Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)
Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Веб-сервіс для відстеження руху міського транспорту
(тема)

Виконав:
Здобувач 4 року навчання,
групи ПЗПІз-21-1
Дмитро ГУЗЄЄВ
(власне ім'я, прізвище)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Наталя КРАВЕЦЬ
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачу Гузєєву Дмитру Михайловичу
(прізвище, ім'я, по батькові)

1. Тема роботи Веб-сервіс для відстеження руху міського транспорту
Затверджена наказом по університету від 05.05.2025р. №74Стз
2. Термін подання студентом роботи до екзаменаційної комісії 06.06.2025
3. Вихідні дані до роботи Розробити веб-сервіс для відстеження руху міського транспорту, яка дозволяє користувачу визначати найближчі зупинки, можливі маршрути, приблизний час прибуття транспорту, і конструювати зв'язуючи маршруту у випадках, де прямого нема.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	20.04.2025	виконано
2	Створення специфікації ПЗ	22.04.2025	виконано
3	Проектування ПЗ	24.04.2025	виконано
4	Розробка ПЗ	31.05.2025	виконано
5	Тестування ПЗ	02.05.2025	виконано
6	Оформлення пояснювальної записки	02.05.2025	виконано
7	Створення заяви щодо самостійного виконання кваліфікаційної роботи	04.06.2025	виконано
8	Створення матеріалів для комп'ютерного захисту	07.06.2025	виконано
9	Перевірка записки керівником	08.06.2025	виконано
10	Перевірка на плагіат	09.06.2025	виконано
11	Нормоконтроль, рецензування	11.06.2025	виконано
12	Оцінка роботи рецензентом	11.06.2025	виконано
13	Отримання відгуку від керівника	11.06.2025	виконано
14	Здача роботи в електронний архів кафедри	11.06.2025	виконано
15	Захист кваліфікаційної роботи	16.06.2025	виконано

Дата видачі завдання 10 лютого 2025р.

Здобувач_____
(підпис)

Керівник роботи _____ доц. кафедри ПІ Наталя КРАВЕЦЬ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 100 стор., 18 рис., 20 джерел.

ВЕБ-СЕРВІС ДЛЯ ВІДСТЕЖЕННЯ РУХУ МІСЬКОГО ТРАНСПОРТУ, PYTHON, TYPESCRIPT, REACT

Об’єкт розробки – веб-сервіс для відстеження руху міського транспорту.

Мета роботи – створення масштабованого програмного застосунку для відстеження руху міського транспорту з можливістю інтеграції різних GTFS-систем та надання користувачам зручного інструменту для планування оптимальних маршрутів пересування містом.

Метод рішення – Python, fastapi, TypeScript, React, Next.js.

У результаті розробки створено програмний застосунок, який дозволяє користувачам відстежувати рух міського транспорту, і будувати маршрути між точками на карті.

ABSTRACT

WEB-APPLICATION FOR TRACKING URBAN TRANSPORT MOVEMENT, PYTHON, TYPESCRIPT, REACT

The object of development is a software application for tracking urban transport movement.

The purpose of development is to create a scalable software system for tracking urban transport with the ability to integrate various GTFS systems and provide users with a convenient tool for planning optimal routes for moving around the city.

The solution method is Python, fastapi, TypeScript, React, Next.js.

As a result of the development, a software application has been created that allows users to track urban transport movement and build routes between points on the map.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

HTTP – Hypertext Transfer Protocol

JSON – JavaScript Object Notation

RPS – Requests per Second

API – Application Programming Interface

AWS – Amazon Web Services

WS – Web Sockets

GTFS – General Transit Feed Specification

RDBMS – Relational Database Management System

ORM – Object-Relational Mapping

AVL - Automated Vehicle Location

SIRI - Service Interface for Real Time Information

NeTEx - Network Timetable Exchange

SSR – Server-Side Rendering

ВСТУП

Актуальність розробки системи відстеження руху громадського транспорту обумовлена декількома важливими факторами сучасного міського життя. В умовах постійного зростання населення міст та збільшення транспортних потоків, ефективна навігація стає критично важливою для покращення якості життя громадян. Традиційні статичні розклади та схеми маршрутів не задовольняють сучасні потреби користувачів через їх низьку адаптивність до змін у русі транспорту, відсутність оперативних даних та неможливість побудови індивідуальних маршрутів.

Глобальне впровадження стандарту GTFS (General Transit Feed Specification) дозволяє уніфікувати представлення даних про громадський транспорт, що відкриває широкі можливості для інтеграції різних транспортних систем в єдиний інформаційний простір. Проте існуючі рішення часто обмежуються окремими містами або транспортними компаніями, що створює фрагментацію користувацького досвіду та ускладнює планування поїздок.

Вирішення цих проблем через створення масштабованої системи з уніфікованим інтерфейсом та можливістю підключення різноманітних GTFS-джерел є актуальним завданням, що відповідає сучасним тенденціям розвитку міської інфраструктури та концепції "розумного міста".

Мета роботи полягає у створенні масштабованого програмного застосунку для відстеження руху міського транспорту з можливістю інтеграції різних GTFS-систем та надання користувачам зручного інструменту для планування оптимальних маршрутів пересування містом.

Завдання роботи:

– аналіз існуючих систем відстеження громадського транспорту та виявлення їх переваг і недоліків;

– розробка архітектури програмного застосунку, що забезпечує масштабованість та ефективну роботу з великими обсягами даних;

– проектування та реалізація модуля імпорту і обробки даних з GTFS-фідів різних транспортних систем;

– розробка алгоритму побудови оптимальних маршрутів з урахуванням можливих пересадок та часових параметрів;

– створення зручного користувацького інтерфейсу для відображення інформації про зупинки, маршрути та розклади руху транспорту;

– реалізація адміністративного інтерфейсу для керування підключеними GTFS-системами;

– тестування системи на продуктивність та стабільність при високих навантаженнях.

Розроблений програмний застосунок може бути застосований у таких сферах:

– повсякденне використання мешканцями міст для оптимізації щоденних маршрутів та економії часу на пересування;

– туристична галузь – для зручної навігації у незнайомих містах туристами без необхідності встановлення різних застосунків для кожного міста;

– міські адміністрації та транспортні компанії – для підвищення якості обслуговування пасажирів та моніторингу ефективності транспортної інфраструктури;

– дослідження транспортних потоків – для аналізу завантаженості маршрутів та оптимізації транспортної мережі міста;

– інтеграція з іншими міськими сервісами в рамках концепції "розумного міста" для створення єдиного інформаційного простору міської інфраструктури.

Практична цінність роботи полягає у створенні універсального рішення, яке може бути швидко адаптоване до потреб різних міст без необхідності розробки окремих систем для кожного з них, що значно скорочує витрати на впровадження подібних систем та підвищує зручність користування громадським транспортом для населення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Темою кваліфікаційної роботи є програмний застосунок для відстеження руху міського транспорту. Цей програмний застосунок має допомагати користувачам у повсякденному житті зручно і швидко дізнаватися де є найближчі зупинки, які види транспорту до них ходять, які маршрути існують між цими зупинками, та давати можливість побудувати маршрут до пункту призначення, враховуючи усі можливі пересадки, та оптимізуючи у відношенні до часу. Система має надавати адміністраторам можливість підключати додаткові GTFS системи, з яких моментально будуть зчитуватись дані, які потім будуть доступні для користувачів. Система має бути масштабуємою до великої кількості підключених GTFS систем, і великої кількості користувачів. Додаток може бути застосовано для будування маршрутів у невідомих містах, для оптимізації повсякденних маршрутів.

Сучасні міста активно впроваджують цифрові технології для покращення роботи громадського транспорту. У багатьох країнах існують різні додатки, інтегровані в систему міського транспорту, що допомагають пасажирам планувати маршрути, відстежувати транспорт у реальному часі та мінімізувати час очікування.

Відстеження руху громадського транспорту є важливою частиною концепції "розумного міста" (Smart City), що спрямована на підвищення ефективності міської інфраструктури через використання інформаційних технологій та аналізу даних. Автоматизовані системи відстеження транспортних засобів (Automated Vehicle Location, AVL) [1] використовують комбінацію GPS-технологій, бездротового зв'язку та серверів обробки даних для надання інформації пасажирам та керування трафіком.

Одним із найпопулярніших сервісів у світі є Google Maps [2], який дозволяє будувати маршрути для пішоходів, водіїв та пасажирів громадського транспорту. Він використовує великі масиви даних та алгоритми машинного навчання для

прогнозування часу прибуття транспорту та оптимізації маршрутів. В Україні найбільш популярним додатком є EasyWay [3] (також відомий як E Way), який забезпечує покриття багатьох міст та надає інформацію про маршрути громадського транспорту.

Основною проблемою реалізації таких додатків є доступ до джерел інформації та API, які надають актуальні дані про рух міського транспорту. У світі найпоширенішим стандартом для обміну такими даними є GTFS (General Transit Feed Specification), розроблений компанією Google [4]. Завдяки цьому формату транспортні агенції можуть надавати інформацію про маршрути, розклади та позицію транспорту.

GTFS складається з двох компонентів:

– GTFS Static - набір текстових файлів CSV формату з фіксованою структурою, що містять інформацію про маршрути, зупинки, розклади тощо;

– GTFS Realtime - розширення, що дозволяє надавати дані про поточний стан транспортної системи в режимі реального часу через Protocol Buffers.

На жаль, в Україні лише Львів, Маріуполь і Чорноморськ офіційно надають свої дані у форматі GTFS. Інші міста використовують власні рішення або не мають публічного API, що ускладнює інтеграцію систем відстеження. Проте GTFS є основним пропонованим стандартом порталом українського міністерства цифрового розвитку, що свідчить про перспективу його ширшого впровадження в українських містах.

GTFS є стандартом комунікації між сервісами, які надають такий сервіс як відстеження руху міського транспорту, тому побудувати архітектуру можна і без конкретних джерел даних. Замість цього можна зробити інтеграцію сервісів, які працюють за цим стандартом, легкою та масштабованою.

Також варто зазначити, що крім GTFS, у світі використовуються й інші стандарти для обміну даними про громадський транспорт, такі як:

– SIRI (Service Interface for Real Time Information) [5] - європейський стандарт, що забезпечує широкий набір сервісів для обміну інформацією про громадський транспорт;

– NeTEx (Network Timetable Exchange) [6] - стандарт для обміну статичними даними про транспортну мережу та розклади;

– TransXChange [7] - британський національний стандарт для обміну даними про розклади автобусів.

1.2 Аналіз конкурентів

1.2.1 Google Maps

На рисунку (рисунок 1.1) можна побачити скріншот з популярного додатку мап Google Maps.

Переваги**:**

– висока швидкість роботи та надійність;

– інтуїтивно зрозумілий інтерфейс з можливістю персоналізації;

– реальне відображення руху транспорту з оновленнями в реальному часі;

– єдиний сервіс для побудови маршрутів пішоходів, водіїв і пасажирів громадського транспорту;

– потужна інфраструктура та постійні оновлення;

– інтеграція з іншими сервісами Google.

Недоліки**:**

– не завжди точно показує місцезнаходження зупинок, особливо в невеликих містах;

– інформація про рух транспорту може бути неточною через затримки в оновленнях;

– у деяких містах України не підтримує GTFS-дані через їх відсутність;

– обмежена функціональність у режимі офлайн;

– високе споживання ресурсів пристрою.

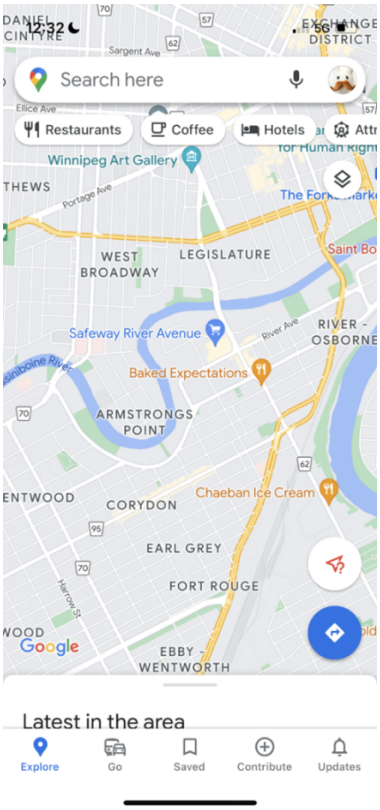

Рисунок 1.1 – Скріншот інтерфейсу Google Maps (рисунок виконаний самостійно)

1.2.2 EasyWay

На рисунку 1.2 скріншот з додатку EasyWay, який є популярним в Україні.

Переваги**:**

– велике покриття міст України, включаючи менші населені пункти;

– орієнтований на громадський транспорт, надає дані про маршрутні таксі, тролейбуси, трамваї та автобуси;

– наявність специфічних для України функцій та особливостей;

– можливість додавання коментарів та оцінок для зупинок.

Недоліки**:**

– інтерфейс часто зависає, загалом незручний і повільний, особливо на старіших пристроях;

– дані про рух транспорту можуть оновлюватися із запізненням;

– відсутність єдиного стандарту для обміну даними з іншими сервісами;

– обмежена функціональність для планування складних маршрутів з пересадками;

– недостатньо розвинені алгоритми прогнозування часу прибуття.

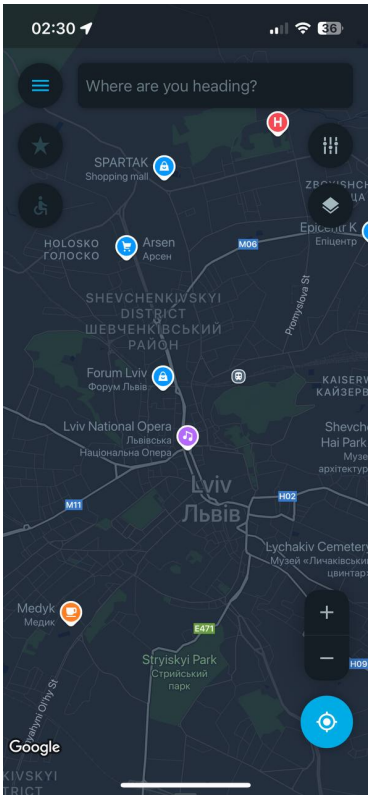

Рисунок 1.2 – EasyWay (рисунок виконаний самостійно)

1.2.3 Moovit

На рисунку 1.3 можна побачити відкриту карту у додатку Moovit.

