруту, та options, які визначають параметри маршруту (наприклад, тип транспорту). Метод викликає інші допоміжні методи для отримання даних про місце та побудови маршруту.

FindPlace(placeData) знаходить місце на основі наданих даних про місце (placeData). Він використовує зовнішні API для пошуку місця за назвою, адресую або координатами. Результати пошуку включають детальну інформацію про знайдені місця, такі як назва, адреса та координати.

FindPlaceRequest(serializedPlaceData) відправляє запит на пошук місця, приймаючи серіалізовані дані про місце (serializedPlaceData). Він взаємодіє із зовнішніми сервісами (наприклад, Google Place API) для отримання інформації про місце. Повертає результати пошуку у структурованому форматі.

UpdatePlaceInput(findedPlaceData) оновлює вхідні дані про місце на основі знайденої інформації (findedPlaceData). Він приймає дані про знайдене місце і оновлює відповідні поля у формі вводу користувача. Це забезпечує актуалізацію інформації про місце після виконання пошуку.

GetRouteRequest(findedPlaceData, options) відправляє запит на побудову

маршруту, використовуючи знайдені дані про місце (findedPlaceData) та опції маршруту (options). Він взаємодіє із зовнішніми сервісами, такими як OpenRouteService API, для отримання інформації про маршрут. Повертає дані про маршрут, включаючи шлях, час у дорозі та інші деталі.

UpdateMap(routeData) оновлює карту на основі даних про маршрут (routeData). Він приймає інформацію про маршрут і візуалізує його на карті, використовуючи бібліотеку для роботи з картами, Mapbox GL. Метод також оновлює маркери, лінії маршруту та інші елементи карти для відображення актуальної інформації про маршрут.

Методи **оптимізації маршруту** (див. рис. 1.6).

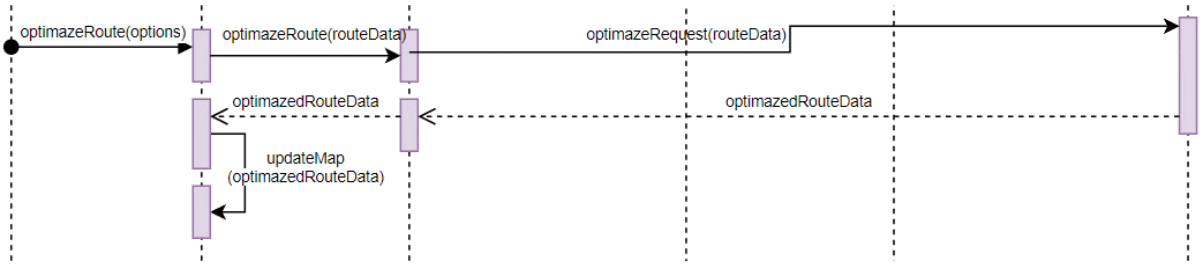

Рисунок 1.6 – Sequence Diagram – optimazeRoute method

OptimizeRoute(options) приймає параметри (опції), які визначають критерії для оптимізації маршруту (наприклад, мінімізація відстані, часу тощо). Використовуючи ці параметри, метод виконує запит для визначення оптимального маршруту.

OptimizeRoute(routeData) приймає дані маршруту, такі як список точок (координат) та інші характеристики маршруту. На основі цих даних виконується оптимізація маршруту для досягнення найбільш ефективного шляху.

OptimizeRequest(routeData) приймає дані маршруту та формує запит до зовнішнього сервісу для оптимізації маршруту. Він включає підготовку та форматування даних, а також відправку запиту до API для отримання оптимізованого маршруту.

UpdateMap(optimizedRouteData) приймає оптимізовані дані маршруту і

оновлює відображення карти. Це включає малювання нового маршруту на карті, оновлення маркерів та інших елементів інтерфейсу, щоб відобразити оптимізований маршрут.

Методи **оновлення маршруту** (див. рис. 1.7).

Рисунок 1.7 – Sequence Diagram – updateRoute method

UpdateRoute(placeData, options) приймає дані про місце (placeData) та опції (options), які визначають критерії для оновлення маршруту. Використовуючи ці параметри, метод оновлює маршрут відповідно до нових даних про місце та заданих параметрів.

FindPlace(placeData) приймає дані про місце (placeData), такі як назва або координати, і здійснює пошук відповідного місця через зовнішній API.

FindPlaceRequest(serializedPlaceData) приймає серіалізовані дані про місце (serializedPlaceData) і формує запит до зовнішнього сервісу для пошуку місця. Він підготовлює та відправляє запит до API, отримуючи дані про знайдене місце.

UpdatePlaceInput(findedPlaceData) приймає дані про знайдене місце (findedPlaceData) і оновлює відповідні поля вводу на інтерфейсі користувача, відображаючи знайдену інформацію.

GetRouteRequest(findedPlaceData, options) приймає дані про знайдене місце (findedPlaceData) та опції (options), які визначають критерії для маршруту. Він формує та відправляє запит до зовнішнього сервісу для отримання маршруту відповідно до заданих параметрів.

UpdateMap(routeData) приймає дані маршруту (routeData) і оновлює відображення карти. Це включає малювання нового маршруту на карті, оновлення маркерів та інших елементів інтерфейсу для відображення оновленого маршруту.

1.4 Висновки до першого розділу

Перший етап розробки картографічного вебсервісу полягав у визначенні вимог до системи. Аналіз вимог включав вивчення потреб користувачів та технічних аспектів для забезпечення надійної та ефективної роботи системи. Основні вимоги були спрямовані на забезпечення зручного та інтуїтивно зрозумілого інтерфейсу, високої продуктивності, безпеки даних та інтеграції з зовнішніми API для отримання геолокаційної інформації та обчислення маршрутів. Ці вимоги стали основою для подальшого проєктування та розробки системи.

На основі визначених вимог було створено концептуальну модель бази даних (БД). Ця модель відображає структуру даних, які необхідно зберігати для забезпечення функціональності вебсервісу. Основними сутностями моделі стали користувачі, маршрути, точки маршрутів та інші пов’язані дані. Концептуальна модель забезпечує логічну організацію даних та їх взаємозв’язки, що є критично важливим для ефективної роботи системи. Модель бази даних була розроблена з урахуванням можливості масштабування та майбутнього розширення функціональності системи.

Аналіз прецедентів включав ідентифікацію ключових сценаріїв використання системи. Було визначено основні випадки використання, такі як реєстрація та вхід користувачів, створення та оптимізація маршрутів, управління маркерами на карті, а також збереження та завантаження маршрутів. Кожен з цих сценаріїв був детально розглянутий для визначення функціональних вимог та необхідних інтеракцій між компонентами системи.

Аналіз прецедентів дозволив створити чітке уявлення про очікувану поведінку системи та взаємодію користувачів з нею.

Перший етап розробки картографічного вебсервісу, який включав аналіз вимог, проєктування концептуальної моделі бази даних та аналіз прецедентів, дозволив закласти міцний фундамент для подальшої реалізації системи. Визначення вимог забезпечило чітке розуміння потреб користувачів та технічних вимог. Концептуальна модель бази даних гарантує ефективну організацію та зберігання даних, а аналіз прецедентів визначив основні сценарії використання системи. Всі ці елементи стали невід’ємною частиною успішного проєктування та розробки картографічного вебсервісу, який забезпечить зручне та ефективне планування маршрутів для користувачів.

2 РЕАЛІЗАЦІЯ СИСТЕМИ

Даний розділ присвячений детальному опису процесу реалізації системи для визначення оптимальних маршрутів, яка використовує сучасні технології та інструменти для забезпечення високої продуктивності та надійності.

Метою цього розділу є надання чіткого та детального опису процесу розробки системи, включаючи вибір технологій, архітектуру системи, реалізацію основних функціональних модулів та інтеграцію з зовнішніми сервісами. У розділі буде висвітлено ключові аспекти розробки, від планування до тестування, з акцентом на практичні рішення та кращі практики, що були використані під час створення системи.

2.1 Реалізація бази даних

У цьому розділі ми детально розглянемо процес налаштування та створення бази даних PostgreSQL для вебдодатку на основі NestJS, використовуючи ORM – бібліотеку TypeORM. NestJS є прогресивним фреймворком для розробки серверних додатків на Node.js, який використовує TypeScript, а TypeORM забезпечує зручний спосіб роботи з базою даних, надаючи можливості для визначення моделей, виконання запитів та керування транзакціями [4]:

- кроки створення бази даних;
- налаштування проєкту NestJS;
- встановлення необхідних пакетів;
- конфігурація підключення до бази даних;
- створення сутностей (Entities);
- створення репозиторіїв;
- міграції бази даних;

– икористання репозиторіїв у сервісах.

Крок 1: налаштування проєкту NestJS. Якщо у вас ще немає проєкту NestJS, створіть його за допомогою Nest CLI.

