НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ **Ігор БОЛБОТ**	_____ **Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення для системи картографічної платформи з візуалізацією даних**»**

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми к.т.н., доцент	_____ (підпис)	**Ганна ВАЙГАНГ**
Керівник бакалаврської кваліфікаційної роботи к.ф.-м.н., доцент	_____ (підпис)	**Віктор Кириченко**
Виконав	_____ (підпис)	**Ігор Ковальчук**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Ковальчуку Ігору Руслановичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи

«Програмне забезпечення для системи картографічної платформи з візуалізацією даних»

затверджена наказом від «19» грудня 2025 р. № 3204 «С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): Інформаційні джерела з веброзробки, технологій Node.js, React, PostgreSQL, картографічних рушіїв (Leaflet, OSRM), інтеграції зовнішніх API (Open-Meteo, TomTom), технології WebSockets (Socket.IO) та UML-моделювання.
Перелік питань, що підлягають дослідженню: Аналіз предметної області картографічних платформ; формування вимог до системи; UML-моделювання; проєктування реляційної бази даних PostgreSQL; розробка серверної та клієнтської частини вебзастосунку; інтеграція із зовнішніми геосервісами та YouTube API; алгоритмічна реалізація побудови маршрутів, збагачення даних та механіки фокус-сесій; тестування та оцінка ефективності системи.
Перелік графічних документів (за необхідності).

Дата видачі завдання «19» грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Віктор Кириченко**

Завдання взяв до виконання _____ (підпис) **Ігор Ковальчук**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) - прикладний програмний інтерфейс.

БД - база даних.

СУБД - система управління базами даних.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, заснований на синтаксисі JavaScript.

OSRM (Open Source Routing Machine) - високопродуктивний рушій маршрутизації для найкоротших шляхів у дорожніх мережах.

REST (Representational State Transfer) - архітектурний стиль для розподілених гіпермедійних систем.

SPA (Single Page Application) - односторінковий вебзастосунок.

SQL (Structured Query Language) - мова структурованих запитів.

UML (Unified Modeling Language) - уніфікована мова моделювання, яка використовується для візуалізації, документування та проєктування програмних систем.

ВСТУП

Актуальність теми. Зазвичай під час планування будь-якої поїздки користувачам доводиться відкривати відразу декілька вкладок або застосунків. В одному сервісі ми прокладаємо маршрут, у другому - шукаємо інформацію про затори, а десь інакше перевіряємо погоду по дорозі. Це створює незручності та розпорошує увагу. Саме тому виникла ідея розробити єдину картографічну вебплатформу TransitMind. Її головна перевага в тому, що вона об'єднує всі ці функції в одному вікні. Користувач не просто бачить лінію маршруту на карті, а відразу отримує прогноз погоди вздовж неї, оцінку завантаженості доріг та навіть може запустити локальну сесію з таймером фокусування. Розробка такого програмного забезпечення є дійсно актуальним завданням, оскільки готових рішень, які б поєднували маршрутизацію з візуалізацією погоди і тайм-менеджментом у такому легкому форматі без зайвих соціальних функцій, наразі бракує.

Мета дослідження полягає у розробці програмного забезпечення картографічної вебплатформи, яке дозволяє будувати та візуалізувати маршрут, відображати просторові дані на карті, а також виводити додаткові показники (погоду та статистику часу в дорозі) у спеціальній інтерактивній панелі.

Для досягнення цієї мети необхідно було виконати такі **завдання**:

- розібратися з предметною областю і чітко прописати вимоги до майбутньої платформи;
- спроєктувати базу даних та створити необхідні UML-діаграми, щоб зрозуміти логіку взаємодії компонентів;
- написати серверну частину (яка працюватиме як агрегатор) та клієнтський інтерфейс, поєднавши різні API: OSRM для розрахунку шляху, Open-Meteo для погоди та TomTom для трафіку;
- імплементувати алгоритми пошуку оптимального шляху (зокрема, застосувати алгоритм Дейкстри для побудови маршрутів на внутрішньому графі) та додати функцію симуляції руху автомобіля;

- перевірити готову систему на помилки та скласти рекомендації щодо її використання.

Об’єкт дослідження - процес візуалізації просторових і тематичних даних у web GIS-подібному застосунку.

Предмет дослідження - методи інтеграції зовнішніх геосервісів (маршрутизація, погода, трафік) та засоби їхньої графічної подачі користувачу.

Методи дослідження. У процесі роботи над проєктом застосовувалися різні підходи. Для розрахунку маршрутів між містами використовувалися методи теорії графів, а саме пошук найкоротшого шляху на зваженому графі. Щоб об'єднати дані від кількох різних REST API на бекенді, застосовувалися принципи системної агрегації. Крім того, знадобилися методи візуальної аналітики для побудови графіків температури (у вигляді компактних SVG-ліній) і діаграм трафіку безпосередньо в панелі застосунку.

Апробація результатів. Результати розробки та ключові архітектурні рішення були представлені під час виступу на науковій конференції. Тема доповіді: «Розробка архітектури та програмного забезпечення системи пошуку маршрутів та відстеження транспорту в реальному часі "Transit Mind"».

Структура роботи. Пояснювальна записка складається зі вступу, чотирьох розділів та висновків. Перший розділ присвячений огляду предметної області та формуванню вимог до застосунку. У другому розділі описано процес проєктування системи та створення логічних моделей. Третій розділ містить деталі технічної реалізації: від налаштування бази даних PostgreSQL до написання коду на React і Node.js та інтеграції зовнішніх API. Четвертий розділ описує тестування готового продукту та рекомендації щодо його розгортання.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Коли людина планує подорож між містами або просто довгу поїздку, їй зазвичай потрібна не лише лінія на карті. Важливо знати, яка буде погода по дорозі, чи є там затори і скільки реально часу займе цей шлях. Більшість сучасних картографічних сервісів чудово будують маршрути, але вони часто не дають цього додаткового контексту в одному зручному вікні. Щоб перевірити прогноз погоди чи знайти ідеальний час для виїзду, доводиться перемикатися між різними сайтами. Саме ця проблема і формує предметну область нашого проєкту.

Ми працюємо на стику веброзробки, геоінформаційних систем (ГІС) та агрегації даних. Основна концепція, навколо якої будується логіка системи - це так зване «збагачення маршруту» (route enrichment). У цій предметній області програмна система має виступати єдиною точкою, яка бере геометрію шляху (полілінію) і накладає на неї зовнішні дані середовища. Наприклад, ми підтягуємо погоду від сервісу Open-Meteo та статистику заторів від TomTom.

Крім того, предметна область нашого проєкту охоплює алгоритми пошуку оптимальних шляхів на графах (зокрема алгоритм Дейкстри для пошуку між містами). Важливою частиною є також психологічний аспект роботи за комп'ютером - створення спеціального режиму фокусування. Тобто система не лише показує карту, а й дозволяє запустити локальну «подорож» із таймером та анімацією руху, щоб користувач міг зосередитися на своїх завданнях, поки його віртуальний транспорт рухається до мети.

1.2 Аналіз вимог до програмної системи

Щоб картографічна платформа TransitMind працювала так, як задумано, ми розділили вимоги до неї на функціональні (що вона має вміти) та нефункціональні (як саме вона має працювати).

Функціональні вимоги Це ті функції, які користувач безпосередньо бачить і з якими взаємодіє у браузері:

1. **Побудова та відображення маршруту:** Користувач повинен мати змогу обрати міста або довільні точки, а система - намалювати лінію маршруту на інтерактивній карті. Для цього використовується бібліотека Leaflet та сервіс маршрутизації OSRM.
2. **Погода в дорозі:** Програма має автоматично брати контрольні точки з маршруту (до 5 точок) і підтягувати для них поточний прогноз (температуру, опади, хмарність).
3. **Аналіз заторів та часу:** Потрібно показувати порівняння часу в дорозі (вільний рух, історичні дані та реальний трафік) у вигляді спеціальних горизонтальних смуг. Також потрібен стовпчиковий графік, який покаже, скільки часу займе поїздка, якщо виїхати в різний час «завтра».
4. **Інтерактивна бічна панель:** Всі метрики маршруту мають виводитись у зручній панелі (drawer), що розгортається збоку. Температуру вздовж траєкторії слід показувати як компактний SVG-графік без осей (sparkline).
5. **Режим фокус-сесії:** Користувач повинен мати можливість натиснути «Старт». Після цього відкривається повноекранна карта з таймером, а маркер починає рухатися вздовж маршруту пропорційно до часу сесії.
6. **Двомовність:** Інтерфейс та легенди на графіках повинні підтримувати перемикання між українською та англійською мовами.