Переваги**:**

– глобальне покриття з підтримкою багатьох міст світу;

– потужні алгоритми для планування маршрутів з пересадками;

– активна спільнота користувачів, що надає оновлення в реальному часі;

– підтримка різних видів транспорту, включаючи велосипеди та самокати.

Недоліки**:**

– обмежене покриття в українських містах;

– висока залежність від активності користувацької спільноти;

– значне споживання заряду батареї при активному використанні GPS;
– не завжди точна інформація в містах без офіційних GTFS-даних;
– у безкоштовній версії обмежені можливості і реклама.

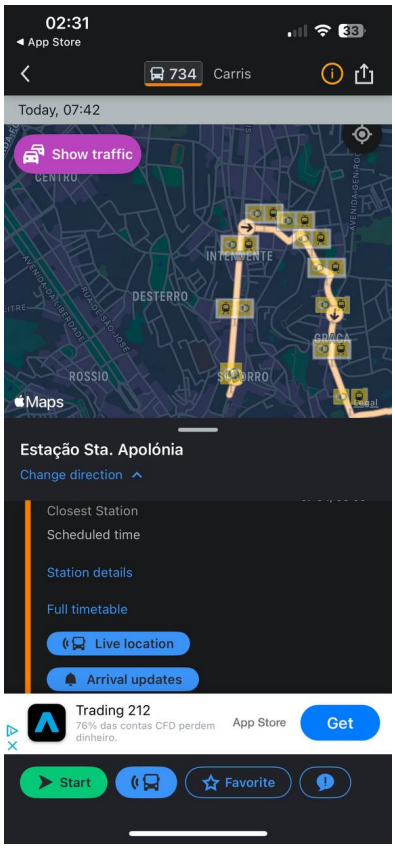

Рисунок 1.3 – Інтерфейс Moovit (рисунок виконаний самостійно)

1.2.4 Citymapper

На рисунку 1.4 скріншот з додатку Citymapper, де відкритий екран побудування маршруту.

Переваги**:**

– елегантний, інтуїтивно зрозумілий інтерфейс;
– детальна інформація про вартість проїзду та можливі знижки;
– інтеграція з сервісами таксі та каршерінгу;
– персоналізовані сповіщення та рекомендації.

Недоліки**:**

– дуже обмежене покриття в Україні (доступний лише в найбільших містах);

– відсутність підтримки деяких типів місцевого транспорту;

– необхідність постійного інтернет-з'єднання для повноцінної роботи;

– відсутність офлайн-режиму для більшості функцій;

– у безкоштовній версії обмежені можливості і реклама;

– повільно працює.

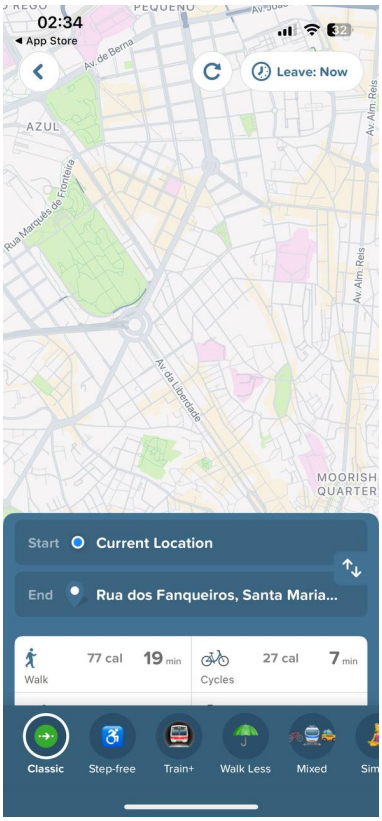

Рисунок 1.1 – Citymapper (рисунок виконаний самостійно)

Проаналізувавши переваги та недоліки існуючих рішень, можна визначити ключові напрямки розвитку для нового програмного застосунку: покращений користувацький інтерфейс, ефективне використання наявних GTFS-даних, можливість роботи в умовах нестабільного інтернет-з'єднання та підтримка різноманітних джерел даних для максимального покриття міст України.

1.3 Виявлення та вирішення проблем

1.3.1 Основні проблеми

Пасажири громадського транспорту часто стикаються з труднощами при плануванні поїздок, оскільки:

– вони не знають, які маршрути існують у конкретному місті, особливо якщо це туристи або люди, які нещодавно переїхали;

– відсутня точна інформація про час прибуття транспорту на зупинку, що призводить до неефективного планування часу та тривалого очікування;

– не всі транспортні засоби обладнані GPS-трекерами, що унеможливлює відстеження їхнього руху в реальному часі, особливо в менших містах або для деяких видів транспорту;

– деякі існуючі додатки мають недоліки в інтерфейсі, швидкодії або оновленні даних, що погіршує користувацький досвід;

– відсутність єдиного стандарту даних у різних містах України ускладнює створення універсального рішення для всієї країни;

– проблеми з доступністю інформації для людей з обмеженими можливостями;

– необхідність підтримки роботи додатка в умовах нестабільного інтернет-з'єднання.

1.3.2 Можливі вирішення

Для подолання зазначених проблем пропонується розробка програмного застосунку, який:

– об'єднає дані з існуючих сервісів і надаватиме актуальну інформацію про міський транспорт у режимі реального часу через єдиний інтерфейс;

– інтегруватиме GTFS-дані, а також працюватиме з іншими форматами (JSON, XML) для збору інформації з регіональних транспортних систем, що не підтримують стандарт GTFS;

– забезпечить зручний та швидкий інтерфейс для користувачів, включаючи можливість побудови оптимальних маршрутів з урахуванням пересадок та часу очікування;

– створить масштабовану інфраструктуру для підключення нових міст та транспортних систем без значних змін у базовій архітектурі;

– реалізує функції доступності для різних категорій користувачів, включаючи людей з обмеженими можливостями.

1.4 Постановка задачі

Для вирішення зазначених проблем необхідно створити веб-додаток, який надаватиме користувачам швидкий доступ до інформації про маршрути та рух громадського транспорту у їхньому місті.

1.4.1 Основні завдання

Розробка зручного та інтуїтивно зрозумілого інтерфейсу, що дозволить користувачам:

– переглядати доступні маршрути громадського транспорту;

– відстежувати рух транспортних засобів у реальному часі;

– планувати оптимальні маршрути з урахуванням пересадок;

– отримувати повідомлення про зміни в розкладі або затримки.

Інтеграція з існуючими системами відстеження транспорту, які працюють за стандартом GTFS:

– розробка універсального адаптера для імпорту та обробки GTFS-фідів;

– створення системи для регулярного оновлення даних;

– підтримка GTFS Realtime для отримання даних у режимі реального часу.

Забезпечення підтримки реального часу (наприклад, через WebSockets або push-сповіщення):

– розробка архітектури на основі подій для оновлення даних у реальному часі;

– оптимізація обміну даними для мінімізації навантаження на мережу.

Оптимізація продуктивності для швидкого завантаження даних навіть у мережах з низькою швидкістю:

– використання ефективних алгоритмів стиснення даних;

– впровадження стратегій поступового завантаження контенту;

– оптимізація запитів до бази даних для швидкого отримання результатів.

1.4.2 Цільова аудиторія

Основною цільовою аудиторією є пасажири громадського транспорту, а саме:

– мешканці міст, які щоденно користуються громадським транспортом для поїздок на роботу або навчання;

– туристи, які потребують швидкого орієнтування у незнайомому місті;

– люди похилого віку, яким потрібен простий та зрозумілий інтерфейс;

– люди з обмеженими можливостями, які потребують спеціальної інформації про доступність транспорту;

– водії таксі або кур'єри, які можуть використовувати додаток для визначення заторів на дорогах;

– бізнес-користувачі, яким важливо точно планувати час для ділових зустрічей.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1. Функціональні вимоги

2.1.1. Користувач

Користувач має доступ до всього функціоналу застосунку без необхідності авторизації. Основні потреби користувача:

– побудова маршрутів від точки А до точки Б;

– перегляд розташування зупинок;

– отримання інформації про приблизний час прибуття та відбуття транспортних засобів від зупинок;

– перегляд статусу конкретного маршруту, зупинки або транспортного засобу.

2.1.2. Оновлення та поширення даних

– транспортні засоби регулярно передають оновлення про своє місцезнаходження та швидкість;

– бекенд-частина обробляє отримані дані та поширює їх серед користувачів, які стежать за конкретним маршрутом чи зупинкою;

– інформація відображається в реальному часі у веб-додатку.

2.1.3. Доступ до інформації

– усі дані про транспорт, маршрути та зупинки доступні без авторизації;

– система не містить персональних даних користувачів та не потребує обмежень доступу.

2.1.4. Додаткові функціональні можливості

– фільтрація та пошук – зручний пошук маршрутів, зупинок і транспортних засобів за назвою або номером;

– історія маршрутів – перегляд раніше побудованих маршрутів для швидкого повторного використання.

2.1.5. Адміністративна панель

– моніторинг системи – перегляд активних транспортних засобів, маршрутів та статусу API-підключень;
– адміністрація підключених GTFS-систем – можливість додавати, видаляти, та модифікувати параметри які відносяться до взаємодії з зовнішніми GTFS-системами;
– можливість додавати маршрути, зупинки, і інші GTFS дані.

2.2. Не функціональні вимоги

2.2.1 Компоненти

Програмний застосунок для відстеження руху міського транспорту призначений для надання користувачам актуальної інформації про місцезнаходження, маршрути та розклад громадського транспорту. Система складається з трьох основних компонентів:

– веб-додаток – клієнтська частина, доступна через браузер;
– бекенд-частина – серверний компонент, який обробляє запити, зберігає та оновлює дані;
– база даних – центральне сховище інформації про маршрути, транспортні засоби, зупинки та GTFS-системи;
– шина комунікації між бекєнд компонентами – шар системи, який використовується для асинхронної комунікації між мікросервісами.

2.2.2 Підтримувані платформи

– Chrome: Мінімальна версія – 120 (на базі Chromium 120);
– Firefox: Мінімальна версія – 121;
– Safari: Мінімальна версія – 17.2;

– Edge: Мінімальна версія – 120 (на базі Chromium 120);

– Opera: Мінімальна версія – 106 (на базі Chromium 120).

2.2.3. Вимоги до продуктивності

Додаток має бути масштабуємий для того, щоб витримати підключення великої кількості GTFS-систем. Для цього треба визначити потенційне максимальне навантаження на систему, на прикладі статистичних даних відносно України, як приклад.

Потенційне навантаження на систему будемо рахувати відносно всієї України.

Станом на 1 січня 2022 року в Україні налічувалося 461 місто, включаючи 2 міста зі спеціальним статусом. Крім того, в країні було 881 селище міського типу та 28 369 сільських населених пунктів [8].

За даними на 1 січня 2020 року, в Україні було 44 міста з населенням понад 100 тисяч осіб, зокрема:

– 15 міст з населенням від 250 до 500 тисяч осіб;

– 5 міст з населенням від 500 до 1 мільйона осіб;

– 3 міста-мільйонники.

Більшість міст України належать до категорії малих міст (374 міста або 81%).

Якщо враховувати що GTFS система буде реалізована для кожного міста, то всього може бути 461 систем, з нерівним розподіленням навантаження для різних систем. 23 міста з 461 міст (тобто 5%) будуть формувати більшість трафіку (30-40%).

Оцінка навантаження на систему:

– добова активність користувачів: орієнтовно 10 мільйонів користувачів на день;

– середній час поїздки: 15 хвилин (900 секунд);

– кількість транспортних засобів (T): За розрахунками, у піковий час у системі може бути до 3,472 транспортних засобів;

– візьмемо верхній ліміт у 10,000 транспортних засобів.

Записи (writes) у систему:

Сервіс буде періодично (раз в 5-10 секунд) робити запит на оновлення даних до всіх GTFS-систем, які були підключені до системи. У найгіршому випадку, якщо враховувати Україну як потенційний масштаб деплойменту, то до системи буде підключено близько 500 GTFS систем, з яких <30 будуть формувати 30-40% трафіку*. У сценарії Polling + Event-based оновлень позицій, у нас буде 500 запитів в секунду, які загалом мають віддати поточні позиції 10,000 транспортних засобів. Це означає 10,000 оновлень / 5 секунд = ~2000 записів позицій у секунду. Вночі більша частина транспортних засобів не працює, тому навантаження на підсистему оновлення позицій буде мати характер нерівномірно навантаженої системи.

Читання (reads) у систему:

- середній час очікування пасажира на зупинці: 10 хвилин (600 секунд);

Кількість таких інтервалів протягом дня: 86,400 / 600 = 144;

- користувачів одночасно на зупинках: 10,000,000 / 144 = 100,000;
- у найгіршому випадку всі перевіряють статус одночасно;
- швидкість читання (RPS): 100,000 RPS.

Це означає, що система повинна підтримувати до 100,000 читань у секунду та 2,000 записів у секунду під максимальним навантаженням.

2.2.4 Міграції і підтримання актуального стану БД у майбутньому

Для забезпечення гнучкості та контрольованості змін структури бази даних у системі доцільно використовувати механізм міграцій. Міграції БД дозволяють версіонувати зміни схеми бази даних, забезпечуючи можливість поетапного оновлення структури без втрати даних [9].

Основними перевагами використання міграцій є: автоматизація процесу оновлення схеми БД на різних середовищах (розробка, тестування, продакшн), збереження повної історії всіх змін структури бази даних, що критично важливо для аудиту та відстеження джерел потенційних проблем, а також можливість відкату до попередніх версій схеми у разі виявлення помилок. Для реалізації міграцій можуть використовуватися спеціалізовані бібліотеки, такі як Alembic для

Python або Liquibase для Java-проектів, які забезпечують надійний та безпечний процес еволюції структури бази даних.

2.3 Обрані інструменти реалізації

Фронтенд було вирішено реалізувати використовуючи бібліотеку React і фреймворк React-Native для кросс-платформенної підтримки.

Бекенд реалізується за допомогою мови програмування Python версії 3.12, і фреймворку для веб-додатків Fastapi.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проектування ПЗ

Після аналізу потенційних взаємодій користувача з системою була створена Use-case діаграма (рисунок 3.1).