Крок 2: встановіть необхідні пакети для роботи з PostgreSQL та TypeOR (див. рис. 2.1).

```
npm install  -  - save @nestjs/typeorm typeorm pg
```

Рисунок 2.1 – завантаження пакетів

Крок 3: конфігурація підключення до бази даних. Відредагуйте файл app.module.ts, щоб додати конфігурацію підключення до бази даних (див. рис. 2.2).

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './user/user.entity';
import { UserModule } from './user/user.module';

@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: 'localhost',
      port: 5432,
      username: 'your - username',
      password: 'your - password',
      database: 'your - database',
      entities: [User],
      synchronize: true,
    }),
    UserModule,
  ],
})
export class AppModule {}
```

Рисунок 2.2 – Приклад головного модуля

Далі створіть нову сутність для користувача. Створіть файл user.entity.ts у папці user (див. рис. 2.3).

```
import { Column, Entity, PrimaryGeneratedColumn } from "typeorm";

@Entity({ name: "user" })
export class User {
  @PrimaryGeneratedColumn()
  id: number;
  @Column({ unique: true })
  email: string;
  @Column()
  displayName: string;
  @Column({ default: "", nullable: true })
  password: string;
}
```

Рисунок 2.3 – Створення сутності для користувача

Створіть нову сутність для шляху. Створіть файл route.entity.ts у папці route (див. рис. 2.4).

```
import {
  Column,
  Entity,
  JoinColumn,
  ManyToOne,
  PrimaryGeneratedColumn,
} from "typeorm";
import { User } from "./User";

@Entity({ name: "route" })
export class Route {
  @PrimaryGeneratedColumn()
  id: number;
  @Column({ unique: true })
  route_id: string;
  @Column({ type: "varchar" })
  name: string;
  @Column({ type: "json", nullable: true })
  properties: string;
  @Column({ type: "json", nullable: true })
  coords: string;
  @Column({ type: "json", nullable: true })
  options: string;
  @Column({ type: "boolean" })
  optimized: boolean;
  @Column({ type: "json", default: [] })
  route_place_data: string;
  @ManyToOne(() => User, { onDelete: "CASCADE" })
  @JoinColumn({ name: "user_id", referencedColumnName: "id" })
  user: User;}
```

Рисунок 2.4 – Створення сутності для шляху

Створення файлу user.module.ts у папці user (див. рис. 2.5).

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { UserService } from './user.service';
import { UserRepository } from './user.repository';

@Module({
  imports: [TypeOrmModule.forFeature([UserRepository])],
  providers: [UserService],
  exports: [UserService],
})
export class UserModule {}
```

Рисунок 2.5 – Модуль користувача

Створення файлу сервісу (див. рис. 2.6).

```
@Injectable()
export class RouteService {
  constructor(
    private userService: UserService,
    @InjectRepository(Route) private readonly routeRepository: Repository<Route>
  ) {}

  async createRoute(route: RouteSaveType) {
    try {
      const {
        coordinates,
        id,
        properties,
        name,
        optimized,
        options,
        places,
        userEmail,
      } = route;
```

Рисунок 2.6 – Сервіс користувача

Виконуючи ці кроки, ви зможете налаштувати підключення до бази даних, створити сутності, репозиторії, а також сервіси для взаємодії з базою даних. Це забезпечить зручний і ефективний спосіб роботи з даними у вашому вебдодатку.

2.2 Реалізація бекенда

Бекенд реалізований використовуючи технології фреймворку NestJS. NestJS є прогресивним фреймворком для створення серверних додатків на Node.js, який використовує TypeScript та забезпечує модульну архітектуру для легкої розширюваності та підтримки. Для реалізації бекенду були використані наступні модулі [4]:

– ConfigsModule;
– OpenrouteModule;
– ConfigModule;
– AuthModule;
– GoogleModule;
– RouteModule;
– TypeOrmModule;
– PassportModule;
– UserModule.

ConfigsModule: цей модуль відповідає за зчитування та управління конфігураційними параметрами додатку. Він забезпечує централізоване місце для зберігання конфігурацій, таких як налаштування бази даних, ключі API та інші параметри середовища.

Основні компоненти:

– зчитування конфігурацій з файлів `.env` або інших джерел;
– забезпечення доступу до конфігураційних параметрів для інших модулів додатку.

OpenrouteModule: цей модуль інтегрується з OpenRouteService API для отримання та обробки маршрутних даних. Він надає функціональність для обчислення оптимальних маршрутів на основі введених даних.

Основні компоненти:

– запити до OpenRouteService API для отримання маршрутів;
– обробка та форматування даних маршрутів для використання в додатку.

ConfigModule: подібно до ConfigsModule, цей модуль використовується для управління конфігураціями додатку. Він забезпечує зручний спосіб доступу до налаштувань та параметрів додатку через глобальний модуль.

Основні компоненти:

– зчитування конфігурацій з файлів конфігурацій;
– забезпечення доступу до конфігураційних параметрів у всьому додатку.

AuthModule: модуль відповідає за аутентифікацію та авторизацію користувачів. Він використовує JWT (JSON Web Token) для забезпечення безпечного входу користувачів та управління їхніми сесіями.

Основні компоненти:

– реалізація механізму входу та реєстрації користувачів;
– генерація та валідація JWT токенів;
– захист маршрутів за допомогою авторизації.

GoogleModule: цей модуль інтегрується з Google API для забезпечення функцій аутентифікації користувачів через Google, а також для роботи з сервісами Google, такими як Google Place API та Google Geocode API.

Основні компоненти:

– аутентифікація користувачів через Google OAuth;
– використання Google Place API для автозаповнення місць;
– використання Google Geocode API для визначення адрес за координатами.

RouteModule: модуль відповідає за управління маршрутами користувачів. Він забезпечує функціональність для створення, оновлення, видалення та перегляду маршрутів.

Основні компоненти:

– створення нових маршрутів;
– оновлення існуючих маршрутів;
– видалення маршрутів;
– перегляд деталей маршрутів.

TypeOrmModule: модуль забезпечує інтеграцію з ORM (Object – Relational Mapping) TypeORM, яка використовується для роботи з базою даних. Він дозволяє легко визначати моделі, виконувати запити та керувати транзакціями.

Основні компоненти:

– конфігурація підключення до бази даних PostgreSQL;
– визначення моделей (ентіті) для таблиць бази даних;
– управління міграціями бази даних.

PassportModule: модуль використовується для реалізації стратегій аутентифікації. Він забезпечує інтеграцію з бібліотекою Passport, яка підтримує різні стратегії аутентифікації, включаючи JWT, Google OAuth та інші.

Основні компоненти:

– налаштування стратегій аутентифікації;
– захист маршрутів за допомогою аутентифікації.

UserModule: модуль відповідає за управління користувачами в системі. Він забезпечує функціональність для створення, оновлення, видалення та перегляду користувачів.

Основні компоненти:

– створення нових користувачів;
– оновлення профілів користувачів;
– видалення користувачів;
– перегляд деталей користувачів.

Реалізація бекенду за допомогою NestJS та описаних модулів дозволяє створити структуровану, гнучку та легко масштабовану систему. Кожен модуль відповідає за конкретний аспект функціональності додатку, що забезпечує чіткий поділ обов’язків та полегшує підтримку і розширення системи.

2.3 API реалізація

У процесі розробки вебдодатку для визначення оптимальних маршрутів ми інтегрували декілька зовнішніх API, які забезпечують необхідні дані та функціональність для реалізації ключових можливостей додатку. У цьому розділі ми детально розглянемо реалізацію Google API та OpenRouteService API.

2.3.1 Google API’s

Google API забезпечують потужні інструменти для роботи з геолокаційними даними, аутентифікацією користувачів та іншими сервісами. У нашому додатку ми використовуємо наступні Google API [2].

Google OAuth: використовується для аутентифікації користувачів. Цей сервіс дозволяє користувачам входити в систему за допомогою своїх облікових записів Google, забезпечуючи безпечний та зручний спосіб аутентифікації.

Реалізація:

– налаштування Google OAuth: створіть проєкт в Google Developer Console, налаштуйте OAuth 2.0 Client ID та Client Secret (див. рис. 2.7);
– інтеграція з NestJS: використовуйте пакет @nestjs/passport та passport-google-oauth20 для налаштування стратегії аутентифікації;
– процес аутентифікації: після успішної аутентифікації користувач отримує JWT токен, який використовується для доступу до захищених ресурсів додатку.