Нефункціональні вимоги Ці вимоги гарантують, що система буде стабільною, безпечною та швидкою:

1. **Безпека ключів API:** Усі запити до платних або закритих сервісів (наприклад, TomTom) мають іти виключно через наш сервер (backend на Node.js). Клієнтський браузер не повинен мати доступу до цих ключів.

2. **Надійність (fallback):** Якщо підключення до TomTom відсутнє або немає API ключа, система не повинна видавати помилку чи блокувати роботу. Замість цього вона має використовувати просту алгоритмічну евристику - рахувати орієнтовне навантаження доріг на основі відстані та середньої швидкості з даних OSRM.
3. **Агрегація запитів (швидкодія):** Щоб не перевантажувати клієнтський браузер, фронтенд має робити лише один комбінований HTTP-запит. Бекенд своєю чергою паралельно збирає погоду, метрики трафіку та швидкість, і повертає готовий збагачений пакет даних.
4. **Простота та незалежність:** Програма повинна працювати без складних систем реєстрації, обов'язкових паролів чи соціальних чатів. Архітектура бази даних має бути мінімалістичною (без зайвих таблиць користувачів) і розрахованою на одного локального користувача.

1.3 Моделювання предметної області

Моделювання предметної області для програмного забезпечення картографічної платформи TransitMind є ключовим етапом проєктування, що допомагає створити зрозумілу та ефективну архітектуру. Основною метою моделювання є розробка концептуальної моделі, яка представляє сукупність логічних об'єктів і їх взаємодій. Оскільки наша система орієнтована на індивідуальне використання (без складних ієрархій користувачів чи адміністративних ролей), моделювання дозволяє чітко сфокусуватися на єдиному потоці дій локального клієнта. Для візуалізації логіки роботи застосунку використано інструментарій UML (Unified Modeling Language), зокрема три основні діаграми: варіантів використання, послідовності та діяльності.

1.3.1 Діаграма варіантів використання. Ця діаграма візуально описує взаємодію між системою та її зовнішніми користувачами (акторами). Вона наочно демонструє, які саме функції платформи доступні для виконання. Основні елементи такої діаграми включають:

- **Актор:** зовнішня сутність, яка взаємодіє з системою. Зображується у вигляді стилізованої фігурки людини.
- **Варіант використання (прецедент):** опис певної функціональної можливості системи (те, що система робить для актора). Зображується у вигляді еліпсів.
- **Зв'язок:** лінія, що показує взаємодію між актором та прецедентом (наприклад, асоціація або зв'язок включення include, коли один процес обов'язково вимагає виконання іншого).

Для програмного забезпечення картографічної платформи TransitMind було сформовано діаграму варіантів використання, що зображена на рис. 1. [1]

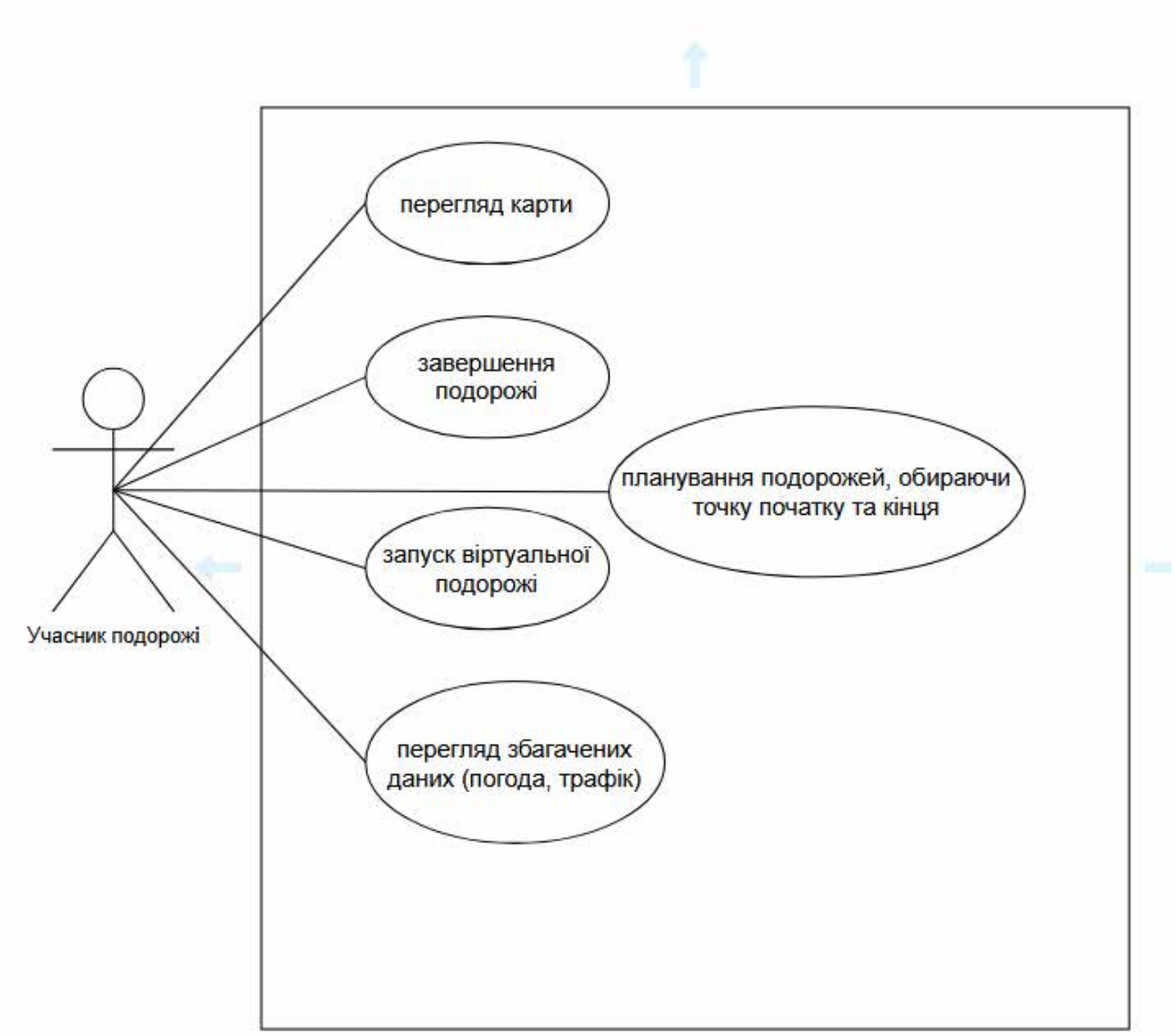


Рис. 1 Діаграма варіантів використання

Під час проєктування було визначено головного актора системи. Оскільки платформа розроблена як індивідуальний інструмент без соціальних мереж та ієрархії доступів, актор у системі лише один. Його опис зведено у табл. 1.1.

Таблиця 1.1

Визначені актори системи

Актор	Короткий опис
Користувач	Єдина дійова особа в системі, яка здійснює індивідуальне планування подорожей. Має доступ до всього доступного функціоналу: побудови маршрутів на інтерактивній карті, перегляду агрегованої аналітики (погоди та трафіку), а також ініціалізації локальних фокус-сесій.

Для визначеного актора було виявлено усі можливі варіанти використання системи. Кожному з них дано коротке формулювання. Виявлені прецеденти та їх опис зведено у табл. 1.2.

Таблиця 1.2

Варіанти використання системи

Актор	Найменування	Формулювання
Користувач	Перегляд інтерактивної карти	Дозволяє актору взаємодіяти з картографічною основою (масштабувати, панорамувати, змінювати базові шари)
Користувач	Побудова маршруту	Дозволяє актору обрати початкову та кінцеву точки, ініціювавши

		розрахунок геометрії шляху
Користувач	Перегляд збагачених даних (погода, трафік)	Дозволяє актору переглядати контекстні метеорологічні та транспортні показники (цей прецедент логічно включає в себе побудову маршруту)
Користувач	Запуск фокус-сесії (симуляції)	Дозволяє актору розпочати віртуальну подорож із таймером та анімацією руху вздовж заданої полілінії

1.3.2 Діаграма послідовності. Дана діаграма є важливим інструментом для представлення логіки обміну повідомленнями між компонентами системи. Вона демонструє, як концептуальні об'єкти взаємодіють між собою в хронологічному порядку протягом виконання базового сценарію. Основні елементи діаграми послідовності:

- **Лінія життя (Lifeline):** представляє існування об'єкта під час виконання сценарію (відображається як вертикальна пунктирна лінія).
- **Активаційна скринька (Activation Box):** вертикальна смуга, що вказує на період, протягом якого об'єкт виконує певну операцію.
- **Повідомлення (Message):** горизонтальні стрілки, що передають запити або відповіді (повернення даних) між об'єктами.