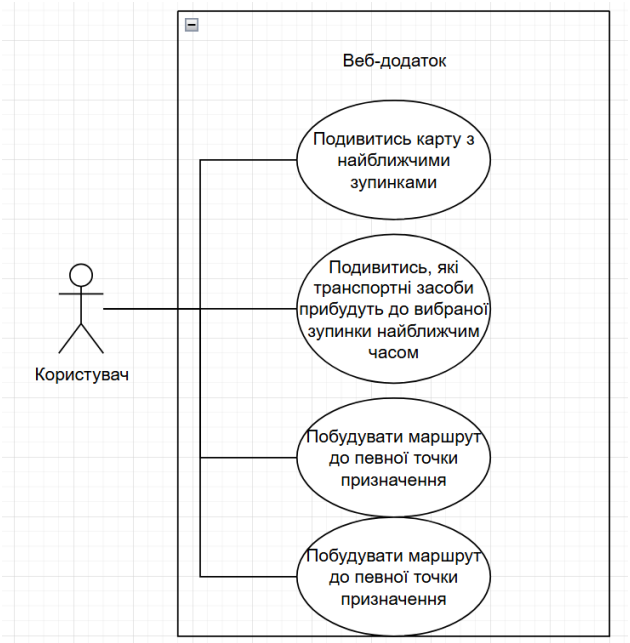

Рисунок 3.1 – Use-case діаграма додатку (рисунок виконаний самостійно)

В веб-додатку, користувач може відкрити карту, яка буде за замовчуванням показувати місцевість міста, у якому користувач знаходиться (також можна вибрати інше місто). На карті будуть показуватись найближчі зупинки у формі їх імені. Кожна зупинка має свою назву. Транспортні засоби, які рухаються до найближчих зупинок, також будуть показуватись на карті. Можна натиснути на зупинку, і тоді буде показано, які маршрути з неї виходять. Позиція транспортних засобів оновлюється кожні 5-10 секунд.

Користувач також може скористатися функцією пошуку, щоб знайти конкретну зупинку або маршрут, вказавши його номер або назву. Додатково передбачена можливість побудови оптимального маршруту, враховуючи час прибуття транспорту та пересадки.

На рисунку 3.2 можна побачити UML-діаграму функціоналу адміністративної панелі.

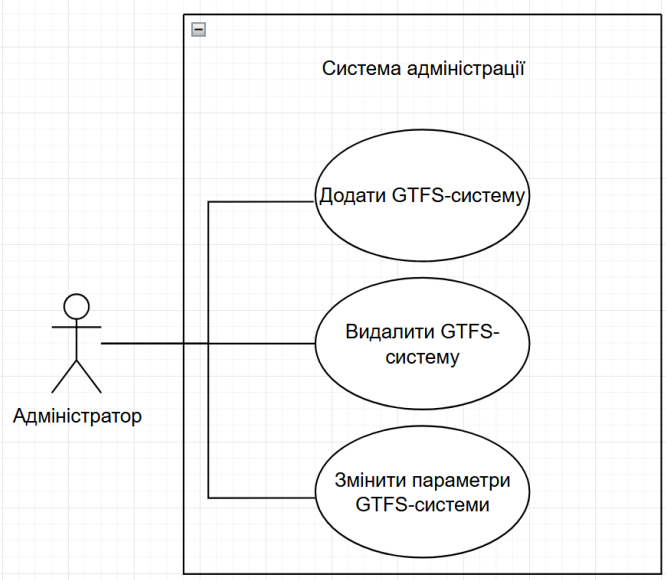

Рисунок 3.2 – UML-діаграма системи адміністрування (рисунок виконаний самостійно)

Рисунок 3.3 демонструє багатокомпонентний функціонал пошуку зупинок у системі, починаючи з додатку і закінчуючи базою даних.

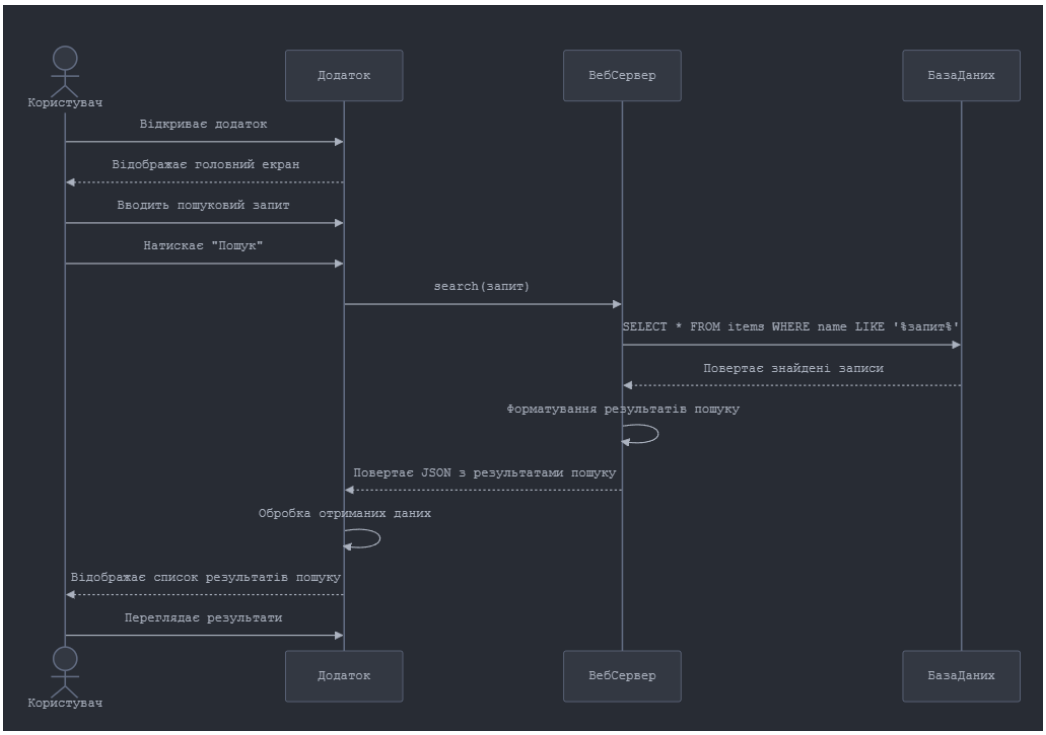

Рисунок 3.3 – UML-діаграма процесу пошуку зупинок (рисунок виконаний самостійно)

На рисунку 3.3 показано детальний процес роботи з пошуком у додатку. Для того, щоб відкрити інтерфейс пошуку, користувач має натиснути на відповідне меню пошуку, яке розташоване як накладний елемент поверх основної карти в користувацькому інтерфейсі на головному екрані додатка.

Після активації пошукового меню користувач має можливість ввести свій пошуковий запит у відповідне текстове поле. Система автоматично обробляє запит і миттєво відображає релевантні результати, які включають в себе як зупинки ТЗ, так і маршрути. Важливою особливістю є те, що біля кожного знайденого результату система показує точну відстань від поточного місцезнаходження користувача, що значно покращує користувацький досвід. Алгоритм сортування результатів пошуку працює за двома основними критеріями: відстанню від користувача та ступенем релевантності до введеного пошукового запиту.

Рисунок 3.4 демонструє Use-Case будування маршруту у системі.

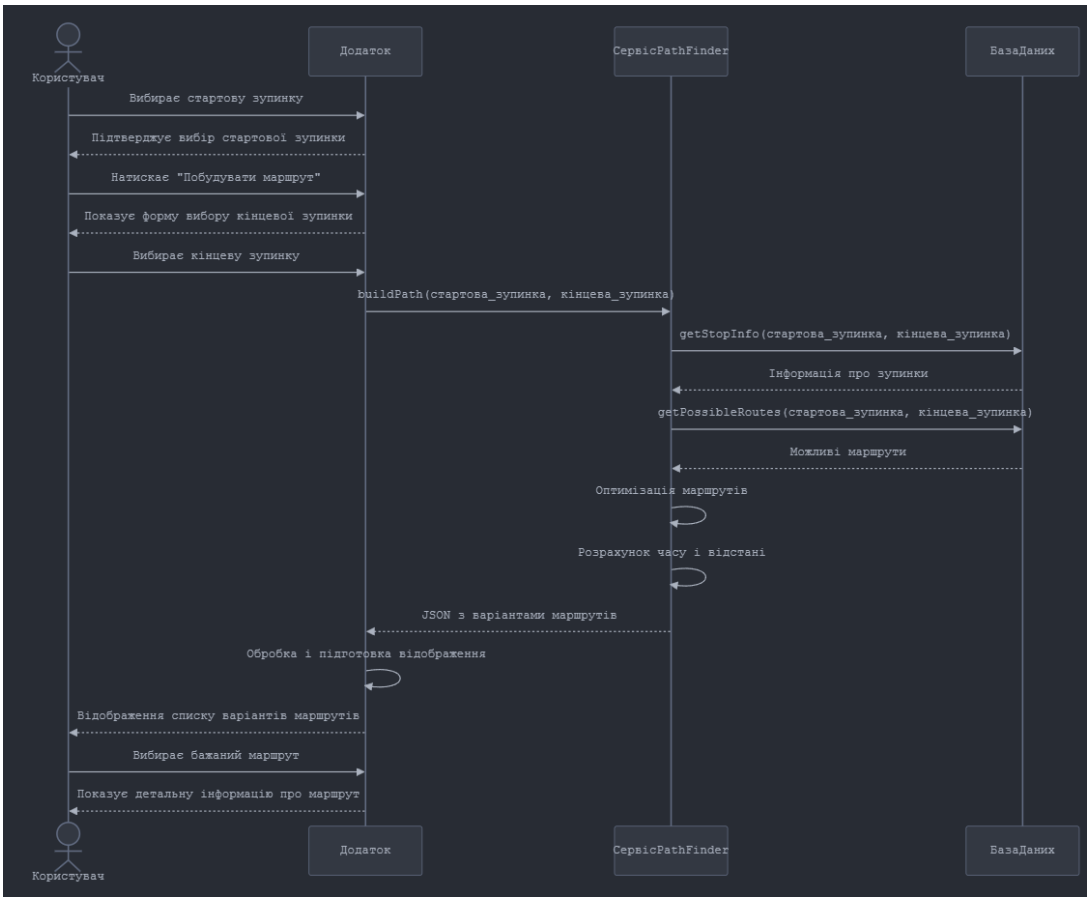

Рисунок 3.4 – UML-діаграма процесу будування маршруту між зупинками (рисунок виконаний самостійно)

Спочатку користувач вибирає початкову зупинку, потім з’являється меню вибору кінцевої зупинки. Після цього, фронтенд відправить запит у API для побудування маршруту до сервісу web-backend, який за допомогою SQL-запиту побудує оптимізований під користувача маршрут, який з’єднує дві вказані точки. У відповіді API буде вказана інформація про початкову і кінцеві зупинки, і дані для кожного елементу маршруту, такі як: геометрія, назва частини маршруту, а також інструкція. Після відповіді API, інтерфейс буде показувати маршрут на мапі, де він буде поділений відповідно до кроків, які треба прийняти користувачу, щоб успішно доїхати від початкової зупинки до кінцевої.

3.2 Проектування архітектури ПЗ

Система веб-додатку для відстеження громадського транспорту реалізована з використанням мікросервісної архітектури, що забезпечує масштабованість, незалежність розгортання та технологічну гнучкість.

3.2.1 Шаблони проектування

У процесі проектування та реалізації було застосовано кілька ключових архітектурних шаблонів проектування (Design Patterns), які забезпечують надійність, продуктивність та підтримуваність системи.

Шаблон Event-Driven Architecture є основою для real-time комунікації в системі відстеження транспорту. Цей підхід забезпечує асинхронну передачу даних про оновлення позицій транспортних засобів між різними компонентами системи.
Реалізація в системі:

- GPS-пристрої транспортних засобів генерують події зміни позиції;
- події передаються через Apache Kafka як центральний event broker;
- мікросервіси підписуються на відповідні топіки подій;
- веб-клієнти отримують оновлення через WebSocket з'єднання в реальному часі.

Переваги застосування:

- слабка зв'язаність між компонентами системи;

– можливість незалежного масштабування сервісів;
– асинхронна обробка подій не блокує основні операції;
– простота додавання нових споживачів подій.

Шаблон Event Sourcing реалізовано для збереження повної історії змін стану транспортних засобів. Замість збереження лише поточного стану, система зберігає послідовність подій, що дозволяє відтворити історію переміщень транспортних засобів.

Реалізація в системі:

– всі події зміни позиції транспорту зберігаються в Apache Kafka;
– кожна подія містить timestamp, ідентифікатор транспортного засобу та географічні координати;
– історичні дані дозволяють аналізувати маршрути та оптимізувати розклади;
– можливість відтворення траєкторії руху транспортного засобу за будь-який період.

Шаблон Health Check впроваджено в кожному мікросервісі для забезпечення моніторингу стану системи та автоматичного управління життєвим циклом сервісів.

Реалізація в системі:

– кожен мікросервіс експонує HTTP endpoint /health-check;
– endpoint повертає детальну інформацію про стан сервісу та його залежностей;
– перевіряється з'єднання з базою даних, доступність зовнішніх API, стан черг повідомлень;
– інтеграція з оркестраторами контейнерів (Kubernetes, Docker Swarm).

Переваги застосування:

– автоматичне виявлення та перезапуск недоступних сервісів;
– можливість автоматичного масштабування на основі стану сервісів;

– проактивний моніторинг та попередження про проблеми;

– інтеграція з системами моніторингу (Prometheus, Grafana).

Шаблон Retry Pattern застосовується для забезпечення стійкості системи до тимчасових збоїв та підвищення загальної надійності операцій.

Реалізація в системі:

– Celery завдання автоматично перезапускаються при виникненні помилок;

– використовується експоненційна затримка (exponential back-off policy) між спробами для уникнення перевантаження;

– встановлено максимальну кількість спроб для запобігання нескінченним циклам;

Конфігурація повторних спроб:

```
@celery.task(bind=True,autoretry_for=(Exception,),
retry_kwargs={'max_retries': 3, 'countdown': 60})
def process_vehicle_position(self, vehicle_data):
    # Логіка
```

Переваги застосування:

– автоматичне відновлення після тимчасових збоїв мережі;

– зменшення втрати даних при короткочасних проблемах;

– підвищення загальної надійності системи;

– зниження необхідності ручного втручання при збоях.

3.2.2 Проектування мок-системи GTFS

Мок-система (див. рис. 3.5) необхідна для того щоб симулювати поведінку справжньої системи, яку можна інтегрувати у систему відстеження міського транспорту. Вона по суті є емуляцією зовнішніх API, які в майбутньому можна буде підключити до системи. Цей сервіс надає endpoint /api/v1/vehicles, який віддає у стандартному форматі дані про поточний стан усіх транспортних засобів.