Google Place API: забезпечує можливість автозаповнення місць на основі введеного тексту. Це дозволяє користувачам швидко знаходити потрібні місця, вводячи лише частину назви або адреси.

```
@Injectable()
export class GoogleOauthGuard extends AuthGuard("google") {
  async canActivate(context: ExecutionContext) {
    const activate = (await super.canActivate(context)) as boolean;
    const request = context.switchToHttp().getRequest();
    await super.logIn(request);
    return activate;
  }
}
  app.use(
    session({
      secret: process.env.SECRET,
      saveUninitialized: false,
      resave: false,
      cookie: {
        maxAge: 600000,
      },
    })
  );
```

Рисунок 2.7 – Налаштування Google OAuth

Реалізація:

– налаштування API: Отримайте API ключ для Google Place API у Google Cloud Console [2];

– використання в додатку: використовуйте пакет axios або будь-який інший HTTP клієнт для надсилання запитів до Google Place API (запити включають текст для автозаповнення, а відповіді містять список місць, що відповідають запиту);

– запит та опції: запит текстового пошуку – це запит HTTP POST такої форми: https://places.googleapis.com/v1/places:searchText.

Передайте всі параметри в тілі запиту JSON або в заголовках як частину запиту POST. Необхідні параметри: FieldMask. Укажіть список полів для повернення у відповіді, створивши маску поля відповіді. Передайте методу маску поля відповіді за допомогою параметра URL-адреси $fields або fields або за допомогою HTTP – заголовка X – Goog – FieldMask. У відповіді немає списку полів, що повертаються за умовчанням. Якщо ви опустите маску поля, метод повертає помилку. TextQuery. Текстовий рядок, за яким шукати, наприклад: «ресторан», «123 головна вулиця» або «найкраще місце для відвідування в Сан-Франциско».

API повертає збіги кандидатів на основі цього рядка та впорядковує результати на основі їх сприйнятої релевантності (див. рис. 2.8).

```
curl  - X POST  - d '{
  "textQuery" : "Spicy Vegetarian Food in Sydney, Australia"
}' \
 - H 'Content - Type: application/json'  - H 'X - Goog - Api - Key:
API_KEY' \
 - H 'X - Goog - FieldMask:
places.displayName,places.formattedAddress,places.priceLevel' \
'https://places.googleapis.com/v1/places:searchText'
```

Рисунок 2.8 – Запит Google Place API

Google Geocode API: використовується для перетворення адрес у географічні координати (геокодування) та навпаки. Це корисно для визначення точного розташування місць за адресами.

Реалізація:

– налаштування API: отримайте API ключ для Google Geocode API у Google Cloud Console;
– використання в додатку: використовуйте пакет axios або будь-який інший HTTP клієнт для надсилання запитів до Google Geocode API (запити можуть містити адресу для геокодування або координати для зворотного геокодування, а відповіді містять відповідні координати або адреси (див. рис. 2.9));
– запит та опції: у наступному прикладі запитуються широта та довгота “1600 Amphitheatre Parkway, Mountain View, CA” та вказується, що вихідні дані мають бути у форматі JSON. https://maps.googleapis.com/maps/api/geocode/json?address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&key=YOUR_API_KEY
адреса – вулиця або плюс – код, який потрібно геокодувати.

Вказуйте адреси відповідно до формату, який використовується національною поштовою службою відповідної країни. Слід уникати додаткових елементів адреси, таких як назви підприємств і номери підрозділів, апартаментів або поверхів. Елементи адреси повинні бути розділені пробілами

```
@Post("geocodeLatLng")
  async geocode(
    @Res() res: Response,
    @Body() body: { value: [lat: number, lng: number] }
  ) {
    const latlng = body.value.join(",");
    try {
      const url =
`https://maps.googleapis.com/maps/api/geocode/json?latlng=${latlng}&key=${
apiKey}&enable_address_descriptor=true`;

      const response = await fetch(url);
      if (!response.ok) {
        throw new Error("Bad request to geocode");
      }
      const data = await response.json();

      return data;
    } catch (error) {
      return res.status(500).json({ error: "Internal server error" });
    }
  }
```

Рисунок 2.9 – Ендпоинт Google Geocode API

2.3.2 OpenRouteService API

OpenRouteService API надає потужні інструменти для роботи з маршрутними даними, включаючи побудову та оптимізацію маршрутів. У нашому додатку ми використовуємо цей API для обчислення оптимальних маршрутів [1].

OpenRouteService API: використовується для обчислення маршрутів між заданими точками, включаючи опції оптимізації, такі як уникнення платних доріг, вибір типу транспорту тощо (див. рис. 2.10).

Реалізація:

– налаштування API: зареєструйтеся на платформі OpenRouteService та отримайте API ключ;

– використання в додатку: використовуйте пакет axios або будь-який інший HTTP клієнт для надсилання запитів до OpenRouteService API (запити включають координати початкової та кінцевої точок, а також додаткові параметри маршруту);

– запит та опції: POST /v2/directions/{profile}/geojson. Параметр параметрів є об’єкт JSON, можна встановити кілька параметрів, згаданих тут: options.avoid_features, options.profile_params, options.vehicle_type, attributes, coordinates, alternative_routes, continue_straight, instructions, units, maximum_speed;
– обробка відповіді: відповіді від OpenRouteService API містять деталі маршруту, включаючи інструкції для руху, відстань та час у дорозі (ці дані використовуються для відображення маршруту в інтерфейсі додатку).

```
async fetchOpenRouteRoute({
  coordinates,
  routeOptions,
}: {
  coordinates: string | [number, number][];
  routeOptions: string;
  body?: any;
}) {
  try {
    const optionsData: RouteOptions = JSON.parse(routeOptions);

    let coordsOpenRoute;
    if (Array.isArray(coordinates)) {
      coordsOpenRoute = coordinates;
    } else {
      coordsOpenRoute = JSON.parse(coordinates);
    }
    const { avoid_features, ...options } = optionsData;
    const jsonCoords = JSON.stringify({
      coordinates: coordsOpenRoute,
      options: { avoid_features },
      ...options,
    });
    const responseOpenRoute = await fetch(
      "https://api.openrouteservice.org/v2/directions/driving - car/geojson",
      { headers, method: "POST", body: jsonCoords }
    );
    const data = (await responseOpenRoute.json()) as any;

    const coordsOpenRouteData = data.features;
    const routeName = generateRouteName(coordsOpenRouteData);

    if (data.error) {
      return { error: data.error };
    }
    return { coordsOpenRouteData, optionsData, routeName };
  } catch (error) {
    throw new Response("Error", { status: 500 });
  }
}
```

Рисунок 2.10 – Ендпоинт OpenRouteService API

2.4 Реалізація клієнту

Реалізація клієнтської частини вебдодатку є критично важливою для забезпечення зручного та інтуїтивно зрозумілого інтерфейсу користувача. У цьому розділі ми розглянемо процес розробки фронтенду для додатку визначення оптимальних маршрутів, який було створено за допомогою сучасних вебтехнологій. Основним завданням було забезпечити високу продуктивність, адаптивність та зручність використання додатку [5].

2.4.1 Компоненти

Для реалізації клієнтської частини були використані наступні технології та інструменти:

- React: бібліотека для створення користувацьких інтерфейсів;
- TypeScript (TS): мова програмування, яка додає типізацію до JavaScript, що дозволяє виявляти помилки на етапі розробки та підвищує надійність коду;
- Vite: цвидкий збірник для сучасних вебпроєктів, який значно скорочує час збірки та розробки;
- Axios: бібліотека для виконання HTTP-запитів, яка спрощує взаємодію з API;
- Tailwind CSS: утилітарний CSS-фреймворк, який дозволяє швидко створювати стильні та адаптивні інтерфейси.

Основні компоненти клієнтської частини. Компонент MapInstance відповідає за відображення інтерактивної карти з можливістю додавання маркерів, визначення маршрутів та взаємодії з користувачем. Він використовує контекст для управління станом карти, API OpenRouteService для обчислення маршрутів та Mapbox GL [6] для відображення карти (див. рис. 2.11).

```
<Map
  onDblClick={handleDoubleClick}
  onClick={handleClick}
  mapboxAccessToken={import.meta.env.VITE_ACCESS_TOKEN}
  mapLib={import("mapbox - gl")}
  initialViewState={{
    longitude: import.meta.env.VITE_LONGITUDE || 35.11697906675258,
    latitude: import.meta.env.VITE_LATITUDE || 47.84785262713706,
    zoom: import.meta.env.VITE_ZOOM || 12,
  }}
```

Рисунок 2.11 – Map component

Основні частини компоненту:

– контекст карти (useMapContext): контекст карти забезпечує глобальний стан для компонента MapInstance, включаючи маркери, маршрути та інші параметри карти (контекст дозволяє легко керувати станом карти та передавати дані між компонентами);
– стан компоненту (useState): компонент використовує useState для управління локальним станом, таким як інформація про наведення, ідентифікатори маршрутів та точок, стан перетягування маркерів, маршрути та інструкції маршруту.

Компонент MapProvider є контекстним провайдером для управління глобальним станом карти у вашому додатку. Він забезпечує контекст для компонентів, які потребують доступу до стану карти та дій для його зміни. Цей компонент використовує бібліотеку use-immer для спрощення роботи з незмінним станом (див. рис. 2.12).