Для програмного забезпечення TransitMind було сформовано діаграму послідовності для процесу розрахунку та збагачення маршруту. [2] Вона зображена на рис. 2.

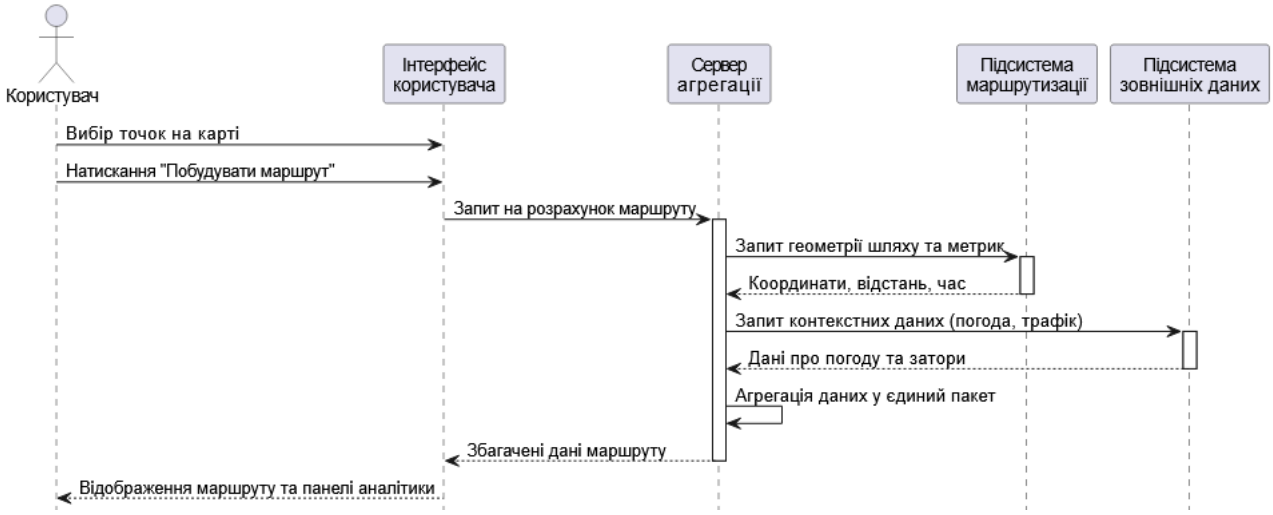


Рис. 2 Діаграма послідовності

На розробленій діаграмі містяться абстрактні логічні об'єкти: «Користувач», «Інтерфейс користувача», «Сервер агрегації», «Підсистема маршрутизації» та «Підсистема зовнішніх даних». Послідовність взаємодій відбувається наступним чином:

1. Користувач взаємодіє з Інтерфейсом, обираючи точки на карті та подаючи команду на побудову маршруту.
2. Інтерфейс відправляє системний запит на розрахунок шляху до Сервера агрегації.
3. Сервер агрегації виконує роль проксі-вузла. Він звертається до Підсистеми маршрутизації для отримання точної геометрії шляху, загальної відстані та базового часу.
4. Отримавши координати, Сервер агрегації ініціює паралельні запити до Підсистеми зовнішніх даних, щоб зібрати контекст (погодні умови та стан трафіку вздовж лінії маршруту).
5. Після отримання відповідей, Сервер агрегації формує єдиний збагачений пакет даних і повертає його до Інтерфейсу.
6. На фінальному етапі Інтерфейс відмальовує полілінію на карті та розгортає панель аналітики з графіками для Користувача.

1.3.3 Діаграма діяльності. Діаграма діяльності використовується для відображення динамічної поведінки системи та моделювання робочих процесів

(потоку виконання від однієї дії до іншої). Основні компоненти діаграми діяльності:

- **Діяльності (Activities):** компоненти, що представляють поведінку (конкретні дії системи або користувача).
- **Перегородки (Swimlanes):** засіб візуального групування дій за зонами відповідальності (наприклад, що робить користувач, а що — система).
- **Розгалуження (Умови):** вузли у формі ромба, де потік виконання може піти різними шляхами залежно від прийнятого рішення.

Для відображення алгоритму роботи користувача з картографічною платформою було сформовано діаграму діяльності, зображену на рис. 3. [3]

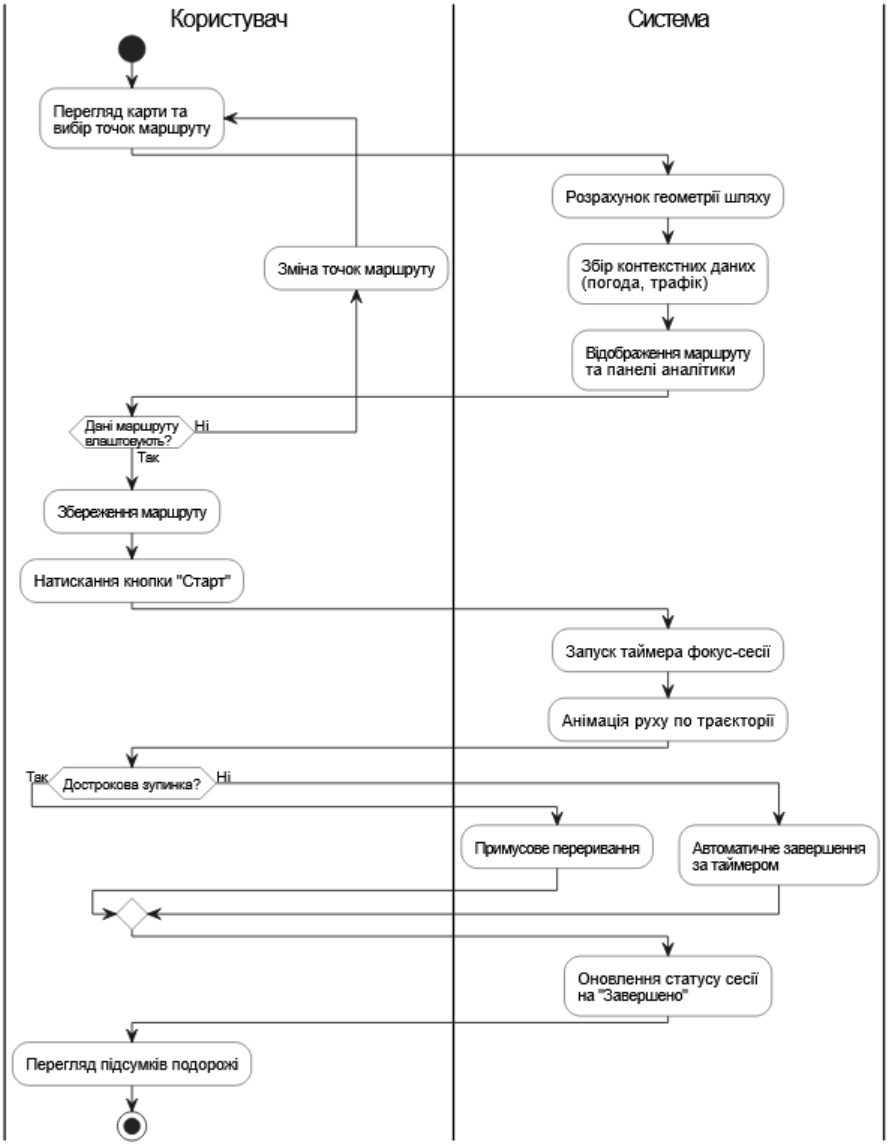


Рис. 3 Діаграма діяльності

Ця модель візуалізує повний цикл роботи, розподілений на дві смуги відповідальності: «Користувач» та «Система». Алгоритм містить такі ключові кроки та розгалуження:

- **Початковий етап:** Користувач обирає бажані точки призначення на інтерактивній карті.
- **Обробка системою:** Відбувається математичний розрахунок геометрії та збір контекстних даних середовища (трафік, погода), після чого результати виводяться в аналітичній панелі.
- **Перше розгалуження (Цикл вибору):** Система очікує рішення користувача. Якщо запропонований маршрут або рівень заторів його не влаштовує, потік повертається на етап зміни координат. Якщо умови прийнятні, відбувається збереження даних.
- **Етап фокус-сесії:** Користувач ініціює старт. Система запускає таймер та анімує переміщення маркера вздовж згенерованої полілінії.
- **Друге розгалуження (Умови завершення):** Процес симуляції може бути зупинений користувачем достроково (примусове переривання) або ж завершиться природним шляхом по закінченню таймера.
- **Фінал:** В обох випадках система фіксує статус сесії як «Завершено», після чого надає користувачу підсумкову статистику.