В базі даних зберігається мок-інформація різних маршрутів, зупинок, і так далі, а також є декілька внутрішніх сервісів (RouteService, StopService, тощо), які

працюють з даними відповідно до їх назви. Основний компонент це VehiclePositionService, який симулює рух і активність у транспортній системі, основуючись на даних з БД. Сервіс надає API визначене у GTFS специфікації, і передає запити на відповідні сервіси.

3.2.3 Проектування gtfs-reader

gtfs-reader це компонент системи, який має отримувати дані через поллінг (polling) механізм з усіх налаштованих GTFS-сервісів, і публікувати ці дані у загальну шину комунікацій, а також надавати інтерфейс для завантаження статичних GTFS даних.

Діаграма системи (див. рисунок 3.5), відображає взаємодію gtfs-reader з потенційним GTFS-сервісом (на малюнку відображений мок-сервіс для прикладу). У базі даних зберігається мета інформація відносно налаштованих сервісів, на які gtfs-reader-worker роблять запити. Результат обробки відповіді з GTFS-сервісів складається у Kafka.

Також до gtfs-reader підключена адміністративна панель, за допомогою якої можна завантажити статичні GTFS дані до бази даних.

3.2.4 Проектування web-backend

Web-backend (див. рис. 3.5) виконує роль єдиного інтерфейсу для веб-додатка до системи, тобто реалізує функціонал запитів до бази даних, і також оновлення real-time даних через WebSocket.

На діаграмі можна побачити, що web-backend є єдиною вхідною точкою у систему для клієнтів. Він надає декілька API точок доступу, і передає запити до відповідних сервісів. В середині сервісу web-backend є два основних компоненти: сервіс «PathBuilderService”, який відповідає за запити побудування маршруту, і VehiclePositionUpdatesService, який відповідає за передачу оновлень транспортних засобів через протокол WebSocket, і є шаром між Kafka і web-backend. Також відображений зв’язок з базою даних, до якої робляться запити відносно статичних GTFS даних, і також для побудування маршрутів.

Діаграма архітектури демонструє складну взаємодію між різними компонентами системи громадського транспорту. В центрі системи знаходиться administrator - адміністративна панель, яка керує всіма процесами через HTTPS протокол. Від неї відходять основні потоки даних до різних сервісів, включаючи контейнеризовані мікросервіси для обробки транспортної інформації та GTFS даних.

Особливо важливим елементом є потік real-time даних, який проходить через систему Kafka кластер, забезпечуючи високопродуктивну обробку потокових даних про позиції транспортних засобів. Цей потік даних в подальшому обробляється спеціалізованими сервісами та передається до web-backend через WebSocket з'єднання для забезпечення миттєвого оновлення інформації на клієнтській стороні.

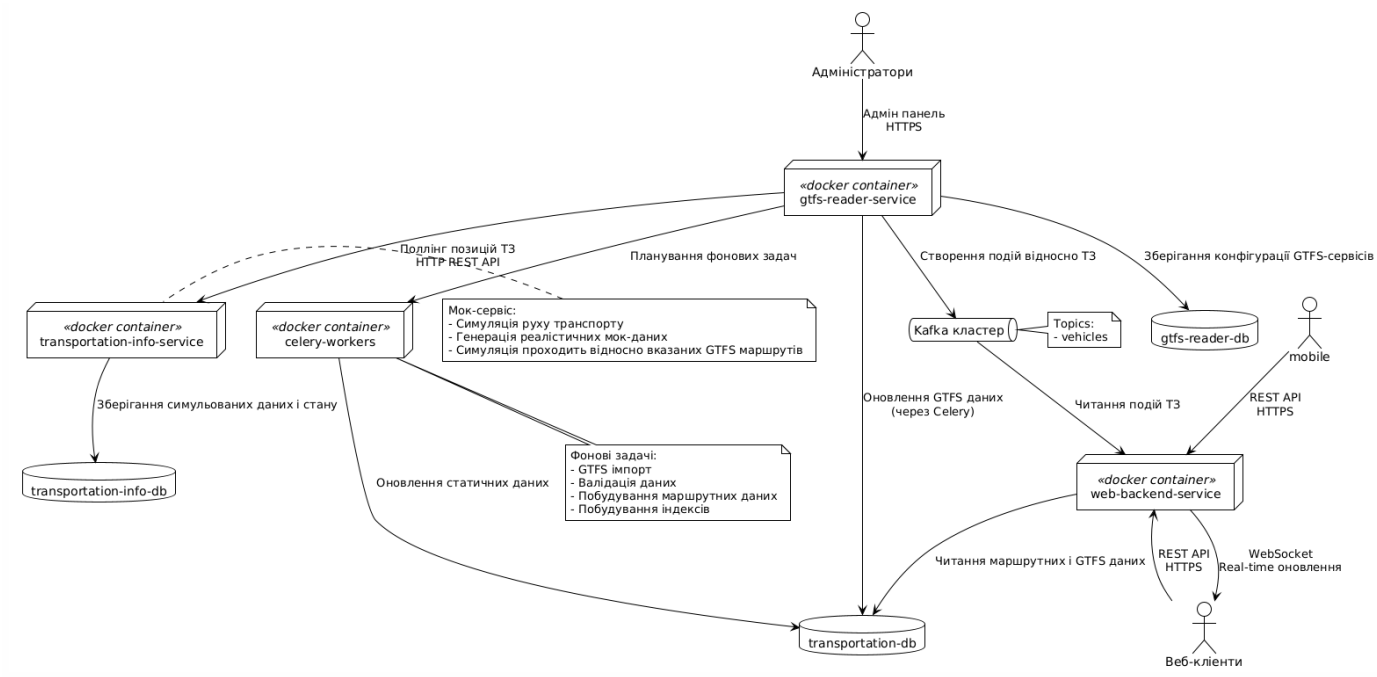

Рисунок 3.5 – Архітектура розгортання бекенду (рисунок виконаний самостійно)

3.3 Проектування структури зберігання даних

Для зберігання даних використовується реляційна база даних PostgreSQL з аддонами PostGIS та pgRouting для збереження і обробки гео-просторих даних.

3.3.1 Зберігання зареєстрованих GTFS систем

Gtfs-reader має необхідність у зберіганні мета-даних для кожного підключеного GTFS-сервісу (див. рис. 3.6).

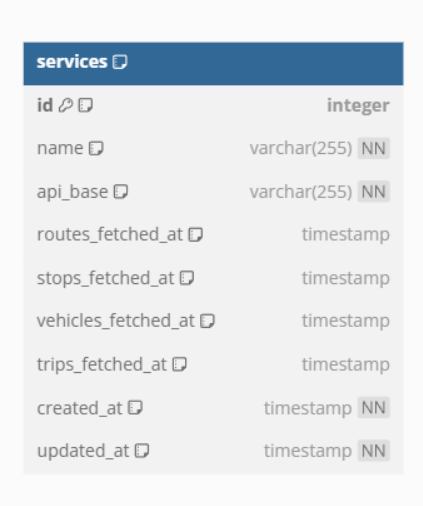

Рисунок 3.6 – Діаграма бази даних transportation_meta_db (рисунок виконаний самостійно)

3.3.2 Діаграма transportation_db

У схемі бази даних transportation_db (рисунок 3.7) можна побачити основні GTFS структури, з якими буде взаємодіяти web-backend.

Таблиця routes (маршрути)

- зберігає інформацію про маршрути громадського транспорту;
- містить дані про номер, назву та тип транспорту;
- таблиця stops (зупинки);
- зберігає інформацію про зупинки громадського транспорту;
- включає назву, географічне розташування та тип зупинки.

Таблиця vehicles (транспортні засоби)

- зберігає інформацію про транспортні засоби;
- містить дані про поточне місцезнаходження та статус;

Таблиця trips (рейси)

- зберігає інформацію про конкретні рейси на маршрутах;

– пов'язує маршрути з розкладом руху.

Таблиця route_stops (зупинки на маршруті)

– зв'язує маршрути та зупинки;

– визначає послідовність зупинок на маршруті.

Таблиця shapes (форми маршрутів)

– зберігає геометричні форми маршрутів на карті;

– містить лінійні дані для відображення шляху руху;

– забезпечує візуальне представлення траєкторії маршруту на картографічному інтерфейсі.

Таблиця shape_points (точки форми)

– зберігає окремі точки, з яких складаються форми маршрутів;

– визначає детальну траєкторію руху транспорту;

– забезпечує високу точність відображення реального шляху руху.

Таблиця stop_times (час прибуття/відправлення)

– зберігає розклад для кожної зупинки в рейсі;

– містить заплановані часи прибуття та відправлення.

Ця реляційна структура бази даних забезпечує повну підтримку стандарту GTFS та дозволяє ефективно обробляти як статичні дані розкладу, так і динамічну інформацію про рух транспорту. Зв'язки між таблицями організовані таким чином, щоб мінімізувати дублювання даних та забезпечити високу продуктивність запитів при роботі з великими обсягами транспортної інформації.

Особлива увага приділена оптимізації геопросторових даних у таблицях shapes та shape_points, які використовують спеціалізовані індекси для ефективного пошуку маршрутів у заданому географічному регіоні. Це дозволяє web-backend швидко відповідати на запити користувачів щодо найближчих зупинок, побудови оптимальних маршрутів та відображення актуальної інформації про рух транспорту в режимі реального часу. Крім того, структура бази даних підтримує масштабування через партиціонування великих таблиць за часовими або географічними критеріями, що забезпечує стабільну роботу системи навіть при значному зростанні кількості користувачів та обсягу транспортних даних.

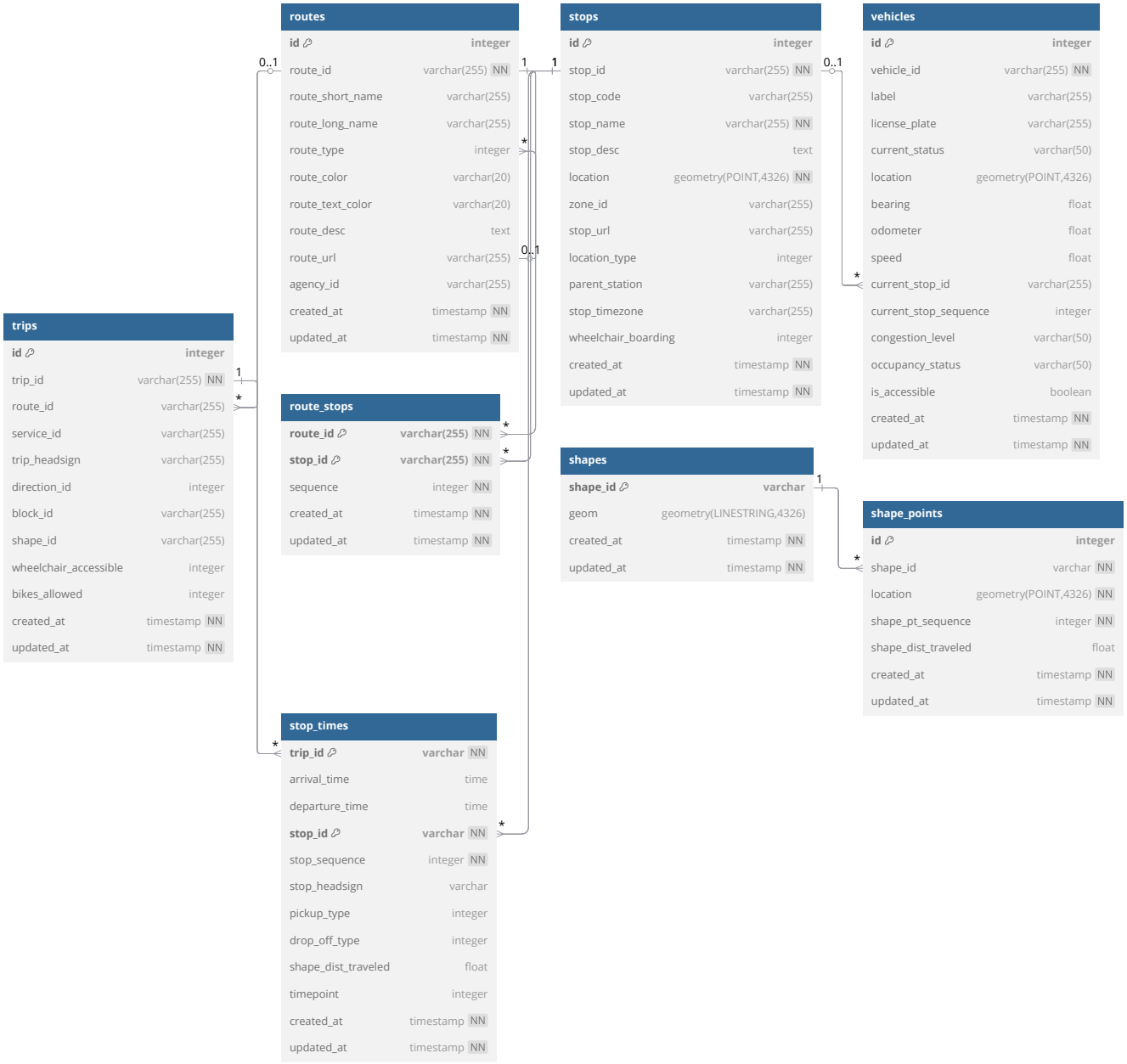

Рисунок 3.7 – Схема бази даних transportation_db (рисунок виконаний самостійно)

3.3.3 Діаграма transportation_db (частина маршрутизації)

Для реалізації функціоналу маршрутизації (побудування маршрутів), було вирішено будувати окремі таблиці, які собою представляють граф зв’язків між транспортними вузлами (див. рисунок 3.8).