Основні частини компоненту:

– ініціалізація стану: компонент починається з визначення початкового стану initialState, який включає координати, маркери, центр карти та місця (start і stop);
– використання редуктора (useImmerReducer): для управління станом використовується useImmerReducer, який забезпечує незмінний стан;
– періодичний виклик функції healthCheck:Перший useEffect викликає функцію healthCheck кожну секунду, якщо карта не завантажується

(state.mapLoading);

– аутентифікація користувача через Cookies:Другий useEffect зчитує access_token з Cookies, декодує його та встановлює користувача у стані за допомогою dispatch;
– забезпечення контексту для дочірніх компонентів: компонент повертає MapContext.Provider, який надає стан та метод dispatch для всіх дочірніх компонентів.

```
export const MapContext =
createContext<MapContextType>(initialState);
const MapProvider: FC<MapProviderProps> = ({ children }) => {
  const [state, dispatch] = useImmerReducer(reducer,
initialState.state);
  useEffect(() => {
    const fetchData = async () => {
      const data = await healthCheck();
      dispatch({
        type: "SET_MAP_LOADING",
        mapLoading: data === "bad" ? false : true,
      });
    };
```

Рисунок 2.12 – Map provider component

Компонент ControlPanel є інтерактивною панеллю керування для карти, яка забезпечує користувачам можливість взаємодії з різними елементами, такими як маршрути, налаштування та інструкції. Він використовує контекст карти для доступу до стану та управління ним, а також бібліотеки motion для анімації та react-map-gl для роботи з картою (див. рис. 2.13).

Основні частини компоненту:

– ініціалізація стану компоненту: використовується useState для управління локальним станом компоненту, таким як відкриття поповеру, стан меню та налаштування сортування;
– функції обробки подій: функція handleToggleMenu обробляє зміну стану меню;
– відображення панелі керування та її елементів: основна частина компонента відповідає за відображення панелі керування та її

елементів, включаючи меню, збережені маршрути, інструкції для маршрутів та інші налаштування.

```
          <div>
            <ToggleButton />
            <motion.div
              className={`z - 20 absolute  scrollbar - thumb - zinc - 800
scrollbar - track - zinc - 900    w - full sm:max - w - [500px] min - w -
[300px]    left - 0 top - 0  `}
              animate={{
                overflowY: "hidden",
                scrollBehavior: "auto",
                scrollbarWidth: "thin",
              }}
              transition={{ duration: 5 }}
            >
              <motion.div
                className=" rounded - br - sm  overflow - y - auto scrollbar -
thin
                sm:max - h - screen"
                variants={itemVariants}
                animate={toggleMenu ? "open" : "closed"}
              >
```

Рисунок 2.13 – Control panel component

2.4.2 Технології

Immer [7] – це бібліотека для роботи з незмінними станами у JavaScript. Вона дозволяє легко працювати з незмінними об’єктами, забезпечуючи простий та інтуїтивно зрозумілий синтаксис для оновлення стану.

Переваги:

– простий синтаксис для оновлення стану, що нагадує роботу з мутабельними об’єктами;
– автоматичне створення нових версій стану при його зміні, забезпечуючи незмінність;
– зниження кількості помилок, пов’язаних з мутабельністю стану.

Використання у проєкті: Immer використовується для управління станом у контексті MapContext, що забезпечує зручну роботу з незмінними станами додатку

Framer Motion [8] – це бібліотека для створення анімацій у React додатках. Вона надає простий та потужний API для створення плавних та інтерактивних анімацій.

Переваги:

– простий у використанні API для створення анімацій.;
– підтримка складних анімацій та інтерактивності;
– висока продуктивність завдяки використанню оптимізованих алгоритмів анімації.

Використання у проєкті: Framer Motion використовується для анімації компонентів панелі керування та інтерфейсу карти, забезпечуючи плавний та привабливий користувацький досвід.

Tailwind CSS [9] – це утилітарний CSS-фреймворк, який дозволяє швидко створювати стильні та адаптивні інтерфейси. Він надає набір класів, які можна використовувати безпосередньо у HTML для застосування стилів.

Переваги:

– висока продуктивність розробки завдяки використанню готових класів;
– гнучкість та можливість кастомізації;
– підтримка адаптивного дизайну та мобільних пристроїв;
– використання у проєкті: Tailwind CSS використовується для стилізації компонентів інтерфейсу, забезпечуючи швидку та ефективну розробку стильних та адаптивних інтерфейсів.

Axios – це бібліотека для виконання HTTP-запитів у JavaScript. Вона забезпечує зручний API для взаємодії з сервером, підтримуючи проміси та асинхронні операції.

Переваги:

– простий та інтуїтивно зрозумілий API для виконання HTTP-запитів;
– підтримка обробки відповідей та помилок;
– можливість налаштування інтерцепторів для обробки запитів та відповідей.

Використання у проєкті: Axios використовується для виконання запитів до серверу та зовнішніх API, таких як OpenRouteService та Google API, забезпечуючи взаємодію між фронтендом та бекендом.

Vite – це швидкий збірник для сучасних вебпроєктів. Він забезпечує блискавичну швидкість збірки та розробки завдяки використанню сучасних технологій, таких як ESBuild.

Переваги:

– швидка збірка та запуск проєкту;
– миттєве оновлення модулів (HMR) для підвищення продуктивності розробки;
– підтримка сучасних функцій JavaScript та CSS.

Використання у проєкті: Vite використовується для збірки та розвитку вебдодатку, забезпечуючи високу продуктивність та швидкість розробки.

2.5 Висновки до другого розділу

Реалізація бази даних базувалася на розробленій концептуальній моделі, яка забезпечила логічну структуру даних для вебсервісу. Було використано PostgreSQL у поєднанні з ORM – бібліотекою TypeORM, що дозволило ефективно управляти схемою бази даних та виконувати операції з даними. Вибір цієї технології забезпечив масштабованість та гнучкість у подальшому розширенні функціональності системи.

Бекенд був реалізований за допомогою фреймворку NestJS, який забезпечує модульну архітектуру та підтримує використання TypeScript для забезпечення надійності коду. Основні модулі, такі як ConfigModule, OpenrouteModule, AuthModule, RouteModule та UserModule, забезпечили необхідну функціональність для обробки запитів користувачів, аутентифікації, зберігання та обробки даних маршрутів.

Важливою частиною системи стала інтеграція з зовнішніми API, що

забезпечило доступ до необхідної інформації та сервісів для обчислення маршрутів та автозаповнення місць. Зокрема, були використані Google API та OpenRouteService API.

Google Place API дозволяє користувачам швидко знаходити потрібні місця, вводячи лише частину назви або адреси. Інтеграція з цим API забезпечила зручний інтерфейс для пошуку місць та автозаповнення.

Google Geocode API використовується для перетворення адрес у географічні координати та навпаки. Це дозволило визначати точні місця на карті за введеними адресами, що забезпечило точність у плануванні маршрутів.

OpenRouteService API використовується для обчислення та оптимізації маршрутів. Цей API забезпечив можливість врахування різних параметрів, таких як найкоротший та найшвидший шлях, що дозволило користувачам отримувати оптимальні маршрути відповідно до їхніх потреб.

Клієнтська частина вебсервісу була реалізована з використанням сучасних вебтехнологій, що забезпечило зручний та інтуїтивно зрозумілий інтерфейс для користувачів. Використання React дозволило створити компонентний підхід, що полегшує підтримку та розширення функціональності.

Було створено різні компоненти для забезпечення функціональності системи, включаючи компоненти для реєстрації та входу користувачів, створення та управління маршрутами, відображення карти та маркерів, а також налаштування опцій маршрутів.

Для забезпечення високої продуктивності та зручності у використанні були використані наступні технології:

– Immer для роботи з незмінними станами;
– Framer Motion для створення плавних та привабливих анімацій;
– Tailwind CSS для швидкої та ефективної розробки стильних та адаптивних інтерфейсів;
– Axios для виконання HTTP-запитів до серверу та зовнішніх API [10];

– Vite для забезпечення швидкої збірки та запуску проєкту [5].

Реалізація системи включала в себе створення бази даних, розробку бекенду та інтеграцію з зовнішніми API, а також розробку клієнтської частини з використанням сучасних вебтехнологій. Всі ці етапи були виконані успішно, що дозволило створити ефективний та зручний вебсервіс для визначення оптимальних маршрутів. Використання сучасних технологій та підходів забезпечило високу продуктивність, гнучкість та масштабованість системи, що відкриває можливості для подальшого розвитку та вдосконалення.

3 ТЕСТУВАННЯ СИСТЕМИ

Тестування системи є критичними етапами у забезпеченні її надійності, стабільності та відповідності вимогам користувачів. У цьому розділі ми розглянемо основні функції системи, які необхідно протестувати для гарантування коректної роботи вебдодатку.