1.4 Огляд інформаційних джерел та існуючих рішень

1.4 Огляд інформаційних джерел та існуючих рішень

Сьогодні існує велика кількість застосунків для навігації, проте більшість із них сфокусована на виконанні однієї базової функції — проведенні маршруту з точки А в точку Б. Водночас зростає потреба у платформах, які здатні не просто вказувати шлях, а й надавати розширений контекст середовища (збагачення маршруту). Користувачам зручно планувати подорожі з урахуванням кліматичних умов та часових рамок в одному вікні, без перемикання між різними

вебсторінками. Нижче наведено аналіз популярних програмних рішень, які частково перекривають функціонал розроблюваної платформи TransitMind.

Google Maps - одна з найвідоміших та найпотужніших картографічних платформ у світі. Вона пропонує деталізовану інформацію про місцевість, організації, заклади та дозволяє будувати маршрути для різних видів транспорту. Система використовує величезний масив даних для розрахунку орієнтовного часу прибуття та відображення заторів у реальному часі. Інтерфейс даної платформи зображено на рис. 4. [4]

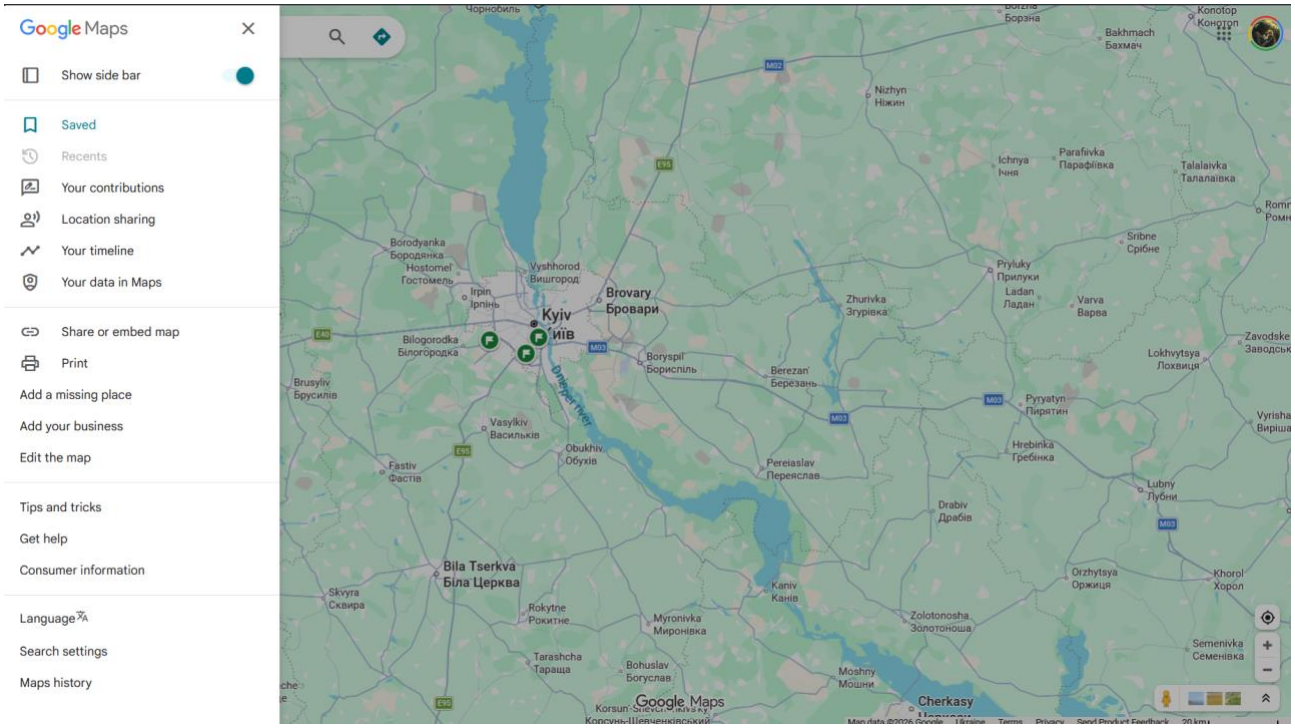


Рис. 4 Інтерфейс платформи Google Maps

Переваги:

- **Глобальне покриття та деталізація:** Найбільша база об'єктів інфраструктури (POI), актуальні супутникові знімки та дуже точна дорожня мережа.
- **Мультимодальна маршрутизація:** Можливість будувати складні шляхи з використанням громадського транспорту, автомобілів та пішохідних зон.
- **Інтеграція з екосистемою:** Тісний зв'язок з іншими сервісами компанії, що дозволяє легко зберігати локації в особистому обліковому записі.

Недоліки:

- **Перевантаженість інтерфейсу:** Карта часто містить забагато візуального шуму (реклама, фотографії закладів, відгуки), що сильно відвертає увагу від безпосереднього аналізу маршруту.
- **Відсутність кліматичного контексту:** Платформа не вміє генерувати деталізований прогноз погоди безпосередньо вздовж прокладеної полілінії шляху.
- **Відсутність інструментів фокусування:** Не передбачено функціоналу для створення ізольованих локальних сесій (симуляцій поїздок) з таймером.

Waze - соціальний навігаційний додаток, орієнтований переважно на водіїв. Головною особливістю платформи є краудсорсинг - дані про дорожню обстановку оновлюються самими користувачами в реальному часі. Це дозволяє миттєво реагувати на появу поліцейських патрулів, аварій чи ремонтних робіт на дорозі. Інтерфейс даного застосунку зображено на рис. 5. [5]

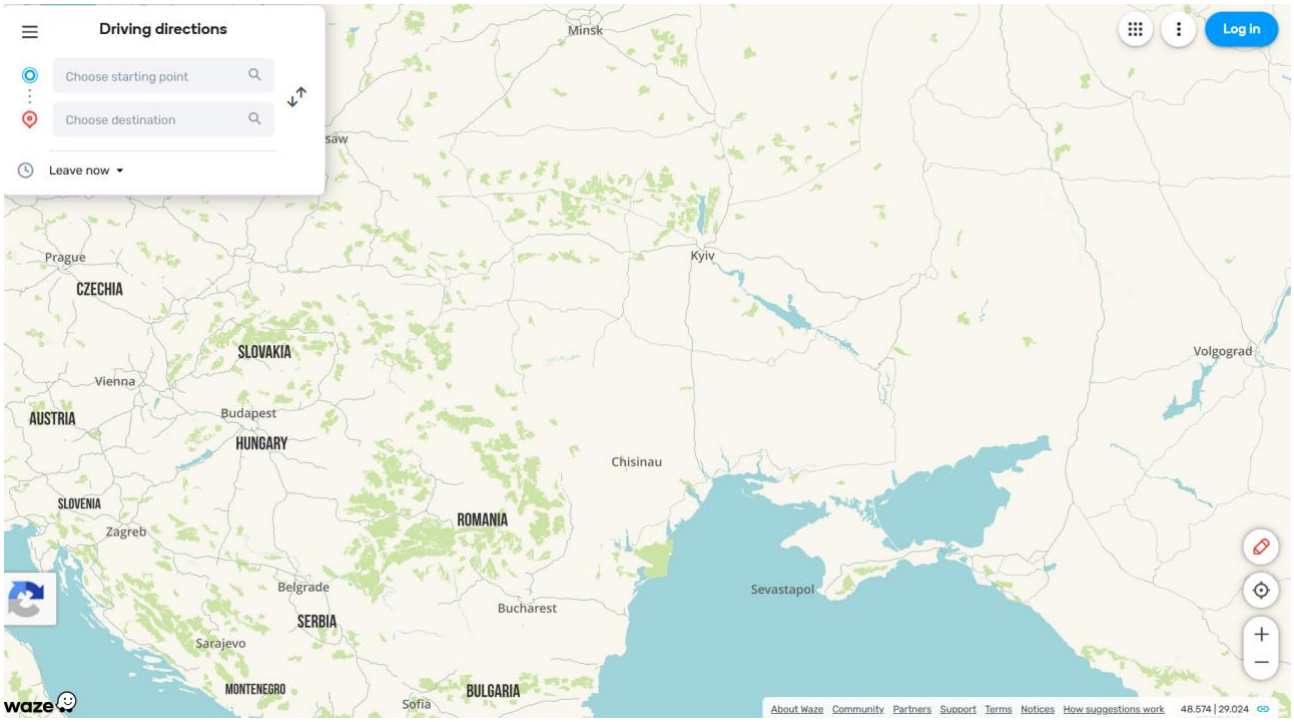


Рис. 5 Інтерфейс навігатора Waze

Переваги:

- **Соціальна взаємодія:** Миттєве сповіщення про дорожні події завдяки активності великої спільноти водіїв.
- **Динамічна перемаршрутизація:** Алгоритми дуже швидко змінюють шлях, якщо на поточній траєкторії виникає непередбачений затор.
- **Гейміфікація:** Наявність ігрових елементів та рейтингів, що стимулює користувачів активно ділитися інформацією.