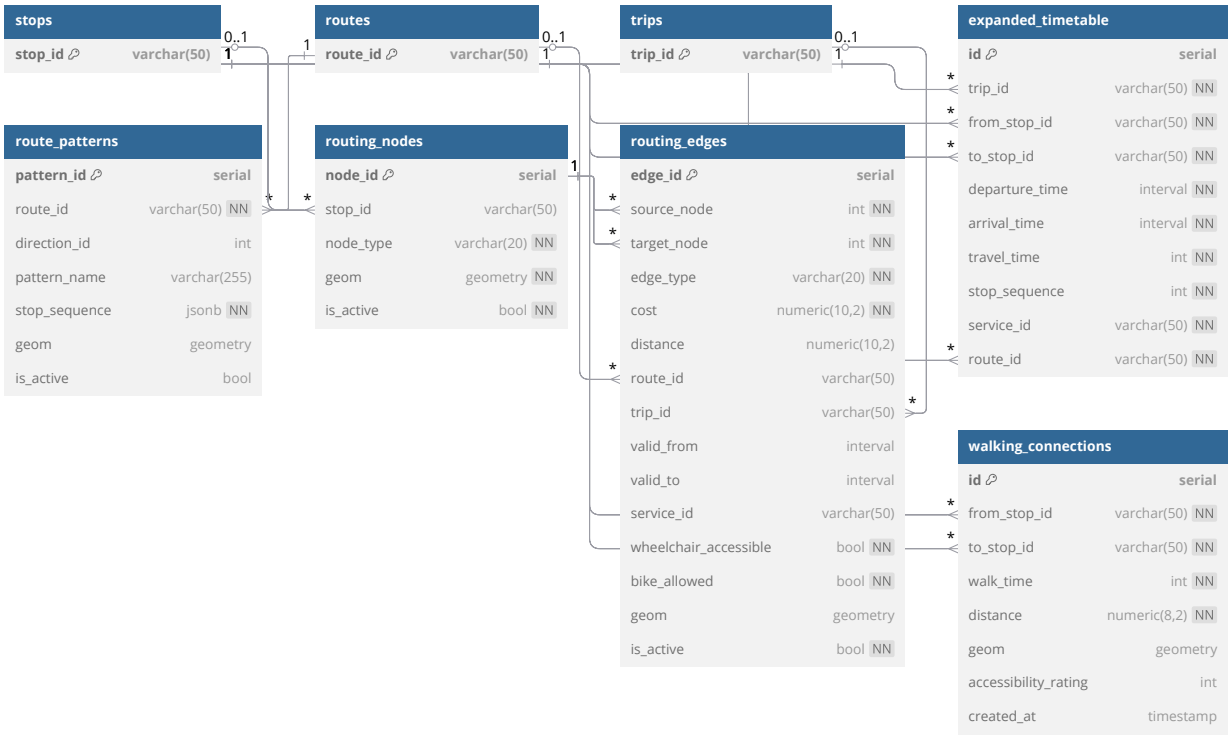

Рисунок 3.8 – Схема маршрутної частини бази даних transportation_db (рисунок виконаний самостійно)

3.3.4 Персоналізоване зберігання даних

Для кожного користувача треба зберігати історію побудування маршрутів, і збереженні («улюблені») зупинки. Для цього було вирішено використовувати технологію LocalStorage, вбудовану у кожен сучасний браузер. Схему зберігання даних у LocalStorage можна побачити на рис. 3.9.

Рисунок 3.9 – Схема браузерного LocalStorage (рисунок виконаний самостійно)

3.4 Приклади найцікавіших алгоритмів та методів

3.4.1 Знайдення найближчої та наступної зупинки для симуляції руху

Алгоритм знаходження найближчої зупинки (повний код можна знайти у Додатку А) використовує метод розрахунку географічної відстані між поточним положенням транспортного засобу та зупинками на маршруті. Основні кроки алгоритму:

1. отримання всіх зупинок для вказаного маршруту;
2. ітерація по зупинках та розрахунок відстані до кожної;
3. вибір зупинки з мінімальною відстанню;
4. визначення наступної зупинки на основі індексу найближчої.

Ключова частина реалізації:

```
for idx, stop in enumerate(stops):
    distance = PositionCalculator.calculate_distance(
        float(cast(float, vehicle.latitude)),
        float(cast(float, vehicle.longitude)),
        float(cast(float, stop.stop_lat)),
        float(cast(float, stop.stop_lon)),
    )
    if distance < min_distance:
        min_distance = distance
        closest_stop = stop
        closest_stop_idx = idx
```

Алгоритм використовує циклічну структуру маршруту, де наступна зупинка після останньої - це перша зупинка маршруту:

```
next_stop_idx = (closest_stop_idx + 1) % len(stops)
next_stop = stops[next_stop_idx]
```

3.4.2 Синхронізація позицій транспортних засобів на клієнті

React хук useRouteVehicleUpdates (див. Додаток Б) реалізує патерн реактивного оновлення даних у реальному часі з використанням WebSocket. Основні компоненти алгоритму:

1. встановлення WebSocket-з'єднання при виборі маршруту;
2. обробка повідомлень з оновленнями позицій транспорту;
3. оптимізована стратегія оновлення стану без зайвих перерендерингів;
4. автоматичне очищення застарілих даних по таймеру.

Оптимізація оновлення позицій транспортних засобів реалізована через ефективне використання React Query:

```
ws.onmessage = (event) => {
  try {
    const update: VehicleUpdate = JSON.parse(event.data);
    queryClient.setQueryData(queryKey, (oldData: VehicleLocation[] = []) =>
{
      // Find if this vehicle already exists in our data
      const existingIndex = oldData.findIndex(
        vehicle => vehicle.vehicleId === update.vehicleId
      );
      if (existingIndex >= 0) {
        // Update existing vehicle location
        const newData = [...oldData];
        newData[existingIndex] = {
          vehicleId: update.vehicleId,
          latitude: update.latitude,
          longitude: update.longitude,
          timestamp: update.timestamp
        };
        return newData;
      } else {
        // Add new vehicle location
        return [...oldData, {...}];
      }
    });
  } catch (error) {
    console.error("Error parsing WebSocket message:", error);
  }
};
```

Алгоритм також включає механізм автоматичного очищення застарілих даних, що забезпечує актуальність відображення транспортних засобів на карті:

```
  useEffect(() => {
    if (!routeId) return;

    const ws = new WebSocket(`ws://localhost:8002/ws/${routeId}`);
    wsRef.current = ws;
    console.log("Opening WebSocket connection", routeId);

    ws.onmessage = (event) => {
      try {
        const data: VehicleResponse = JSON.parse(event.data);
        setVehicles((prev) => {
          const   updated   =   prev.filter((v)   =>   v.vehicle_id   !==
data.vehicle_id);
          return [...updated, data];
        });
      } catch (err) {
        console.error("Failed to parse vehicle update:", err);
      }
    };

    return () => {
      console.log("Closing WebSocket connection", routeId);
```

```
      ws.close();
    };
  }, [routeId]);
```

Комбінація WebSocket для реальночасових оновлень та React Query для керування станом забезпечує ефективну синхронізацію даних з мінімальним навантаженням на клієнтський пристрій та мережу.

3.5 Створення UI / UX

У цьому розділі було схематично описано дизайн веб-додатку і описані загальні підходи які були використані для дизайну.

3.5.1 Загальний підхід і стиль

Ергономічність

– головний пошуковий рядок зроблено широким і помітним у верхній частині екрану, тому що пошук маршруту є найчастішою дією користувачів;

– карта займає 90% екрану, тому що візуальне представлення маршрутів і положення транспорту є ключовою інформацією для користувача;

– масштабування карти реалізовано як через скролінг, так і через кнопки, оскільки це підвищує зручність користування для різних категорій користувачів.

Адаптивність/гнучкість

– адаптивний дизайн було розроблено таким чином, щоб веб-сайтом можна було користуватись на пристроях різних розмірів;

– елементи інтерфейсу автоматично перебудовуються при зміні орієнтації пристрою, оскільки це забезпечує зручність використання в різних ситуаціях;

– розмір шрифту можна збільшувати без порушення структури інтерфейсу, тому що доступність для користувачів з вадами зору є пріоритетом.

Інтуїтивна зрозумілість

– іконки для типів транспорту відповідають загальноприйнятим стандартам, тому що це дозволяє користувачам швидко розпізнавати їх значення;

– перетягування маркера на карті автоматично розраховує маршрути до цієї точки, оскільки це інтуїтивний спосіб планування подорожі;

– візуальне підтвердження дій (підсвічування кнопок, анімації переходів) реалізовано, тому що це зменшує невизначеність при взаємодії з додатком;

– повідомлення про помилки сформульовані простою мовою з пропозицією рішення, оскільки це зменшує фрустрацію користувачів.

3.5.2 Специфікація

Кольорова схема

– синій колір (#1E88E5) обрано основним, оскільки він асоціюється з надійністю та стабільністю, що важливо для транспортного додатку;

– кожен тип транспорту має власний колір (автобуси - зелений, трамваї - червоний, тролейбуси - синій), тому що це прискорює візуальну ідентифікацію;

– контрастність тексту з фоном перевищує стандарт 4.5:1, оскільки це забезпечує читабельність у різних умовах освітлення;

– для виділення активних маршрутів використовується синій колір (#0000FF), тому що він створює чіткий візуальний акцент без агресивної яскравості.

Типографіка

– шрифт Roboto було обрано через його високу читабельність на цифрових пристроях та широку підтримку в різних операційних системах;

– розмір шрифту для основного тексту встановлено 16px, оскільки це забезпечує комфортне читання на пристроях різних розмірів;

– для номерів маршрутів використовується напівжирне накреслення, тому що ця інформація є критично важливою для швидкої ідентифікації;

– для часу прибуття застосовується моноширинний шрифт, що забезпечує кращу читабельність числових даних.

3.5.1 Основний екран

На рисунку 3.10 можна побачити основний екран. На фоні інтерактивна карта, а зверху пошук, який є основним інтерактивним елементом. Також схематично відзначені зупинки, на які можна нажати щоб подивитись інформацію про них.

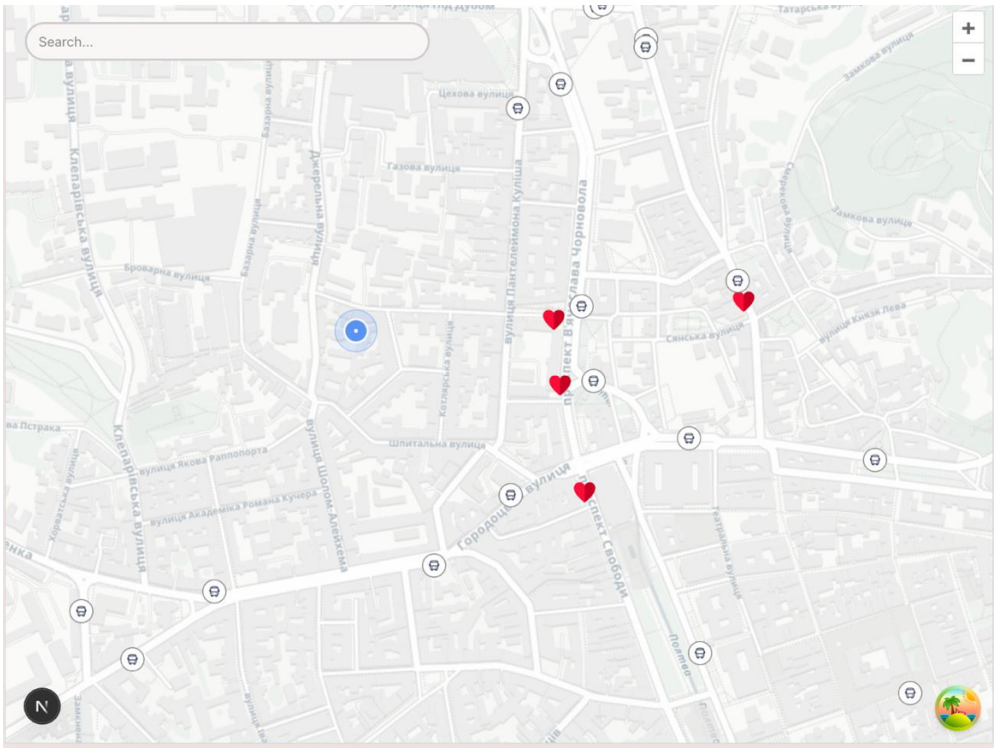

Рисунок 3.10 – Основний екран додатку (рисунок виконаний самостійно)

3.5.2 Екран вибраної зупинки

На рисунку 3.11 можна побачити елемент інтерфейсу що надає інформацію про усі можливі маршрути, які йдуть з вибраної зупинки, а також кнопку будування маршрутів від зупинки. Інтерфейс також включає елементи керування мапою: кнопки масштабування у правому верхньому куті мапи, що дозволяє користувачу дивитись мапу у різних масштабах.

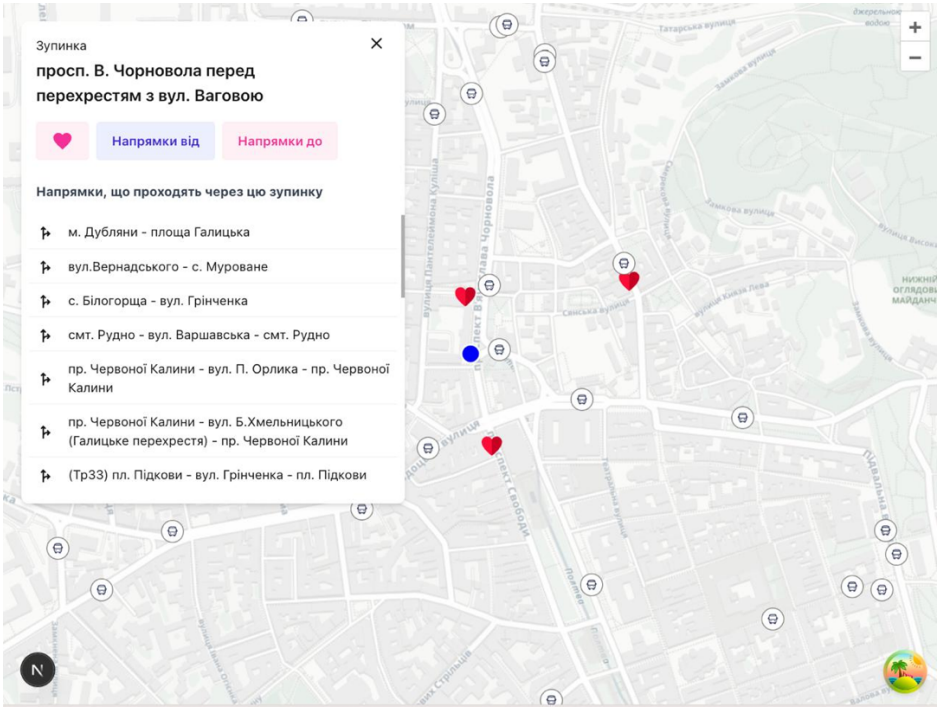

Рисунок 3.11 – Екран з вибраною зупинкою (рисунок виконаний самостійно)

Рисунок 3.11 також демонструє, що у кожної зупинки є індикатор статусу збереження зупинки у улюблені (відображається сердечком). Він червоного заповненого кольору, якщо зупинка була збережена користувачем. Якщо вона не була збережена, то індикатор буде не заповненим. На екрані зупинки є дві кнопки: Побудувати напрямки від, і побудувати напрямки до зупинки. Якщо натиснути на одну з цих кнопок, з’явиться вікно будування маршрутів з або до відповідної зупинки.