3.1 Основні функції системи

Реєстрація користувача: користувач повинен мати можливість зареєструватися в системі за допомогою облікового запису Google. Для користувачів юез облікового запису є можливість створити запис сервісу для подальшого використання (див. рис. 3.1).

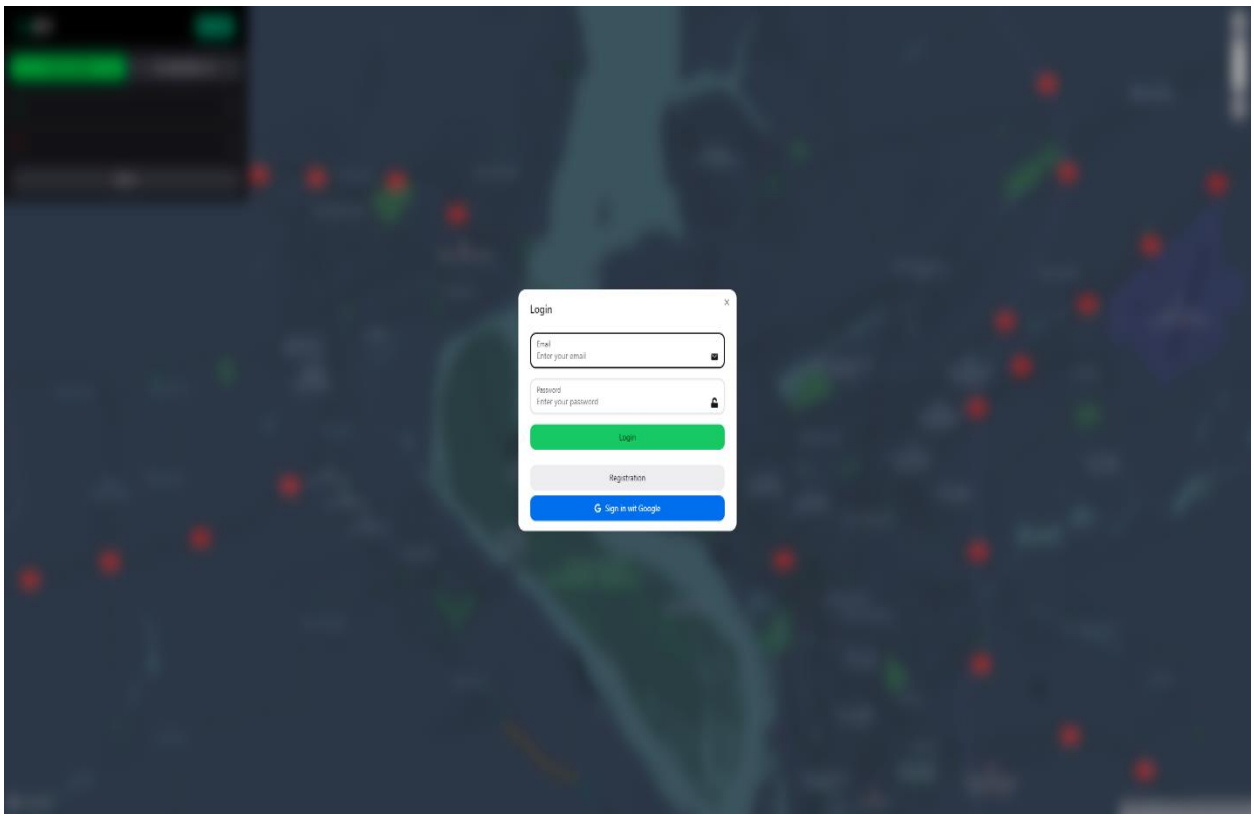

Рисунок 3.1 – Реєстрація користувача Google

Надайте доступ до вашого облікового запису (див. рис. 3.2).

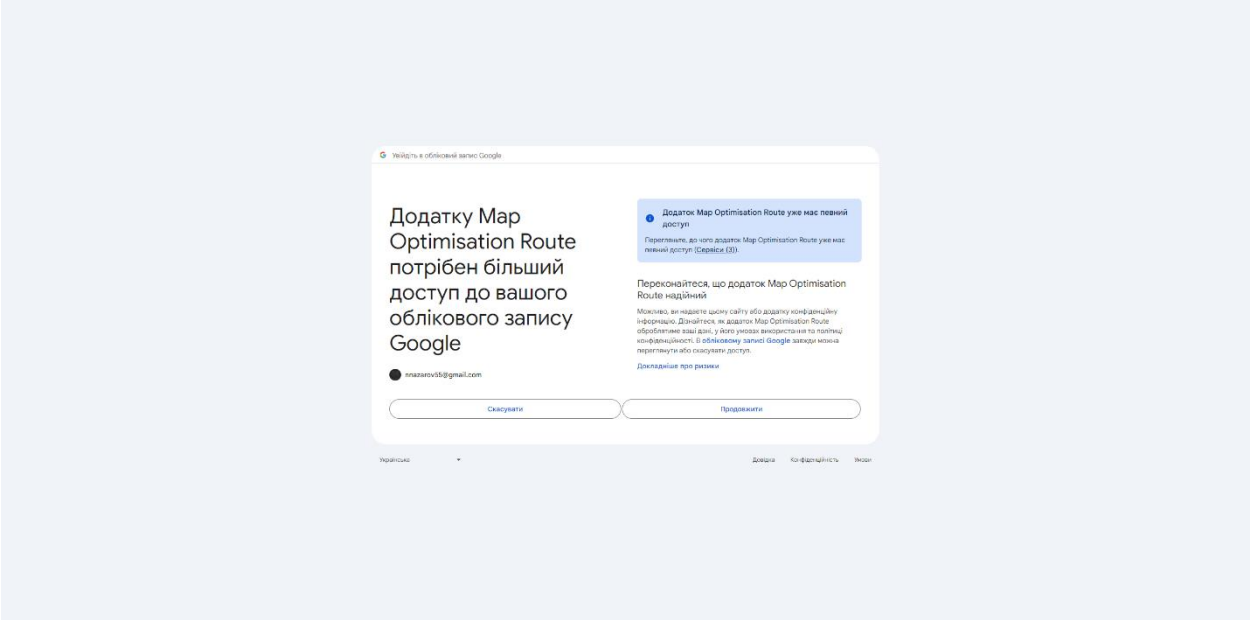

Рисунок 3.2 – Вхід до аккаунта Google

Ваш обліковий запис буде відображений на панелі керування (див. рис. 3.3).

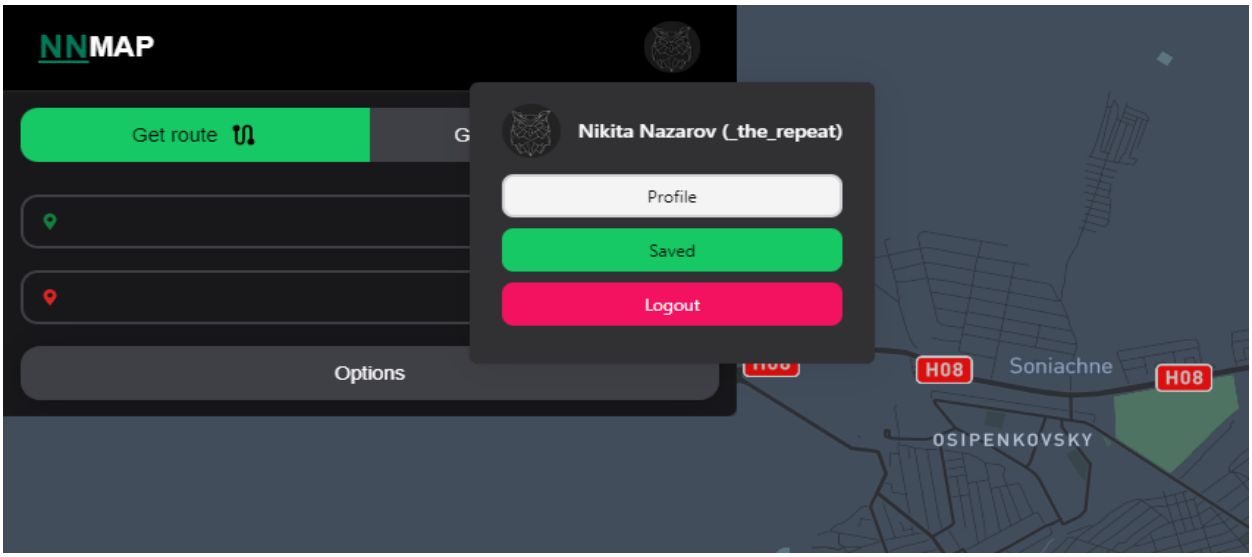

Рисунок 3.3 – Аккаунт Google

Створення маршруту: користувач повинен мати можливість створити маршрут, задавши початкову та кінцеву точки. Система повинна

використовувати OpenRouteService API для обчислення маршруту. Маршрут повинен бути відображений на інтерактивній карті (див. рис. 3.4).