Недоліки:

- **Надмірне відвернення уваги:** Велика кількість спливаючих повідомлень та соціальних елементів повністю суперечить концепції спокійної "фокус-сесії".
- **Вузька спеціалізація:** Додаток призначений виключно для активного водіння і не дуже підходить для попереднього аналітичного планування подорожей з екрану комп'ютера.

OSRM Web Client (Open Source Routing Machine) - відкритий базовий вебклієнт, що демонструє можливості рушія OSRM. Він використовує дані OpenStreetMap для надзвичайно швидкого розрахунку найкоротших шляхів на дорожньому графі. Інтерфейс даного клієнта зображено на рис. 6. [6]

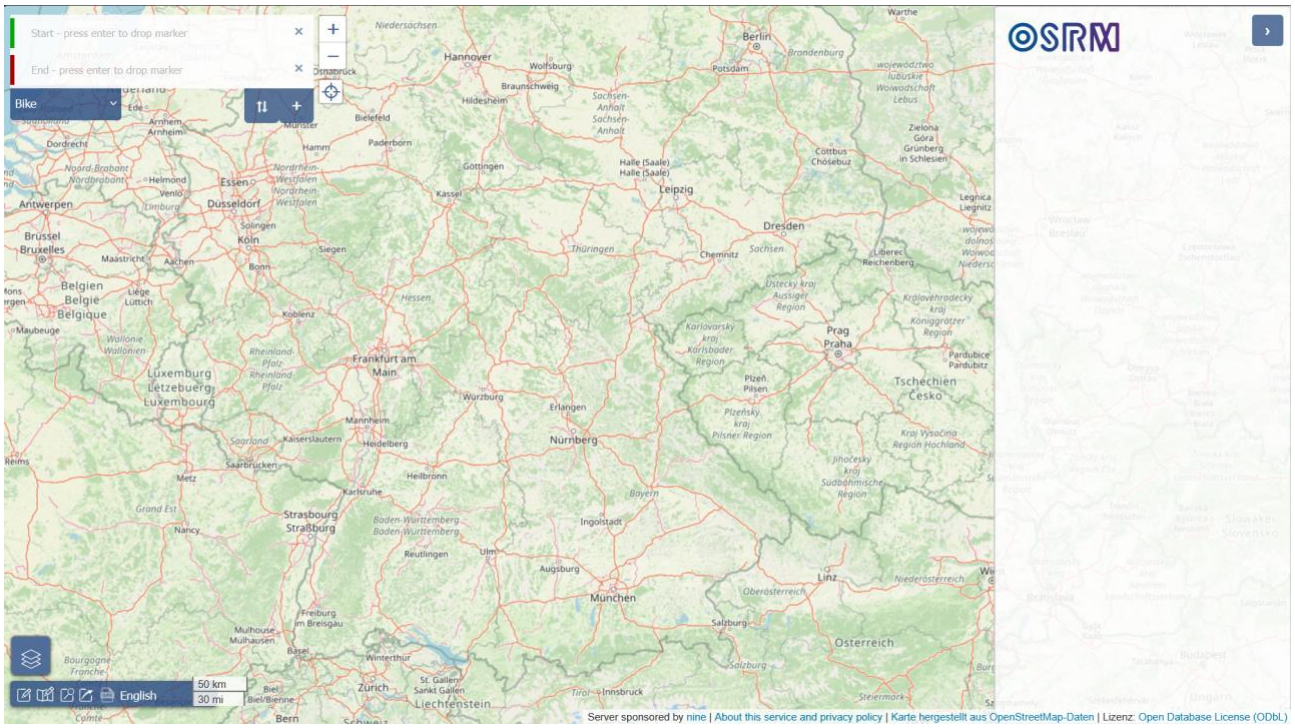


Рис. 6 Інтерфейс базового вебклієнта OSRM

Переваги:

- **Висока швидкодія:** Математичні алгоритми рушія здатні обчислювати складні маршрути за лічені мілісекунди.
- **Відкритий вихідний код:** Можливість вільного використання та інтеграції базових геометрій шляху у власні проєкти.

Недоліки:

- **Відсутність збагачення даних (Enrichment):** Це лише "голий" математичний алгоритм, який видає координати і базовий час. У ньому немає агрегації погодних даних чи статистики трафіку.
- **Спартанський інтерфейс:** Платформа не орієнтована на кінцевого користувача, не має панелей аналітики чи можливості збереження історії поїздок.

Підсумовуючи, можна зробити висновок, що на ринку існує прогалина у ніші агрегаторів. Користувачі змушені використовувати навігатори для побудови шляху, окремі сайти для моніторингу погоди та сторонні інструменти для тайм-менеджменту. Тому розробка системи TransitMind, яка об'єднає всі ці інструменти в єдиному мінімалістичному інтерфейсі, є цілком виправданою.

1.5 Постановка завдання

Головним призначенням розроблюваного проєкту є створення програмного забезпечення картографічної вебплатформи, що забезпечує побудову та візуалізацію маршруту, відображення просторових даних на карті та графічне подання додаткових атрибутів (метеорологічних і транспортно-часових) у спеціальній інтерактивній панелі.

Для досягнення цієї мети необхідно виконати такі конкретні завдання:

1. **Аналіз проблематики:** дослідити існуючі картографічні сервіси, виявити їхні слабкі сторони в контексті агрегації зовнішніх даних та сформувати перелік вимог до розроблюваної системи.
2. **Проєктування бази даних:** розробити оптимізовану реляційну структуру бази даних PostgreSQL, яка міститиме лише необхідні сутності (міста, дороги як ребра графа, збережені маршрути та сесії), свідомо відмовившись від перевантаження системи непотрібними таблицями.
3. **Розробка серверної частини (агрегатора):** створити бекенд на базі Node.js та Express, який виступатиме єдиною точкою входу для клієнтських запитів. Сервер має самостійно звертатися до зовнішніх API (OSRM для маршрутизації, Open-Meteo для погоди, TomTom для трафіку), формувати зведений пакет збагачених даних (enrichment) та безпечно віддавати його клієнту.
4. **Створення інтерактивного клієнтського інтерфейсу:** за допомогою бібліотеки React та картографічного рушія Leaflet реалізувати зручний користувацький інтерфейс. Окремим завданням є розробка розширюваної бічної панелі (drawer), де метрики подаватимуться у вигляді чистих SVG-графіків температури (sparklines) та діаграм порівняння часу.
5. **Реалізація алгоритмічного ядра:** інтегрувати алгоритм Дейкстри для пошуку найкоротших шляхів на абстрактному графі міст, а також розробити математичний механізм інтерполяції для симуляції руху маркера вздовж згенерованої полілінії під час локальної фокус-сесії.

6. **Тестування та документація:** перевірити надійність системи, зокрема її здатність використовувати евристичні методи оцінки завантаженості доріг за відсутності підключення до платних комерційних API, та скласти рекомендації щодо її експлуатації.