Напрямки відображаються як елементи списку у вікні вибраної зупинки. Зліва є іконка маршруту. Наприклад, показано маршрути до таких локацій як м. Дубляни - площа Галицька, вул. Вернадського - с. Мурована, смт. Руднє - вул. Варшавська - смт. Руднє, та інші.

Заголовок відображає назву зупинки, як найважливіший шматок інформації, яку треба відобразити на цьому екрані.

Вибрана зупинка також відображається синим круглим елементом на мапі, що корисно для геолокації об’єкту користувачам. Якщо закрити вікно вибраної зупинки, то вона перестане бути вибраною, і знову покажеться пошук.

3.5.3 Екран вибраного маршруту

На рисунку 3.12 відображається вибраний маршрут Y (назва маршруту), і також номер автобусу, який по цьому маршруту ходить. На карті виділений сам маршрут у географічному розумінні.

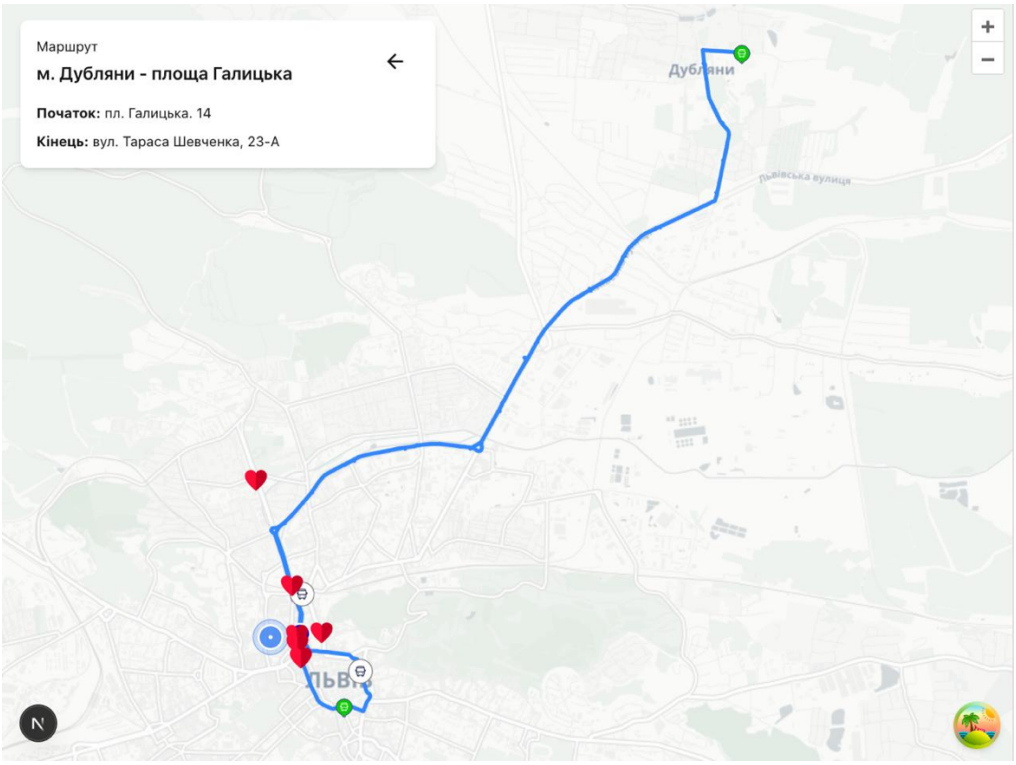

Рисунок 3.12 – Вибраний маршрут (рисунок виконаний самостійно)

На рисунку 3.12 також можна побачити, що маршрут відображається як синя ломана лінія на мапі.

При виборі маршруту, на мапі відображаються зелені індикатори автобусів (або інших видів транспортних засобів), що на даний час рухаються по цьому маршруті. Це корисно для того, щоб користувач міг розуміти, як довго чекати поки транспорт приїде.

Тут мапа досить далеко віддалена, щоб побачити, що відображається менша кількість зупинок. Це пов’язано з тим, що був реалізований у інтерфейсі адаптивний рендеринг зупинок, у цілях запобігання відображення занадто багатої кількості точок на карті. «Улюблені» зупинки завжди показуються, незалежно від того, у якому форматі масштабування знаходиться мапа користувача.

3.5.4 Екран пошуку маршруту

На рисунку 3.13 відображені результати пошуку за запитом «про», і мінімальна інформація про кожен результат (тип результату, географічна відстань).

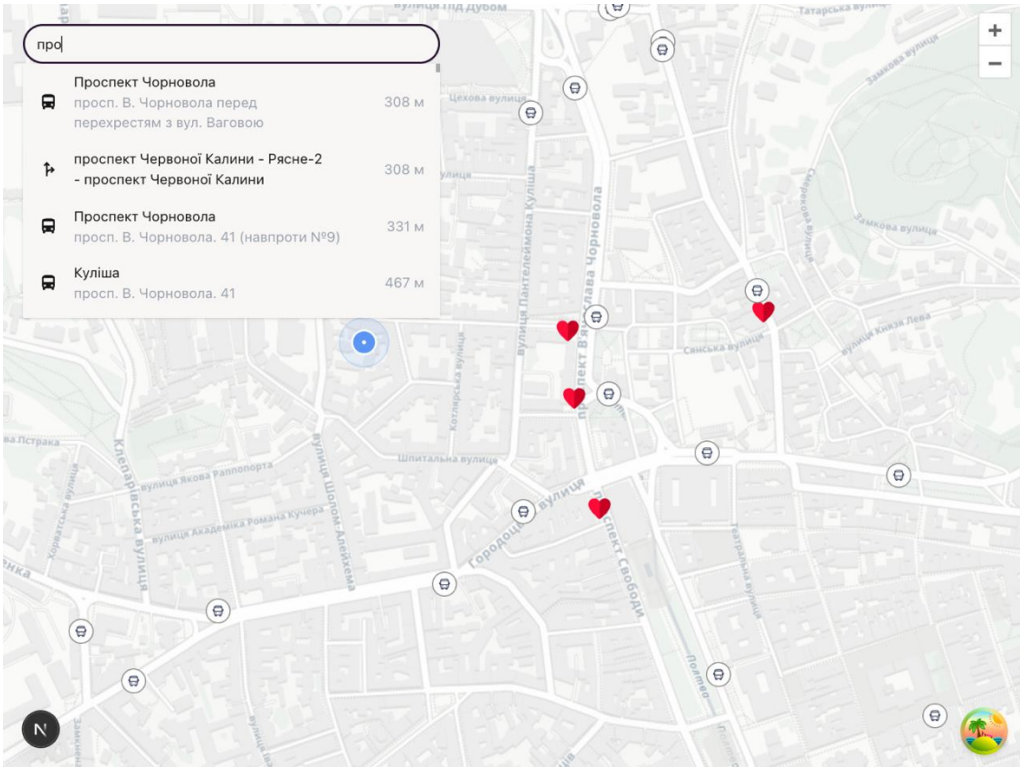

Рисунок 3.13 – Пошук та результати пошуку (рисунок виконаний самостійно)

Рисунок 3.13 демонструє, що кожний результат пошуку відображає відстань до нього у одиницях вимірювання метричній системі (зазвичай метри або кілометри) - у даному випадку показані відстані 308 м, 331 м та 467 м. У кожного результату також є іконка зліва, яка означає тип результату пошуку, тобто зупинка або маршрут. У елементів списку є назва самого результату, і більш детальний (але все одно короткий) опис трохи нижче. Менш релевантні деталі відображаються сірим шрифтом для того щоб користувач міг сфокусуватись на основній деталі, а саме назві результату.

Якщо натиснути на елемент списку, то відповідна сутність буде вибрана на мапі, і буде показаний відповідний екран.

Список результатів пошуку є максимальну висоту, щоб не заповнювати весь екран при великій кількості результатів, і також має вбудований скроллбар.

3.5.5 Екран побудування маршруту

На рисунку 3.14 видно екран результату побудови маршруту від зупинки «проспект Свободи» до зупинки «Парк 700-річчя Львову». На мапі відображається сам маршрут, а у панелі зліва перелічені варіанти подорожі по цьому маршруту.

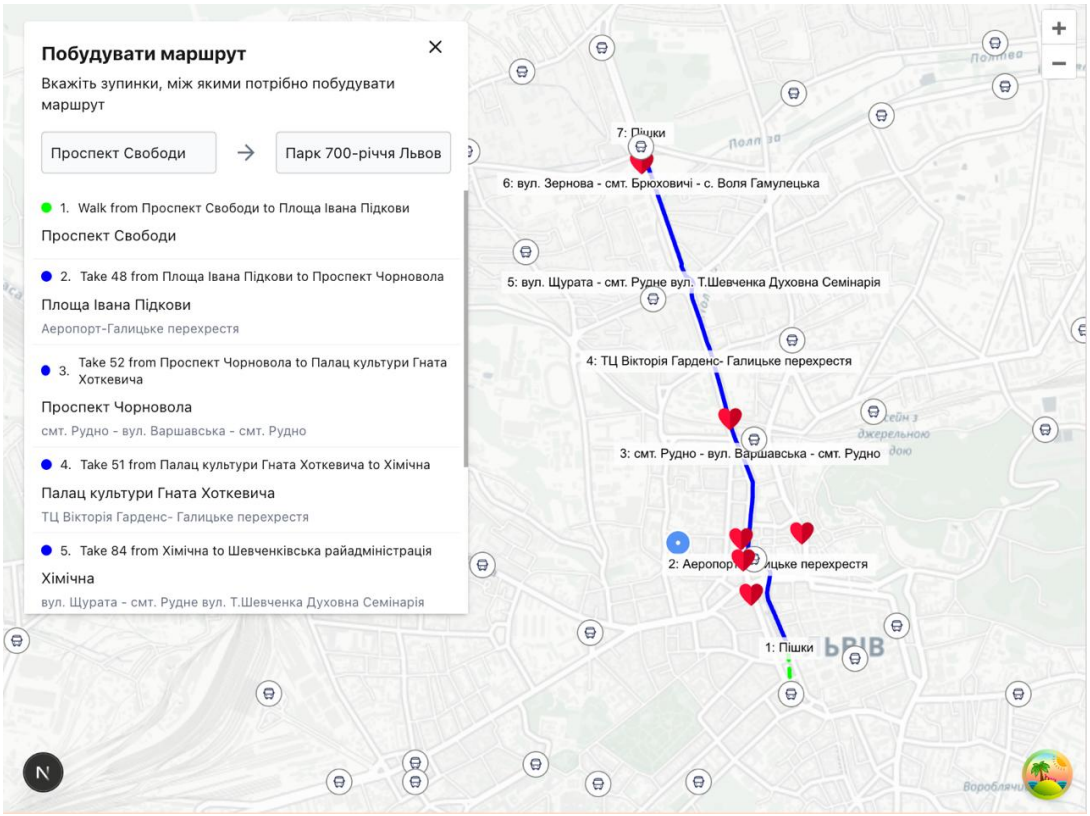

Рисунок 3.14 – Інтерфейс побудови маршруту (рисунок виконаний самостійно)

Рисунок 3.14 також показує, що для кожного елементу побудованого маршруту є інструкції (на екрані можна побачити одразу після номеру пункту), в якій буде відображено тип пересування (пішки або на транспорті), і також номер транспорту у випадку якщо треба їхати. Для кожного елементу також показується назва маршруту, до якого цей елемент відноситься.

Елементи мають кольоровий круглий індикатор у лівій частині, який відображає відповідний тип елементу: зелений для пішого, синій для маршруту транспортного засобу, і червоний для останнього пункту. Це допомагає користувачам краще орієнтуватись у маршруті.

Також на екрані є поля, де можна змінити параметри пошуку маршруту.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 База даних

Для збереження постійних даних було вирішено використовувати PostgreSQL, як одна з найбільш популярних баз даних, яка має високу швидкість роботи, і має підтримку більшості мов та середовищ програмування.

Postgres як система баз даних тут підходить краще тому що:

1. Це одна з найбільш розповсюджених RDBMS. Це спростить підтримку цієї системи у майбутньому (чим популярніше система, тим більше спеціалістів і ресурсів онлайн для вирішення різноманітних проблем, а також більше реалізованих плагінів для розширення функціоналу системи).
2. Стабільність. Ця СУБД відома своєю надзвичайною надійністю та стійкістю до збоїв навіть при високих навантаженнях. PostgreSQL має перевірені часом механізми для забезпечення цілісності даних, включаючи журналювання транзакцій наперед (Write-Ahead Logging), що гарантує відновлення даних після непередбачених збоїв. Система також підтримує реплікацію, що додатково підвищує доступність даних та забезпечує безперебійну роботу. Важливо, що розробники PostgreSQL завжди приділяють особливу увагу якості коду та ретельному тестуванню перед випуском нових версій, що мінімізує ризик появи критичних помилок. Багато великих організацій та підприємств покладаються на PostgreSQL саме через цю надійність та стабільність, використовуючи її для додатків, де простої та втрата даних є неприпустимими.
3. Відкритий код та безкоштовність. PostgreSQL є повністю безкоштовною та відкритою СУБД, що дозволяє економити на ліцензіях та надає можливість налаштовувати систему під конкретні потреби.
4. Висока продуктивність та масштабованість. PostgreSQL здатна ефективно працювати з великими обсягами даних та витримувати значні навантаження.

5. Повна відповідність стандартам SQL. PostgreSQL максимально дотримується стандартів SQL, що полегшує міграцію з інших систем.
6. Розширюваність. PostgreSQL має модульну архітектуру, яка дозволяє легко розширювати функціонал за допомогою додаткових розширень та плагінів.
7. Підтримка різноманітних типів даних, включаючи JSON, що особливо корисно для сучасних додатків.
8. Надійні механізми для забезпечення цілісності даних, включаючи транзакції ACID.
9. PostGIS є потужним розширенням для PostgreSQL, яке перетворює її на геопросторову базу даних. Це розширення надзвичайно корисне для симуляції руху транспорту та роботи з географічними координатами завдяки таким функціям:
   а) підтримка просторових об'єктів (точки, лінії, полігони) та операцій над ними;
   б) можливість індексації геопросторових даних для швидкого пошуку;
   в) функції для обчислення відстаней, перетинів, буферних зон та інших просторових відношень;
   г) вбудовані алгоритми для побудови маршрутів (наприклад, pgRouting як додаткове розширення);
   д) можливість виконання просторових запитів для відстеження та аналізу руху транспортних засобів у реальному часі;
   е) сумісність з різними форматами геоданих та GIS-системами.