Додавання маркера на карту: користувач повинен мати можливість додати маркер на карту, натиснувши на неї. Система повинна визначити та відобразити адресу для нового положення маркера за допомогою Google Geocode API (див. рис. 3.5).

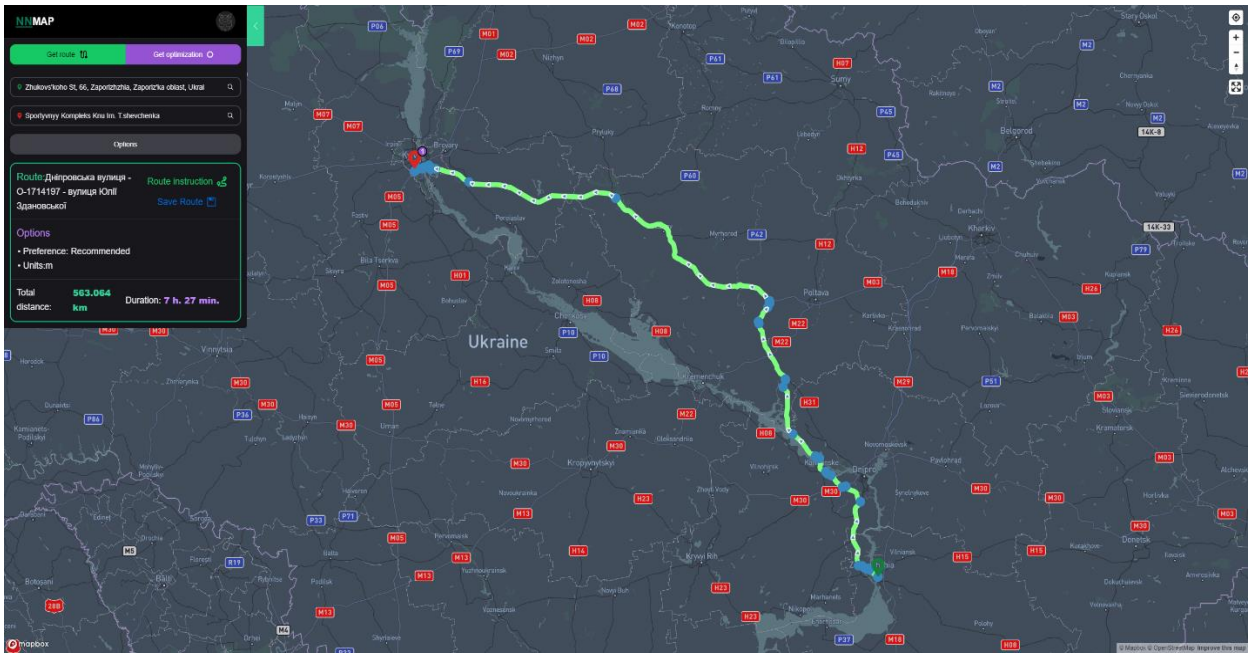

Рисунок 3.4 – Шлях на мапі

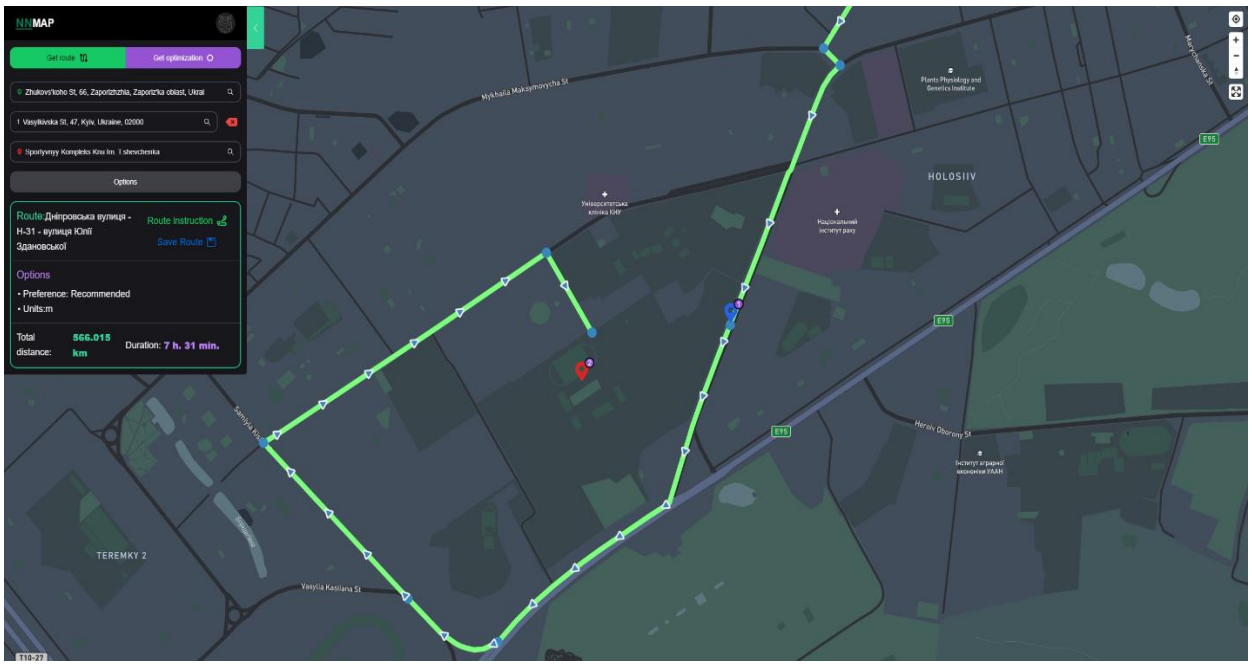

Рисунок 3.5 – Доданий маркер

Налаштування опцій маршруту: користувач повинен мати можливість налаштувати опції маршруту (див. рис. 3.6).

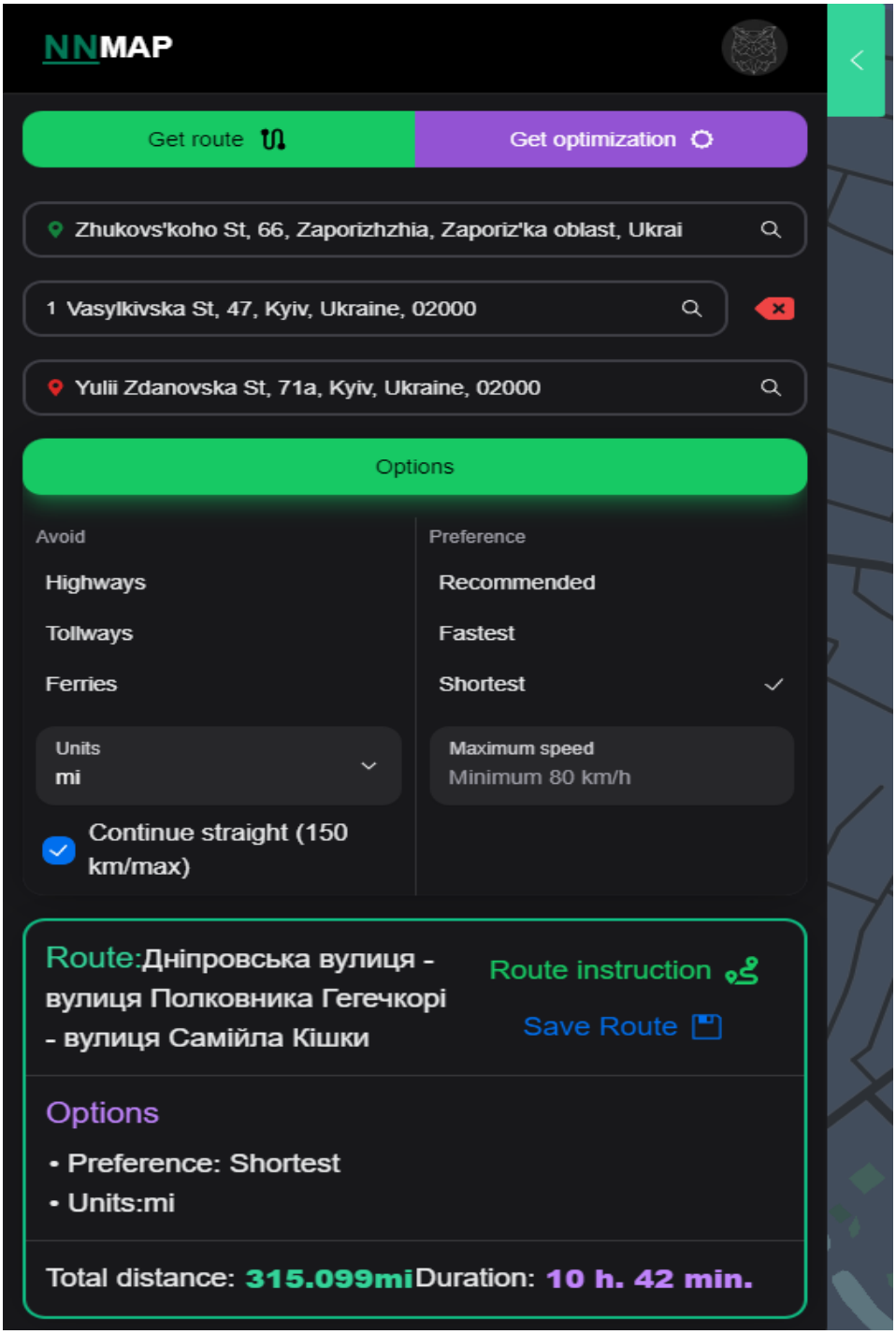

Рисунок 3.6 – Опції маршруту

Оптимізація маршруту: система повинна оптимізувати маршрут для найкоротшого або найшвидшого шляху з урахуванням заданих параметрів (див. рис. 3.7).