Вирішення цих завдань дозволить отримати повноцінний програмний продукт, здатний змінити підхід до індивідуального планування подорожей у вебсередовищі.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

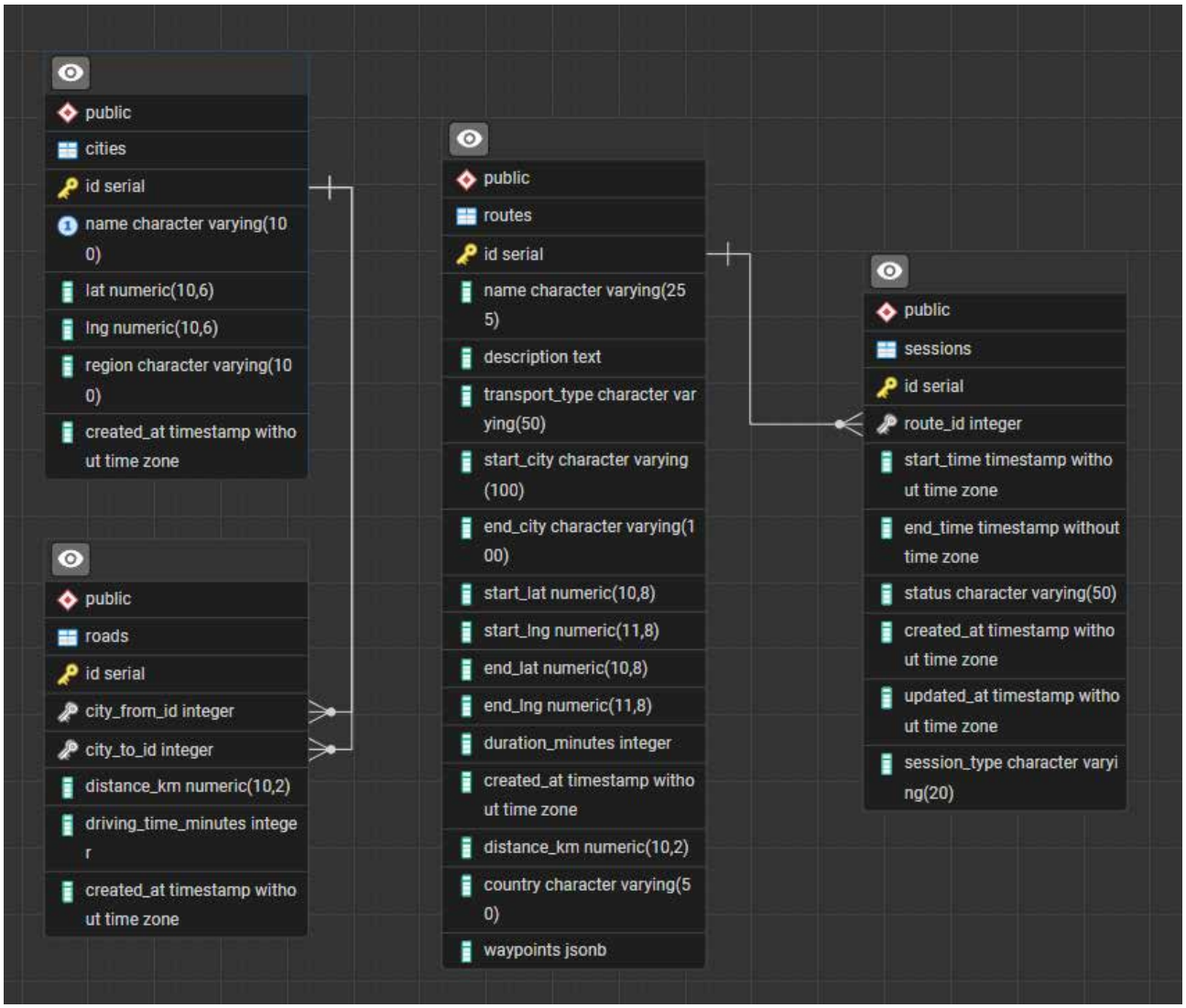
Будь-яка картографічна система працює з великими масивами геопросторових даних, тому перед безпосередньою програмною реалізацією необхідно чітко спроєктувати структуру збереження цієї інформації. Найкращим інструментом для візуалізації структури бази даних є ER-діаграма (Entity-Relationship). Вона являє собою високорівневу концептуальну модель, яка демонструє логічні сутності реального світу та зв'язки між ними. Основними компонентами такої діаграми є сутності (представлені прямокутниками), атрибути (описують властивості сутностей) та зв'язки (відображають взаємодію типу «один-до-одного» або «один-до-багатьох»).

Оскільки архітектура платформи TransitMind орієнтована на індивідуальне планування подорожей без обов'язкової реєстрації чи соціальної взаємодії, логічна модель даних була максимально оптимізована. Під час міграції бази даних із системи було свідомо видалено надлишкові таблиці профілів, друзів та чатів. У базі даних залишено лише ті сутності, які безпосередньо відповідають за маршрутизацію та роботу фокус-сесій.

Основні сутності бази даних TransitMind:

1. **cities (Міста):** ключова сутність, що виступає вершинами абстрактного дорожнього графа. Містить унікальний ідентифікатор (id), географічну назву, точні координати (lat, lng у форматі numeric), регіональну прив'язку та час створення запису (created_at).
2. **roads (Дороги):** сутність, що відіграє роль ребер графа. Вона з'єднує два міста (через зовнішні ключі city_from_id та city_to_id) і зберігає вагові коефіцієнти для алгоритму Дейкстри: фізичну відстань у кілометрах та орієнтовний час у дорозі у хвилинах.
3. **routes (Маршрути):** комплексна сутність для збереження згенерованих шляхів. Зберігає назву, опис, тип транспорту, координати стартової та кінцевої точок, загальну тривалість і відстань, а також детальний масив контрольних точок (waypoints) у форматі JSONB.
4. **sessions (Сесії):** сутність, що фіксує локальні фокус-сесії (віртуальні симуляції подорожі). Зберігає час старту, час завершення, поточний статус виконання (status), тип сесії та має жорстку прив'язку до конкретного маршруту через ключ route_id.

Для платформи TransitMind було сформовано ER-діаграму бази даних у третій нормальній формі (3НФ), яку зображено на рис. 7.



Опис зв'язків між сутностями на розробленій моделі:

- **Відношення між cities та roads:** одне місто може бути початковою або кінцевою точкою для багатьох доріг, що утворює зв'язок «один-до-багатьох» (реалізовано через два окремі ключі зв'язку city_from_id та city_to_id).
- **Відношення між routes та sessions:** кожен збережений маршрут може бути пройдений користувачем у режимі фокус-сесії декілька разів (зв'язок «один-до-багатьох»). Водночас кожна сесія належить лише одному унікальному маршруту.

Така мінімалістична структура усуває дублювання інформації та забезпечує максимальну швидкодію серверної частини під час виконання алгоритмів обходу графа.

2.2 Діаграма класів та кооперацій

Після проєктування реляційної бази даних логічним кроком є розробка архітектури самого програмного забезпечення.

2.2.1 Діаграма класів. Ця діаграма використовується для моделювання статичної структури об'єктно-орієнтованих систем. Вона демонструє класи, їхні атрибути, операції (методи) та зв’язки між ними. Графічно клас зображується у вигляді прямокутника, поділеного на три секції: верхня містить назву, середня - атрибути, а нижня - доступні методи. Хоча TransitMind використовує сучасний компонентний підхід (бібліотека React для клієнтської частини та фреймворк Express для серверної), класична діаграма класів чудово підходить для опису розподілу відповідальності в системі. Сформовану діаграму зображено на рис. 8.