4.2 Mock-сервіс

Для того щоб продемонструвати те, як працює система, треба щоб їй було з якими даними працювати. Оскільки відкритих GTFS API в Україні не було знайдено, було вирішено для спрощення розробки розробити свій Mock-сервіс, який би працював по протоколу GTFS, і віддавав би симуляцію даних переміщення

транспортного засобу по різним містам, тобто симулював активність громадського транспорту на карті.

Для втілення в життя цього сервісу, було використано мову програмування Python версії 3.12 і веб-фреймворк Fastapi, який також буде використовуватись для інших мікросервісних компонентів для більш консистентної архітектури.

База даних тут окрема, в тому же самому PostgreSQL кластері що і інші дані (див. секцію про Базу даних нижче).

Для взаємодії з базою даних використовується ORM SqlAlchemy і надбудова над нею SQLModel (з підтримкою сучасного методу типізації у Python). Це одна з двох найбільш популярних бібліотек ORM в екосистемі Python. Іншою є Django ORM, але її краще усього використовувати разом з веб-фреймворком Django.

Дані для початкової конфігурації мок-сервісу беруться з відкритого репозиторію [10]. Звідти був завантажений ZIP файл який в собі має дані у CSV форматі які були використані для того щоб зробити міграцію.

4.2.1 Міграції

Міграції це необхідний компонент кожної сучасної програмної системи яка взаємодіє з базами даних. Вони реалізують механізм апгрейду і даунгрейду схеми бази даних, відповідно до «файлів міграцій», які описують ітеративні зміни до таблиць у базі даних.

У випадку цієї системи було вирішено використовувати alembic. Він також дає можливість автоматичної генерації файлів міграцій на основі описаних ORM-моделей в SQLAlchemy.

4.2.2 Механізм роботи

Мок-сервіс має симулювати активність справжньої GTFS-системи. Для цього треба вміти будувати маршрути між точками на карті, і рухати віртуальні транспортні засоби по цим маршрутам, симулюючи зупинки.

4.3 Агрегатор даних з різних сервісів (gtfs-reader)

Система має бути масштабованою (scalable) у сенсі адекватної роботи незалежно від об'ємів вхідних даних (GTFS сервісів), і розширюваною для додавання нових даних.

Для втілення у життя цих потреб, був реалізований окремий компонент (gtfs-reader), який забезпечує динамічне розширення системи з точки зору додавання нових GTFS сервісів для споживання даних. Завдяки цьому компоненту система є масштабованою у тому сенсі, що можна незалежно збільшувати кількість gtfs-reader працівників (workers), які розподілятимуть між собою запити до різних GTFS сервісів та виводитимуть результати у лог-подібному форматі в Kafka, звідки інші сервіси можуть споживати ці дані та обробляти їх.

gtfs-reader має доступ до єдиної бази даних, в якій зберігається конфігурація для кожного підключеного GTFS сервісу. Модель цієї конфігурації виглядає так:

```
class Service(BaseModel):
    __tablename__ = "services"
    name = Column(String, nullable=False)
    api_base = Column(String, nullable=False)
    routes_fetched_at = Column(DateTime, nullable=False)
    stops_fetched_at = Column(DateTime, nullable=False)
    vehicles_fetched_at = Column(DateTime, nullable=False)
    trips_fetched_at = Column(DateTime, nullable=False)

    def __repr__(self):
        return       f"<Service(routes_fetched_at={self.routes_fetched_at},
stops_fetched_at={self.stops_fetched_at},
vehicles_fetched_at={self.vehicles_fetched_at},
trips_fetched_at={self.trips_fetched_at})>"
```

Наприклад, для того щоб підключитись до Мок-сервісу, треба зробити такий запис у базу даних:

```
INSERT INTO services (name, api_base, routes_fetched_at, stops_fetched_at,
vehicles_fetched_at, trips_fetched_at)
VALUES   ('mock-data-service',   'http://transportation-info-service:8000',
NULL, NULL, NULL, NULL);
```

Після цього, gtfs-reader буде періодично робити HTTP запити на різні ендпоінти у цього сервісу, брати звідти дані, і складати у відповідні топіки у Кафка.

Підтримку авторизації у сервісах було вирішено для демонстрації не робити.

Також, gtfs-reader реалізує адміністративний інтерфейс для завантаження статичних GTFS-даних до системи у певному, стандартизованому, форматі. Для реалізації адміністративного функціоналу використовувалась бібліотека SQLAdmin.

4.4 Комунікація між компонентами системи

У системі є поллінг (polling) частина (gtfs-reader воркери), які запитують статус у підключених GTFS систем. Їм треба передавати ці дані до клієнтів. Було вирішено використовувати Apache Kafka як евент-брокер через його високу швидкість роботи, яка потрібна враховуючи потенційний об’єм оновлень.

Конфігурація:

- топік: vehicle_updates;
- cleanup policy: Compact;
- size limit per topic: 1 GB.

Apache Kafka - це розподілена потокова платформа, яка забезпечує надійний механізм передачі повідомлень між компонентами системи. Вона працює за принципом підписки/публікації, де виробники (producers) публікують повідомлення в теми (topics), а споживачі (consumers) підписуються на ці теми та обробляють дані. Це ідеально підходить для системи GTFS, де gtfs-reader воркери публікують дані, а клієнтські сервіси їх споживають.

Висока пропускна здатність та низька затримка є ключовими перевагами Kafka, що робить її ідеальною для обробки потоків даних транспортної системи в реальному часі. GTFS системи генерують значні обсяги даних про розклади, маршрути та місцезнаходження транспортних засобів, які потрібно швидко передавати клієнтам. Kafka здатна обробляти мільйони повідомлень за секунду, що задовольняє вимоги до швидкодії системи.

Масштабованість Kafka дозволяє легко нарощувати потужності системи в міру збільшення кількості GTFS-сервісів або клієнтів. Кластер Kafka можна горизонтально масштабувати, додаючи нові вузли без переривання роботи системи. Це особливо важливо для систем моніторингу транспорту, де кількість

даних може стрімко зростати в години пік або при додаванні нових транспортних мереж.

Стійкість до відмов забезпечується механізмом реплікації даних між брокерами Kafka. Це гарантує, що дані не будуть втрачені навіть при виході з ладу окремих компонентів системи. Враховуючи критичність транспортних даних для кінцевих користувачів, така надійність є необхідною умовою для безперебійної роботи системи.

Політика компактування (Compact), вибрана для топіків, оптимально підходить для даних GTFS, оскільки дозволяє зберігати тільки останній стан для кожного ключа. Це особливо важливо для даних про місцезнаходження транспортних засобів, де актуальна лише найсвіжіша інформація. При цьому обмеження розміру в 1 ГБ на топік допомагає контролювати використання ресурсів сервера.

Модель споживання даних у Kafka дозволяє мати множину незалежних споживачів, що можуть обробляти дані з різною швидкістю. Це важливо для системи GTFS, де різні клієнтські сервіси можуть мати різні вимоги до обробки даних - від відображення на карті в реальному часі до аналітики та планування маршрутів.

4.5 Web Server

Веб-сервер також використовує Fastapi, і реалізує інтерфейс для веб-додатку для отримання GTFS даних для відображення на карті.

Він також використовує протокол WebSocket для real-time оновлення позицій транспортних засобів на карті у додатку.

Загалом для отримання даних використовується ORM SQLModel, але для деяких випадків, де потребується спеціалізований синтаксис, використовується «сирий» SQL.

4.6 WebSocket як протокол оновлення карти у веб-додатку

WebSocket - це протокол зв'язку, який забезпечує двосторонній канал передачі даних через одне TCP-з'єднання. На відміну від традиційного HTTP, де клієнт завжди ініціює запити, WebSocket дозволяє серверу активно надсилати дані клієнту без додаткових запитів, що є ідеальним для передачі оновлень про місцезнаходження транспортних засобів у реальному часі.

Низька затримка є ключовою перевагою WebSocket для відстеження руху транспорту. Після встановлення з'єднання дані передаються з мінімальною затримкою, що дозволяє користувачам веб-додатків бачити актуальне місцезнаходження автобусів, потягів чи інших транспортних засобів майже миттєво. Це критично важливо для планування поїздок та прийняття рішень у динамічному міському середовищі.

Ефективність використання ресурсів пристрою (який може бути мобільним, таким як ноутбук, або телефон) досягається завдяки тому, що WebSocket підтримує постійне з'єднання, усуваючи необхідність встановлювати нове з'єднання для кожного оновлення. Це зменшує навантаження на батарею мобільного пристрою та економить мобільний трафік, що особливо важливо для користувачів у русі.

Архітектурно WebSocket створює природний міст між Kafka та веб-клієнтом. Сервер може підписатися на відповідні топіки Kafka як споживач і передавати отримані оновлення безпосередньо підключеним клієнтам через WebSocket-з'єднання. Це забезпечує безперервний потік даних від системи GTFS до кінцевого користувача без складних проміжних перетворень.

Масштабованість рішення на основі WebSocket забезпечується можливістю балансування навантаження між кількома серверами та підтримкою кластеризації. При зростанні кількості користувачів додатку можна горизонтально масштабувати WebSocket-сервери, щоб задовольнити збільшений попит без втрати продуктивності.

Можливість фільтрації та персоналізації даних є важливою перевагою WebSocket для веб-додатків. Сервер може надсилати клієнту лише ті оновлення, які стосуються конкретних маршрутів чи транспортних засобів, якими цікавиться

користувач, замість передачі всього потоку даних з Kafka. Це додатково оптимізує використання мережевих ресурсів та покращує користувацький досвід.

4.7 Сервіс пошуку шляху

Для реалізації можливості будування маршрутів від однієї точки до іншої, потрібно реалізувати окремий компонент, який буде використовувати дані з transportation бази даних. Компонент має назву path-finder.

Щоб побудувати шлях від точки А до точки Б, треба враховувати усі варіанти побудування маршрутів (різні зупинки, різні маршрути, різні комбінації маршрутів), геометрію потенційного шляху, потенційний час затримки транспорту, поточну локацію користувача, і також поточну завантаженість доріг.

Для зберігання даних графу потенційних маршрутів і зупинок, було вирішено використати аддони до PostgreSQL: PostGIS і pgRouting для взаємодії з цими графовими гео-даними.

GTFS дані самі по собі не є достатніми для того щоб використовувати їх у PostGIS/pgRouting, тому треба спочатку зробити обробку цих даних. Отже, для обробки нам треба мати тільки статичні дані, що означає що ми можемо її робити “on-demand”, тобто після завантаження статичних даних. Для цього було реалізовано дві кнопки у адміністративній панелі: «Імпортувати GTFS дані», і «Перебудувати маршрутну схему».

4.7.1 Імпортування даних

Для реалізації імпорту використовується бібліотека gtfs_kit, що реалізує автоматичну генерацію SQL команд для вставки даних. Під час імпортування GTFS даних, дані перевіряються на валідність, і очищаються від некоректних даних:

```
def import_gtfs_static(self, feed: Feed):
    drop_invalid_columns(feed)
    drop_zombies(feed)
    connection = self.db.connection()
    shapes = gk.get_shapes(as_gdf=True, feed=feed)
    if shapes is None or shapes.empty:
        raise GTFSImportServiceError(
            GTFSImportServiceError.Code.NO_DATA,
            "No shapes found",
        )
```

Імпортування даних це довгий процес, і тому синхронне виконання може викликати “504 Gateway Timeout” в адміністративній панелі, тому було вирішено делегувати такого роду задачі у системі яка реалізує розподілену чергу завдань (Distributed task queue), що дозволяє виконувати такі задачі у фоновому режимі.

Для цього проекту був використаний Celery. Делегація виконання завдань (функції) виглядає так:

```
import_gtfs_static.delay(
      str(fs_path),
)
```

4.7.2 Підготовка маршрутних даних

Кнопка «Перебудувати маршрутні дані» в адміністративній панелі починає Celery задачу «rebuild_routing_data»:

```
@expose("/rebuild-routing", methods=["GET", "POST"])
async def rebuild_routing_page(self, request: Request):
    ...
    if request.method == "POST":
        rebuild_routing_data.delay()
        return await self.templates.TemplateResponse(
            request, "rebuild_routing_data_started.html"
        )
```

Можна детальніше описати процес підготовки даних.

1. Спочатку треба видалити старі маршрутні дані з таблиць.
2. Треба побудувати routing_nodes використовуючи дані з таблиці stops – вони будуть нашими вузлами.
3. Використовуючи дані з stop_times, stops, і routing_nodes, треба побудувати routing_edges – вони будуть ребрами графу.
    a. для кожного з цих ребр також треба визначити суб-геометрію (послідовність точок у полярній системі координат), щоб потім можна було легко відображати її на мапі під час будування маршруту;
4. Побудувати дані для таблиці walking_connections, щоб мати у базі даних такі ребра, які відображають зупинки, між якими достатньо маленька відстань, щоб її можна було пройти пішки. Максимальну відстань між зупинками для цього було вирішено встановити на рівні 800 метрів.

4.7.3 Опис алгоритму пошуку шляху між зупинками

З вхідних даних ми маємо:

– маршрути, які обозначають зв’язки між зупинками (ребра графу) – routing_edges;

– самі зупинки, які інтерпретуються як вузли графу - stops;

– таблиця stop_times, яку можна використовувати для визначення часу прибуття транспортних засобів до конкретних зупинок.

Для обробки запиту побудування маршруту ми отримаємо наступні дані від користувача:

– зупинка А – зупинка з якої треба почати будувати маршрут;

– зупинка Б – кінцева зупинка.