Рисунок 3.7 – Неоптимізований шлях

Шлях складеться з 7 пунктів:

– Sobornyi Ave, 175, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69000;
– 87B, Peremohy St, 87Б, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69000;
– Mayakovs’koho Ave, 24A, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69061;
– Peremohy St, 90, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69061;
– Maydan Profspilok, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69061;
– Myru St, 3, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69000;
– Patriotychna St, 80, 60, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69061.

Інформація до шляху:

– загальна довжин неоптимізованого шляху складає:13.207 км;
– загальна тривалість неоптимізованого шляху складає: 0 г. 13хв.;
– налаштування: найшвидший шлях;
– шлях після проведення оптимізації (див. рис. 3.8);
– загальна довжин оптимізованого шляху складає:10.967 км;
– загальна тривалість оптимізованого шляху складає: 0 г. 15хв.;
– налаштування: найшвидший шлях (див. рис. 3.9).

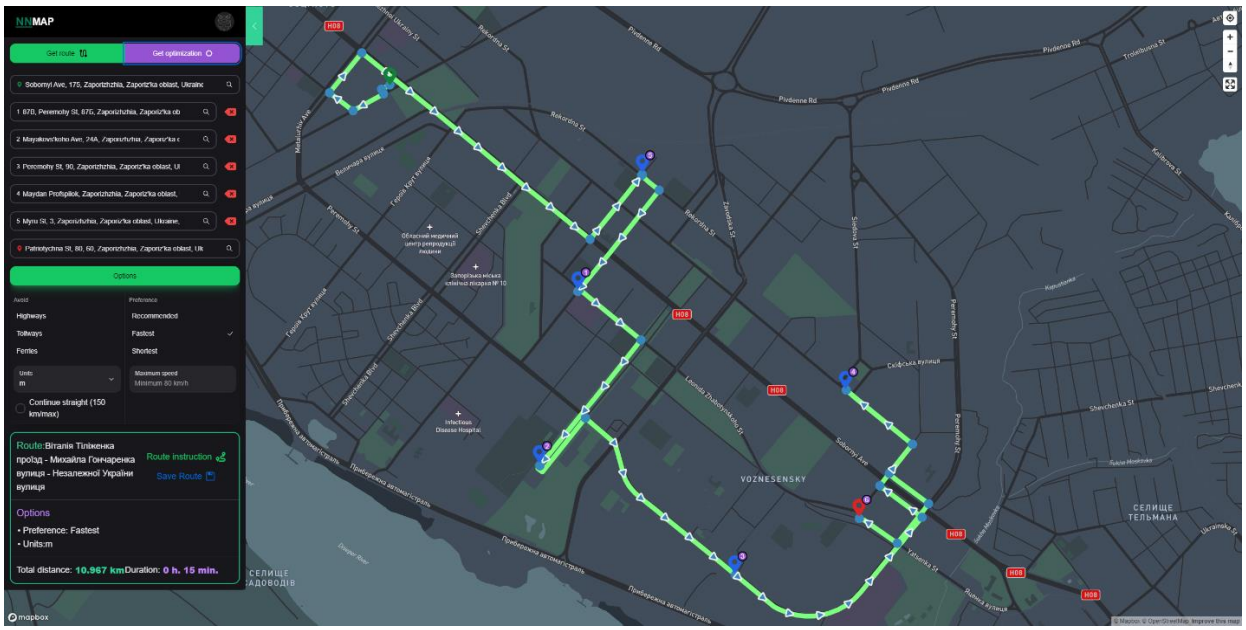

Рисунок 3.8 – Найшвидшій оптимізований шлях

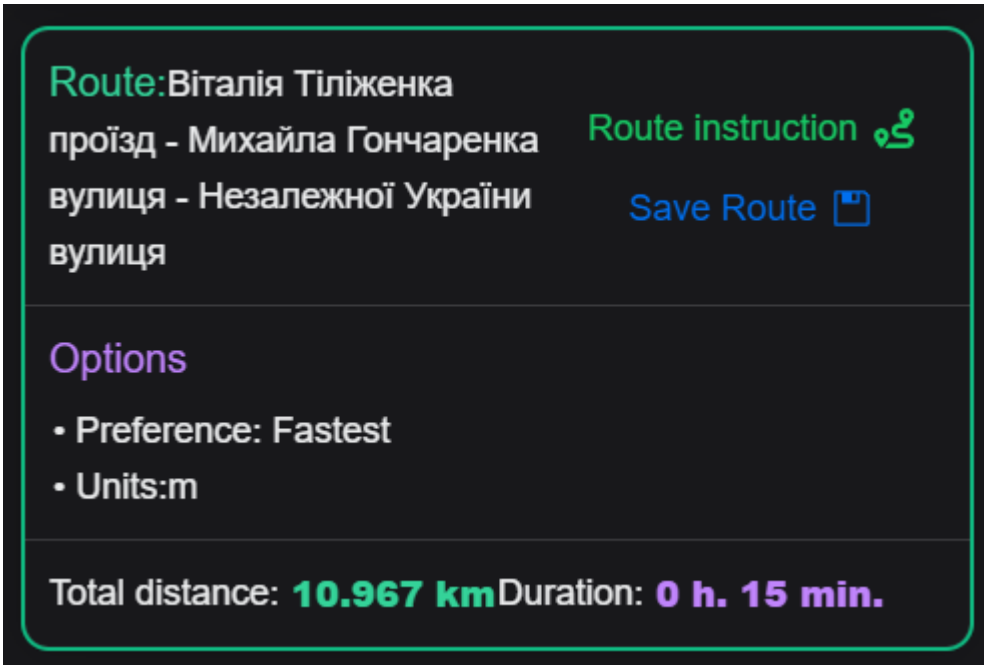

Рисунок 3.9 – Дані оптимізованого найшвидшого шляху

Шлях після проведення оптимізації з налаштуванням «найкоротший шлях» (див. рис. 3.10):

– загальна довжин оптимізованого шляху складає:10.331 км;
– загальна тривалість оптимізованого шляху складає: 0 г. 24хв.;
– налаштування: найкоротший шлях (див. рис. 3.11).

Рисунок 3.10 – Найкоротший оптимізований шлях

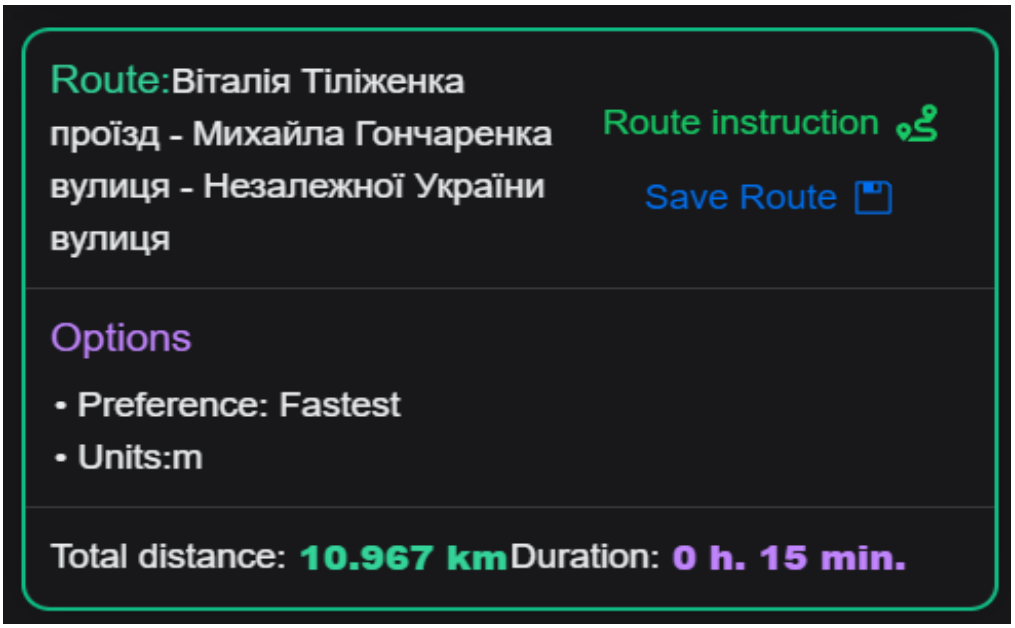

Рисунок 3.11 – Дані найкоротшого оптимізованого шляху

Система успішно оптимізує маршрут для найкоротшого або найшвидшого шляху, враховуючи задані параметри. Аналіз проведених тестів демонструє ефективність використання різних налаштувань оптимізації маршруту. Нижче наведено детальний аналіз результатів для різних сценаріїв оптимізації.