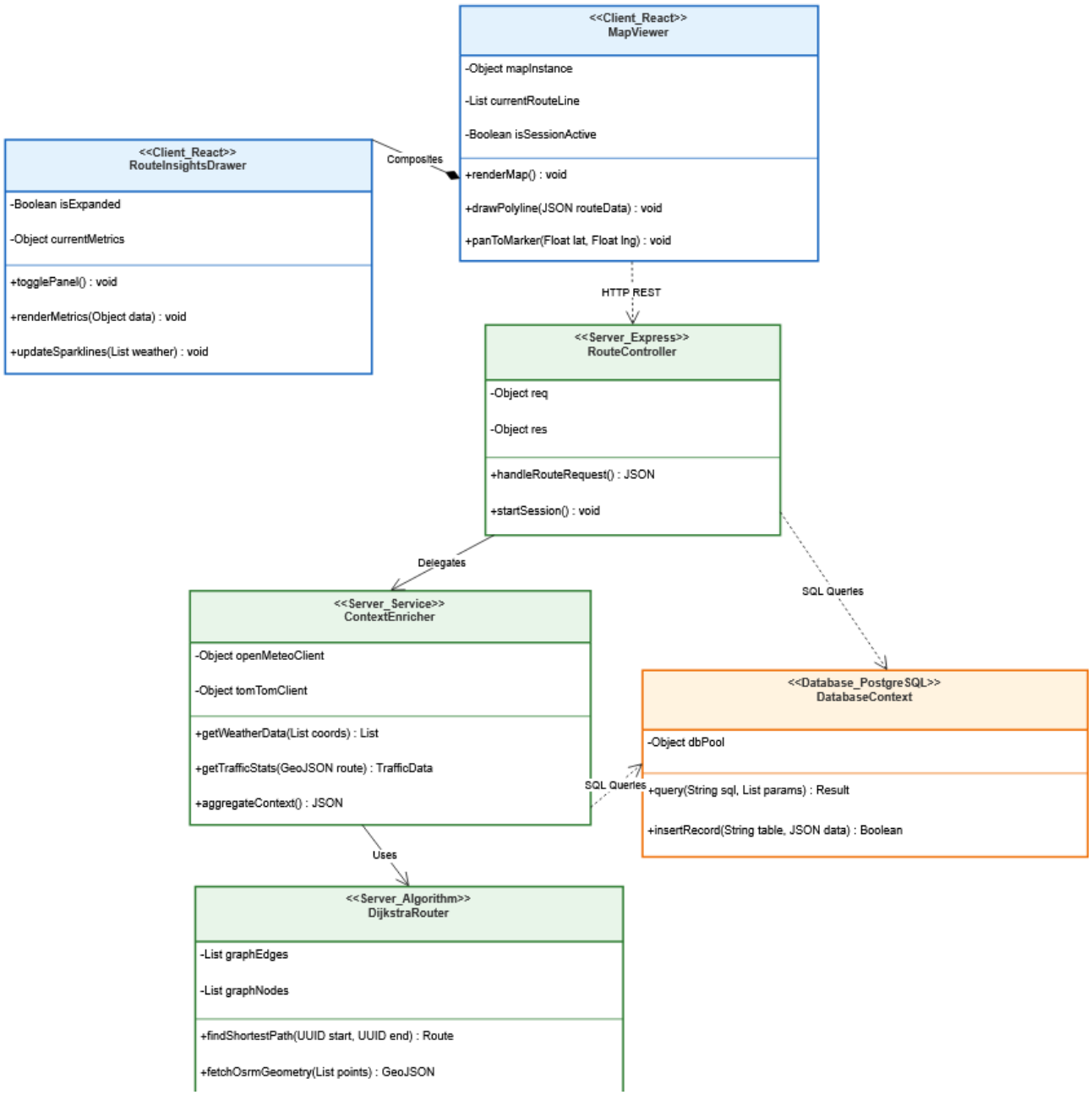


Рис. 8 Діаграма класів системи TransitMind

Архітектурно логічні класи поділені на три рівні взаємодії:

- **Клієнтський рівень (React):** класи MapViewer та RouteInsightsDrawer. Вони відповідають за ініціалізацію картографічного рушія Leaflet, відмальовку поліліній та візуалізацію панелі аналітики (пов'язані між собою відношенням композиції).
- **Серверний рівень (Node.js):** контролер RouteController, який приймає клієнтські HTTP-запити і делегує роботу сервісам агрегації. Клас ContextEnricher звертається до математичного ядра DijkstraRouter для

прорахунку графа та до зовнішніх API для збагачення маршруту погодними і транспортними даними.

- **Рівень бази даних:** абстракція DatabaseContext, яка інкапсулює логіку підключення до пулу PostgreSQL та виконання SQL-запитів (insertRecord, query).

2.2.2 Діаграми кооперацій. Якщо діаграма класів показує статику, то діаграма кооперацій розкриває динаміку - як саме об'єкти системи співпрацюють для досягнення результату. Для платформи TransitMind розроблено дві діаграми кооперацій, які описують найважливіші процеси: агрегацію зовнішніх даних та управління фокус-сесією.

Діаграму кооперації для базового процесу «Збагачення маршруту» зображено на рис. 9.

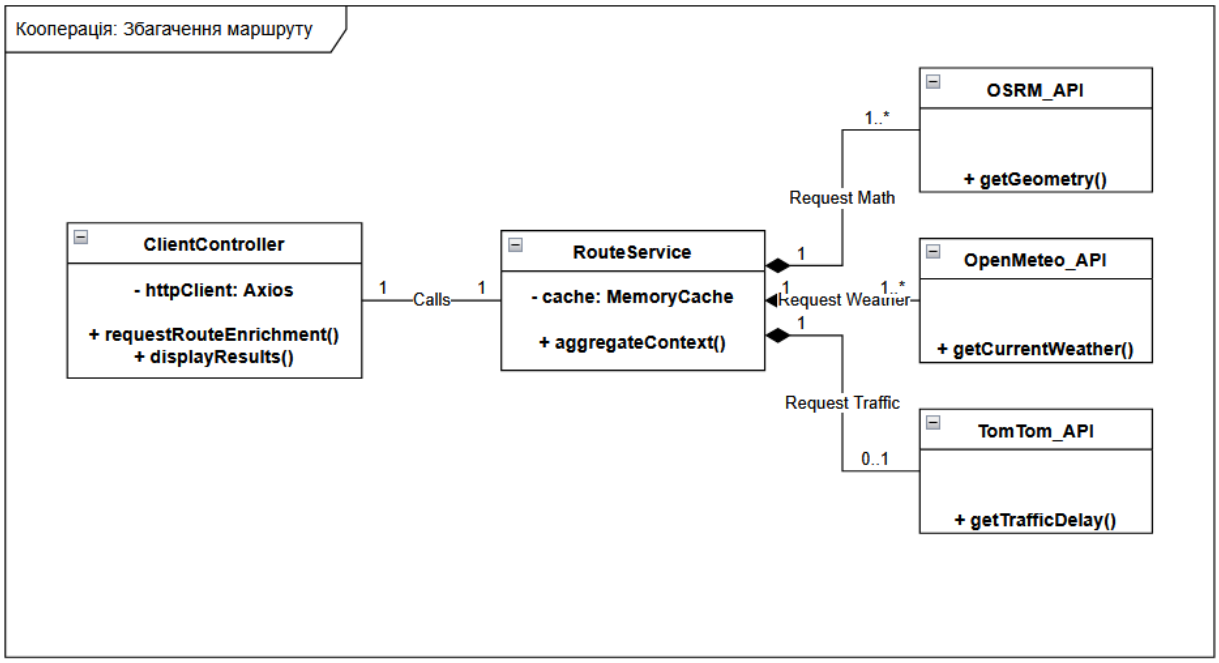


Рис. 9 Діаграма кооперацій: Збагачення маршруту

Під час виконання цього сценарію взаємодія відбувається за наступним ланцюгом: об'єкт ClientController відправляє HTTP-запит до серверного RouteService. Цей сервіс паралельно звертається до зовнішніх інстансів: OSRM_API (для математичного розрахунку геометрії), OpenMeteo_API (для кліматичного контексту) та TomTom_API (для даних про трафік). Після

отримання відповідей викликається метод aggregateContext(), який структурує інформацію у єдиний JSON-пакет і повертає її клієнту. Це гарантує чітке розділення відповідальності (Separation of Concerns).

Другою важливою взаємодією є процес «Запуску фокус-сесії», який фіксує старт віртуальної подорожі в системі. Діаграму цієї кооперації зображено на рис. 10.