Як результат виконання алгоритму маємо віддати масив, елементом якого є частина маршруту, якою треба скористатися, щоб просунутись до кінцевої точки:

```
class RoutePart(BaseModel):
    seq: int
    route_id: str | None
    stop_id: StopId | None
    stop_name: str | None
    route_long_name: str | None
    distance: float | None
    edge_type: str | None
    stop: Stop
    stop_geometry: PointGeometry | None
    duration: datetime.timedelta | None
    geometry: LineStringGeometry | None
    instruction: str | None
```

Для побудування самого маршруту використовується комплексний SQL запит, який будується на виклику вбудованої у pgRouting функції pgr_dijkstra:

```
raw_path AS (
    SELECT
        path.seq,
        path.node,
        path.edge,
        path.cost,
        path.agg_cost
    FROM pgr_dijkstra(
        'SELECT edge_id as id, source_node as source, target_node as target,
cost FROM routing_edges WHERE is_active = true',
        (SELECT node_id FROM start_node),
        (SELECT node_id FROM end_node),
        directed := true
    ) AS path
    WHERE path.edge IS NOT NULL
```

),

4.8 Фронтенд (клієнтська) частина

Як фреймворк для розробки фронтенд частини було прийнято використовувати React з мовою програмування TypeScript і фреймворком Next.js. Next.js реалізує «бандлінг» додатку для різних платформ (версій браузерів), SSR, і маршрутизацію на сторінках. React - це JavaScript-бібліотека для побудови користувацьких інтерфейсів, яка використовує компонентний підхід та віртуальний DOM для ефективного оновлення UI.

Реактивність та декларативність React забезпечують миттєве відображення змін у даних про рух транспорту. Коли від WebSocket з'єднання надходять оновлення про місцезнаходження автобусів або потягів, інтерфейс автоматично перемальовується, відображаючи актуальну інформацію без додаткового коду для маніпуляції DOM чи нативними елементами інтерфейсу.

Компонентна архітектура React сприяє повторному використанню коду та спрощує розробку складних інтерфейсів. Наприклад, компонент відображення маршруту або карти з транспортними засобами може використовуватися як у веб-версії, так і в мобільному додатку з мінімальними змінами. Це прискорює розробку та полегшує підтримку коду в довгостроковій перспективі.

Велика екосистема готових компонентів та бібліотек для React [11] значно прискорює розробку GTFS-клієнта. Існують готові карти, інструменти візуалізації даних, компоненти для роботи з WebSocket та багато іншого. Це дозволяє зосередитися на бізнес-логіці додатку, а не на розробці базових функціональних можливостей.

Для рендерингу мапи, використовується бібліотека Openlayers.

Оптимізована продуктивність React забезпечується завдяки віртуальному DOM, який мінімізує кількість операцій з реальним DOM, а React-Native використовує нативні компоненти для рендерингу, що гарантує високу продуктивність веб-додатку. Це критично важливо для додатку, який постійно отримує та відображає оновлення про рух транспорту в реальному часі.

Швидкий цикл розробки та Hot Reloading, реалізований у Next.js, значно прискорюють процес створення та тестування додатку. Розробники можуть бачити зміни в інтерфейсі майже миттєво після їх внесення, не перезапускаючи додаток повністю, що особливо цінно при налагодженні складних інтерактивних елементів, таких як карти руху транспорту.

Як метод взаємодії з бекендом, використовується бібліотека Axios. Для менеджменту стану у браузері, використовується бібліотека @tanstack/react-query [12], що забезпечує вбудоване кешування, обробку помилок, і обробку різних станів завантаження даних.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Тестування веб-застосунку

Для тестування веб-застосунку відстеження руху громадського транспорту, використовувалось мануальне тестування і також «юніт» тести. Основними підвидами мануального тестування є як функціональне, так і нефункціональне, так як у front-end розробці потрібно враховувати обидва види тестування.

Мануальне тестування виконувалось для перевірки функціоналу веб-сторінки, і загалом включало у себе запуск всього бекенду, бази даних, і веб-серверу локально. Мануальним тестуванням було перевірено основні послідовності використання веб-додатку, такі як побудування маршруту (табл. 5.1), і відстеження оновлення транспортних засобів (табл. 5.2.). Для коректної перевірки функціоналу треба реалізувати необхідні передумови, які приводять начальний стан клієнтської системи до однакового стану, а саме очищення стану браузеру, який відноситься до веб-додатку: скидання браузерних cookie-файлів, очищення системи зберігання клієнтських даних LocalStorage, і очищення сесії. Також, для забезпечення покриття більшої кількості Use-case-ів користувачів, мануальне тестування проводилось на усіх підтримуємих платформах (браузерах), і різних розмірах екранів, для перевірки адаптивності і коректного відображення елементів веб-інтерфейсу.

Юніт тестування було виконано тільки для бекенд частини для автоматизації тестування і швидкого визначення нових багів і проблем у системі. Як приклад юніт тесту наведено кейс виконання симуляції переміщення транспортних засобів у мок-сервісі transportation-info-service (тест-кейс №3).

З інструментів для реалізації системи автоматизованого тестування для бекенду, було використано популярний фреймворк для юніт тестів в екосистемі Python pytest. Також, Fastapi надає додаткові функції для більш зручного тестування саме end-point-ів.

Також, для перевірки взаємодії додатку з БД, було розроблено кілька інтеграційних тестів, , і дозволяють перевірити інтеграцію компонентів.

Використання цих трьох методів тестування надає можливість запобігти проблем одразу після завантаження змін до кодової бази.

Таблиця 5.1 – Тест-кейс №1 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №1		
Опис функції:	Побудування маршруту		
Власник тесту:	Гузєєв Дмитро Михайлович		
Дата створення:	31.05.2025		
Мета тесту:	Перевірити коректність реалізації основного варіанту використання системи		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	Відкрити веб-застосунок	Користувач має доступ до сайту, який відкритий	Пройдено
Пошук необхідної зупинки			
№	Опис випадку	Очікуваний результат	Висновок
1	Вибрати пошук	Пошук сфокусований і показується історія пошуку	Пройдено
2	Ввести пошуковий запит	Показується результат пошуку – зупинки і маршрути які в назві або описі мають текст запиту	Пройдено
3	Обрати одну з зупинок	На мапі була вибрана відповідна зупинка, і пошук змінився на інформацію про маршрути для зупинки	Пройдено
Результати тестування			

Кінець таблиці 5.1

1	Натиснути на кнопку «Побудувати маршрут від»	Відкрився екран побудування маршруту, де у полі «Початок» вибрана зупинка з попереднього екрану	Пройдено
2	Натиснути на поле пошуку «Кінець», і ввести назву кінцевої зупинки	З’явився список результатів пошуку зупинок, які підходять до вказаного запиту. У результатах пошуку є тільки зупинки.	Пройдено
3	Натиснути на результат пошуку	Був побудований маршрут між двома зупинками, і він показується на карті.	Пройдено
Результати тестування			
Тестувальник: Гузєєв Д.М.	Дата прогону тесту: 31.05.2025	Результат тесту (P/F/B): ПРОЙДЕНО (P)	

Таблиця 5.2 – Тест-кейс №2 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №2		
Опис функції:	Коректне відображення транспортних засобів на мапі		
Власник тесту:	Гузєєв Дмитро Михайлович		
Дата створення:	31.05.2025		
Мета тесту:	Перевірити коректність оновлення і рендерингу позиційної інформації про транспортні засоби для вибраного маршруту		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок

Кінець таблиці 5.2

1	Відкрити веб-застосунок	Користувач має доступ до сайту, який відкритий	Пройдено
Вибір маршруту			
№	Опис випадку	Очікуваний результат	Висновок
1	Вибрати пошук	Пошук сфокусований і показується історія пошуку	Пройдено
2	Ввести пошуковий запит з метою знайдення маршруту	Знайдено як мінімум один маршрут, і відповідний результат пошуку показується у списку результатів під пошукової стрічкою	Пройдено
3	Обрати один з маршрутів	На мапі був вибраний і сфокусований відповідний маршрут, і з затримкою у декілька секунд з’явились індикатори транспортних засобів, що обслуговують цей маршрут	Пройдено
4	Почекати до 30 секунд	Позиції транспортних засобів були оновлені на мапі	Пройдено
5	Навести на один з транспортних засобів	Відображається інформація про ідентифікатор транспортного засобу, і також час останнього оновлення	Пройдено
Результати тестування			
Тестувальник: Гузєєв Д.М.	Дата прогону тесту: 31.05.2025	Результат тесту (P/F/B): ПРОЙДЕНО (P)	

Приклад юніт тесту 5.3 – Тест-кейс №3

Цей юніт тест перевіряє роботу алгоритму емуляції руху транспорту у transportation-info-service. Для повної версії тесту див. Додаток В.

```
mock_current_stop = Mock(stop_id="stop3", stop_lat=2.0, stop_lon=2.0)
mock_next_stop = Mock(stop_id="stop4", stop_lat=3.0, stop_lon=3.0)
with patch.object(
    vehicle_position_service.stop_service,
    "find_closest_stop",
    return_value=(mock_current_stop, mock_next_stop),
):
    # Act
    result = vehicle_position_service.advance_vehicle_state(
        sample_vehicle, sample_vehicle_state, current_time
    )

    assert not result.is_at_stop
    assert result.current_stop_id == StopId("stop3")
    assert result.next_stop_id == StopId("stop4")
    assert result.next_stop_position == (3.0, 3.0)
    assert result.start_time == current_time
    assert result.stop_start_time is None
```

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи створено масштабований програмний веб-сервіс для відстеження руху міського транспорту з інтеграцією GTFS-систем. Розроблена система вирішує поставлені завдання: проведено аналіз існуючих рішень, розроблено ефективну архітектуру, створено модуль імпорту та додаткової обробки GTFS-даних, реалізовано алгоритм побудови маршрутів та зручний користувацький інтерфейс.

Під час виконання роботи було також проведено аналіз найбільш популярних існуючих рішень. Обґрунтовано вибір технологічного стеку, що включає React.js з Next.js для фронтенду, Python і Fastapi для бекенду та PostgreSQL для зберігання даних, і PostgreSQL з плагінами pgRouting і PostGIS загалом як метод вирішення задач на маршрутизацію.

Створено інтуїтивний користувацький інтерфейс з адаптивним дизайном, що забезпечує комфортне використання на різних пристроях. Інтерфейс включає інтерактивну карту, пошук маршрутів та можливість будувати маршрути.

Реалізовано систему відстеження транспорту в реальному часі з використанням WebSocket-з'єднань, що забезпечує миттєве оновлення інформації про місцезнаходження транспортних засобів.

Застосунок демонструє високу продуктивність та стабільність при навантаженнях, забезпечуючи швидку побудову маршрутів навіть при складних сценаріях з пересадками. Універсальність рішення дозволяє адаптувати його для різних міст без додаткової розробки.

Практична цінність підтверджується можливістю використання у щоденному житті містян, туристичній галузі та для транспортних компаній. Веб-додаток можна використовувати для того, щоб отримувати інформацію про конфігурацію транспортної мережі у будь якому підключеному місті. За його допомогою також можна планувати подорожі і щоденне пересування по місту.

Завдяки розширяємій і масштабуємій архітектури, у перспективі можна досить просто додавати новий функціонал, такий як підтримку мобільних додатків, працюючих нативно (тобто без браузерної прослойки). З плануємих функцій,

можна перечислити імплементацію додаткових стандартів даних, інтеграцію з іншими сервісами «розумного міста», реалізацію авторизації для синхронізації стану і налаштувань між девайсами; реалізацію push-сповіщень, і загалом відображення інформації щодо затримки транспорту.

Для полегшення впровадження застосунку, у перспективі також треба налаштувати CI/CD pipeline, який дозволив би при завантаженні змін коду у репозиторій, одразу отримувати автоматичний «фідбек» відносно якості коду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Automatic vehicle location [Електронний ресурс] – URL: https://en.wikipedia.org/wiki/Automatic_vehicle_location (дата звернення: 31.05.2025)
2. Essential Google Maps Statistics & Trends to Watch in 2025 [Електронний ресурс] – URL: https://www.loopexdigital.com/blog/google-maps-statistics#:~:text=Google%20Maps%20is%20immensely%20popular,Maps%20Platform%20core%20products%20weekly (дата звернення: 31.05.2025)
3. EasyWay [Електронний ресурс] – URL: https://www.eway.in.ua/ (дата звернення: 31.05.2025)
4. GTFS Static Specification [Електронний ресурс] – URL: https://developers.google.com/transit/gtfs (дата звернення: 31.05.2025)
5. SIRI [Електронний ресурс] – URL: https://transmodel-cen.eu/index.php/siri/ (дата звернення: 31.05.2025)
6. NetEX [Електронний ресурс] – URL: https://transmodel-cen.eu/index.php/netex/ (дата звернення: 31.05.2025)
7. Transexchange [Електронний ресурс] – URL: https://www.gov.uk/government/collections/transxchange (дата звернення: 31.05.2025)
8. Міста України (за населенням) [Електронний ресурс] – URL: https://uk.wikipedia.org/wiki/%D0%9C%D1%96%D1%81%D1%82%D0%B0_%D0%A3%D0%BA%D1%80%D0%B0%D1%97%D0%BD%D0%B8_%28%D0%B7%D0%B0_%D0%BD%D0%B0%D1%81%D0%B5%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC%29?utm_source (дата звернення: 31.05.2025)
9. What are database migrations? [Електронний ресурс] – URL: https://www.prisma.io/dataguide/types/relational/what-are-database-migrations (дата звернення: 31.05.2025)
10. Відкриті GTFS статичні дані міста Львів [Електронний ресурс] – URL: https://github.com/vasnab/lviv-gtfs (дата звернення: 31.05.2025)

11. A collection of awesome things regarding the React ecosystem [Електронний ресурс] – URL: https://github.com/enaqx/awesome-react (дата звернення 31.05.2025)
12. Бібліотека для стейт-менеджменту для React [Електронний ресурс] – URL: https://tanstack.com/query/latest (дата звернення: 31.05.2025)
13. Newman S. Building Microservices: Designing Fine-Grained Systems / S. Newman. – 2nd ed. – O'Reilly Media, 2021. – 624 p.
14. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems / M. Kleppmann. – O'Reilly Media, 2017. – 616 p.
15. Banks A. Learning React: Modern Patterns for Developing React Apps / A. Banks, E. Porcello. – 2nd ed. – O'Reilly Media, 2020. – 310 p.
16. Петін В. А. Веб-програмування: сучасні технології розробки інтернет-додатків / В. А. Петін. – Х. : Фоліо, 2022. – 512 с.
17. Мельник О. В. Методи та засоби побудови розподілених веб-систем реального часу : дис. ... канд. техн. наук : 05.13.06 / О. В. Мельник. – К., 2022. – 187 с.
18. Apache Kafka Documentation [Електронний ресурс] – URL: https://kafka.apache.org/documentation/ (дата звернення: 31.05.2025)
19. PostgreSQL Documentation [Електронний ресурс] – URL: https://www.postgresql.org/docs/ (дата звернення: 31.05.2025)
20. GitHub репозиторій [Електронний ресурс] – URL: https://github.com/comonadd/public-transportation-app-final (дата звернення: 09.06.2025)