Налаштування: **найшвидший шлях**.

Шлях складається з 7 пунктів:

– Sobornyi Ave, 175, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69000;
– 87В, Peremohy St, 87Б, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69000;
– Mayakovs’koho Ave, 24A, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69061;
– Peremohy St, 90, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69061;
– Maydan Profspilok, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69061;
– Myru St, 3, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69000;
– Patriotychna St, 80, 60, Zaporizhzhia, Zaporiz’ka oblast, Ukraine, 69061.

Неоптимізований шлях:

– загальна довжина: 13.207 км;
– загальна тривалість: 0 год. 13 хв.

Оптимізований шлях (найшвидший шлях):

– загальна довжина: 10.967 км;
– загальна тривалість: 0 год. 15 хв.

Після оптимізації для найшвидшого шляху загальна довжина маршруту скоротилася на 2.240 км (17%). Однак загальна тривалість маршруту збільшилася на 2 хвилини. Це свідчить про те, що система змогла знайти більш ефективний маршрут, який зменшив загальну відстань, але враховує дорожні умови, що вплинули на загальну тривалість подорожі.

Налаштування: **найкоротший шлях:**

– оптимізований шлях (найкоротший шлях);
– загальна довжина: 10.331 км;
– загальна тривалість: 0 год. 24 хв.

Після оптимізації для найкоротшого шляху загальна довжина маршруту скоротилася ще більше на 0.636 км (5.8%) у порівнянні з оптимізованим маршрутом для найшвидшого шляху. Однак загальна тривалість маршруту значно збільшилася на 9 хвилин. Це вказує на те, що система успішно знайшла найкоротший можливий шлях, але за рахунок збільшення часу в дорозі через менш ефективні дорожні умови.

Збереження та завантаження маршруту представлено на рисунку 3.12.

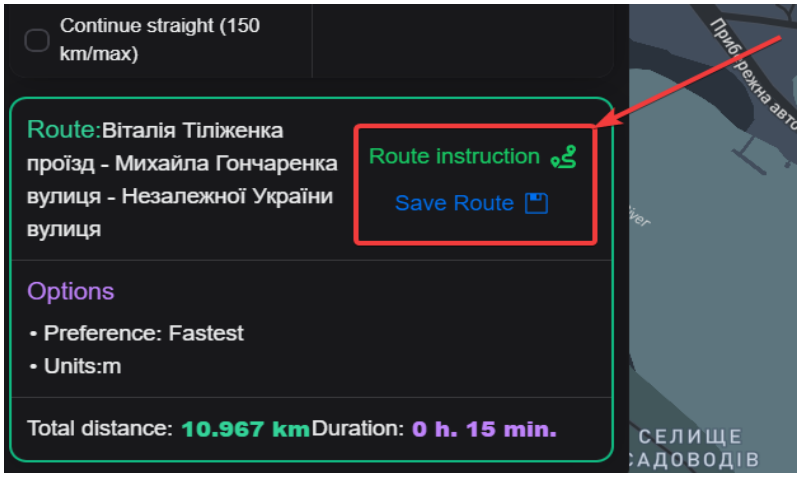

Рисунок 3.12 – Збереження маршруту

3.2 Висновки до третього розділу

Система ефективно оптимізує маршрут відповідно до заданих параметрів. Оптимізація для найшвидшого шляху скорочує загальну довжину маршруту, забезпечуючи баланс між відстанню та тривалістю подорожі. Оптимізація для найкоротшого шляху мінімізує загальну відстань, але може збільшити час подорожі через менш оптимальні дорожні умови. Ці результати підтверджують, що система здатна адаптувати маршрути до потреб користувачів, забезпечуючи гнучкість та ефективність у плануванні подорожей.

Тестування системи показало, що всі основні функції працюють коректно та відповідають вимогам користувачів. Система успішно інтегрується з зовнішніми API, забезпечуючи необхідну функціональність для планування та оптимізації маршрутів. Виявлені під час тестування недоліки були виправлені, що дозволило досягти високої якості програмного забезпечення. Завдяки проведеному тестуванню та валідації система готова до використання та забезпечує зручний і ефективний інструмент для користувачів.

ВИСНОВКИ

У цій роботі було розроблено картографічний вебсервіс для визначення оптимальних маршрутів з використанням сучасних вебтехнологій та API. Система дозволяє користувачам ефективно планувати маршрути, забезпечуючи можливість вибору найкоротшого або найшвидшого шляху. Нижче наведено основні висновки, отримані під час виконання роботи.

Система була реалізована на основі фреймворку NestJS для бекенду та React для фронтенду. Використання TypeScript забезпечило статичну типізацію, що підвищило надійність та стабільність коду. Були використані такі основні модулі:

- ConfigsModule: забезпечує централізоване управління конфігураційними параметрами;
- OpenrouteModule: інтеграція з OpenRouteService API для обчислення маршрутів;
- AuthModule: аутентифікація користувачів за допомогою Google OAuth;
- RouteModule: управління маршрутами користувачів;
- UserModule: управління інформацією про користувачів.

Для клієнтської частини були використані такі сучасні технології, як Immer, Framer Motion, Tailwind CSS, Axios та Vite, що забезпечило високу продуктивність та зручність у використанні.

Інтеграція з зовнішніми API. Було успішно інтегровано декілька зовнішніх API:

- Google OAuth: для аутентифікації користувачів;
- Google Place API: для автозаповнення місць на основі введеного тексту;
- Google Geocode API: для визначення адрес за координатами;
- OpenRouteService API: для обчислення та оптимізації маршрутів.

Інтеграція з цими API забезпечила систему необхідними даними та функціональністю для ефективної роботи.

Система була ретельно протестована на відповідність основним функціям:

– реєстрація та вхід користувачів;
– створення та управління маршрутами;
– додавання та зміна маркерів на карті;
– оптимізація маршрутів для найшвидшого та найкоротшого шляху.

Тестування показало, що система успішно оптимізує маршрути відповідно до заданих параметрів. При виборі найшвидшого шляху система скорочує загальну довжину маршруту, збільшуючи тривалість подорожі незначно. При виборі найкоротшого шляху система мінімізує відстань, але значно збільшує час подорожі. Аналіз результатів оптимізації маршрутів показав:

– найшвидший шлях: загальна довжина маршруту скорочується на 17%, а тривалість подорожі збільшується на 2 хвилини;
– найкоротший шлях: загальна довжина маршруту скорочується ще більше на 5.8%, але тривалість подорожі збільшується на 9 хвилин.

Це свідчить про те, що система ефективно адаптує маршрути до потреб користувачів, забезпечуючи баланс між відстанню та часом подорожі.

Розробка картографічного вебсервісу для визначення оптимальних маршрутів виявилася успішною. Система забезпечує користувачам зручний інтерфейс для планування маршрутів, використовуючи сучасні вебтехнології та зовнішні API. Оптимізація маршрутів відповідно до заданих параметрів демонструє гнучкість та ефективність системи, що робить її корисним інструментом для повсякденного використання. Виконана робота підтвердила ефективність обраних технологій та підходів, а також відкрила можливості для подальшого розвитку та вдосконалення системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. OpenRouteService API Documentation. URL: https://api.openrouteservice.org/ (дата звернення: 04.03.2024).
2. Google Maps Geocoding API Documentation. URL: https://developers.google.com/maps/documentation/geocoding/overview (дата звернення: 29.03.2024).
3. Google Maps Places API Documentation. URL: https://developers.google.com/maps/documentation/places/web-service (дата звернення: 11.03.2024).
4. NestJS Documentation. URL: https://nestjs.com/ (дата звернення: 22.03.2024).
5. Vite Guide. URL: https://vitejs.dev/guide/ (дата звернення: 04.03.2024).
6. Mapbox Maps API Documentation. URL: https://docs.mapbox.com/api/maps/ (дата звернення: 04.03.2024).
7. Immer. URL: https://immerjs.github.io/immer/ (дата звернення: 29.04.2024).
8. Framer motion: https://www.framer.com/motion/ (дата звернення: 01.04.2024).
9. Tailwind. URL: https://tailwindcss.com/docs/ (дата звернення: 01.04.2024).
10. Axios. URL: https://axios-http.com/docs/intro (дата звернення: 03.03.2024).