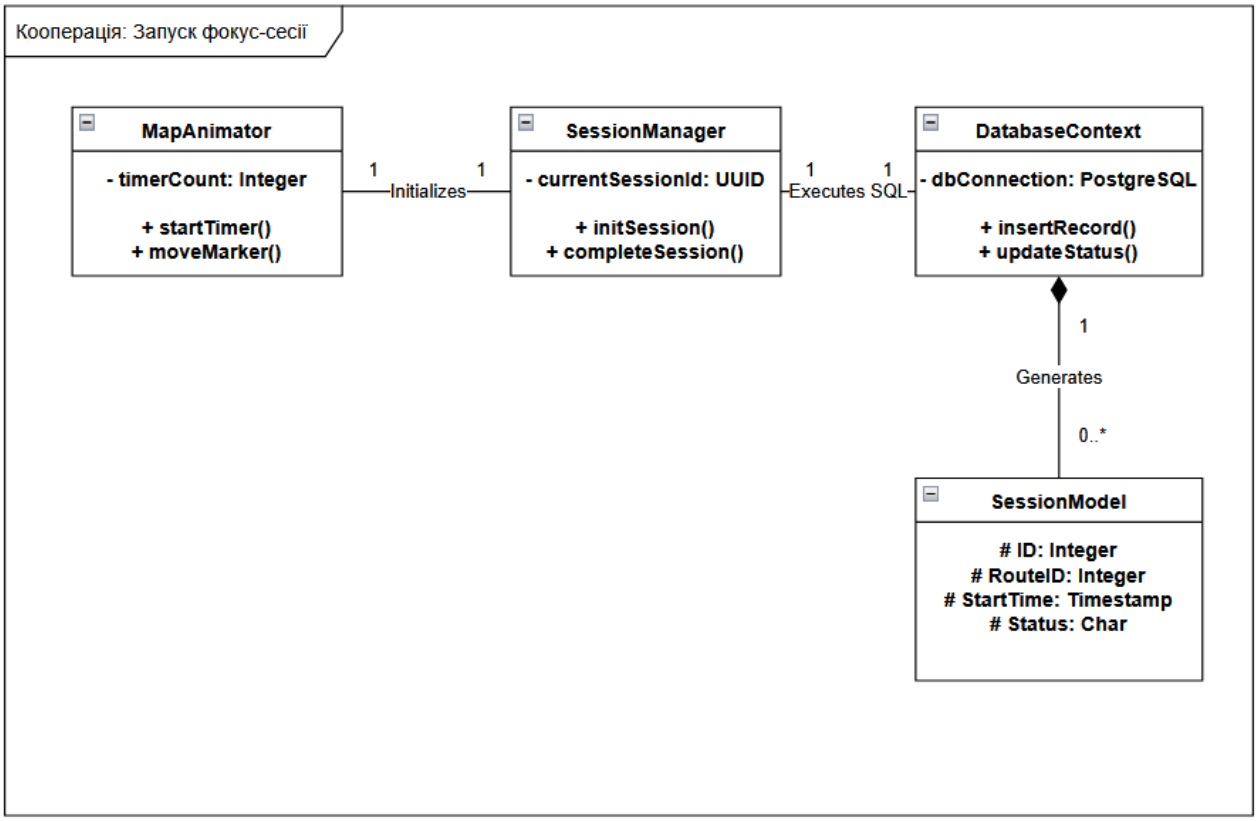


Рис. 10 Діаграма кооперацій: Запуск фокус-сесії

У цьому сценарії клієнтський об'єкт MapAnimator ініціює запит на початок подорожі до серверного об'єкта SessionManager. Сервер звертається до пулу бази даних (DatabaseContext) для створення нового запису. Після успішної генерації моделі SessionModel та повернення ідентифікатора сесії, клієнт активує локальні методи startTimer() та moveMarker(), ініціюючи анімацію руху по полілінії.

2.3 Діаграма пакетів

Діаграма пакетів (package diagram) є одним із ключових інструментів для структурування великих програмних систем. Її головна мета полягає у розбитті

архітектури застосунку на менші, логічно зв'язані підсистеми (пакети) з метою полегшення розуміння коду та мінімізації залежностей між окремими модулями. Для картографічної платформи TransitMind було сформовано діаграму пакетів, яка наочно демонструє розподіл логіки між клієнтським застосунком (на базі збирача Vite) та серверним середовищем. Діаграму зображено на рис. 11.

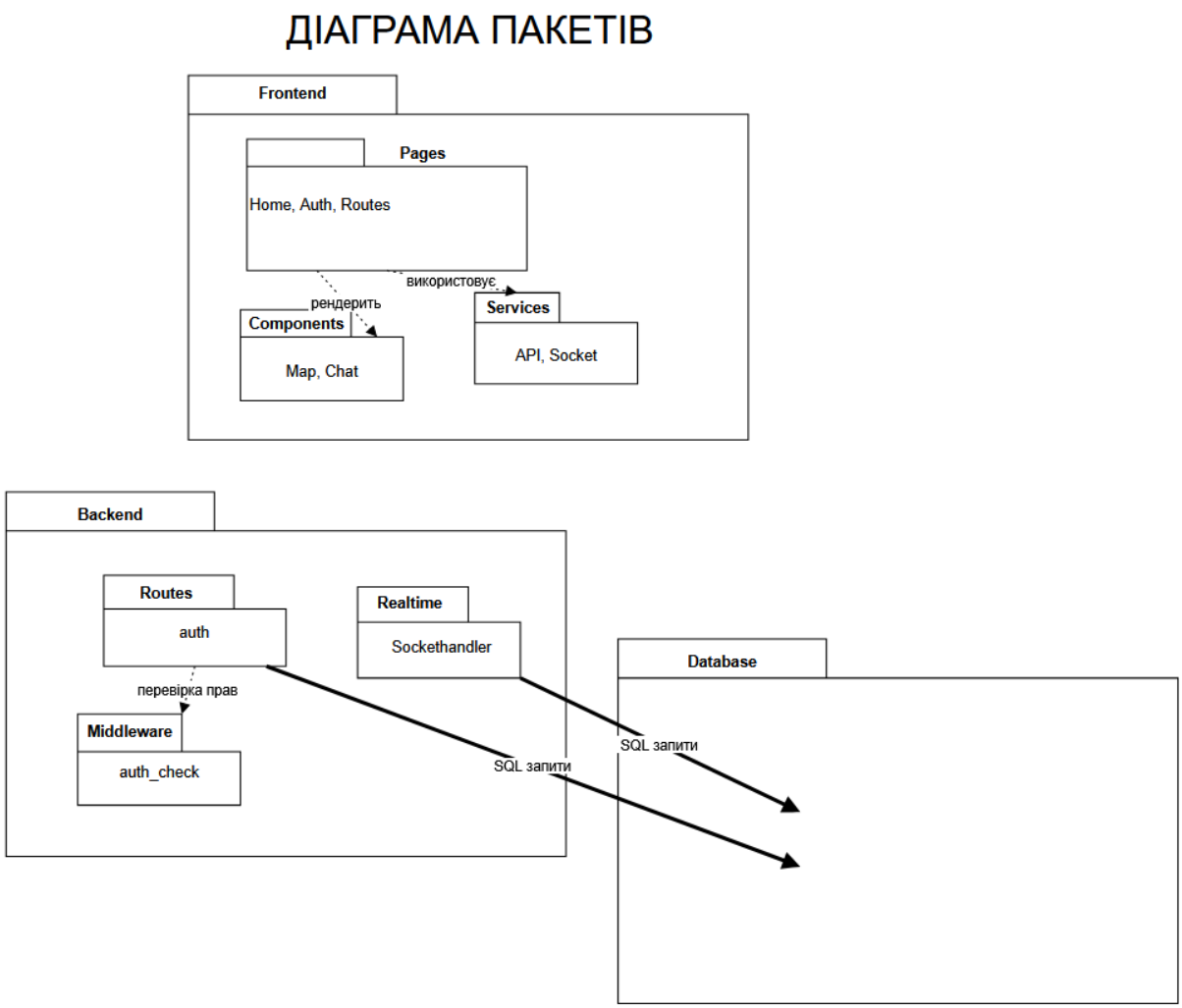


Рис. 11 Діаграма пакетів системи TransitMind

Розроблена система поділяється на два глобальні простори (субсистеми):

1. **Клієнтська частина (React Application).** Включає пакет компонентів користувацького інтерфейсу (UI_Components), пакет сервісних функцій для маршрутизації HTTP-запитів (API_Services) та пакет управління станом застосунку (State_Context).

2. **Серверна частина (Node.js Application).** Містить пакет контролерів (REST_Controllers), пакет зовнішніх інтеграцій (External_Integrations) для зв'язку з картографічними сервісами та пакет доступу до бази даних (Data_Access_Layer).

Наявність чітких векторних залежностей (односторонніх зв'язків типу <<use>>) від клієнта до контролерів, а від контролерів до зовнішніх сервісів гарантує, що браузер користувача ніколи не має прямого доступу до приватних ключів API або таблиць бази даних.

2.4 Діаграма компонентів

Діаграма компонентів відображає концептуальну картину взаємодії між фізичними та логічними частинами системи. Вона дозволяє візуалізувати архітектуру програмного забезпечення на найвищому рівні абстракції, показуючи, які саме зовнішні системи, сервери та бази даних залучені до роботи продукту, і які протоколи зв'язку між ними використовуються.

Враховуючи, що платформа TransitMind активно інтегрується із зовнішніми джерелами даних (технологія route enrichment), діаграма компонентів є критично важливою для розуміння руху інформації. Діаграму компонентів зображено на рис. 12.

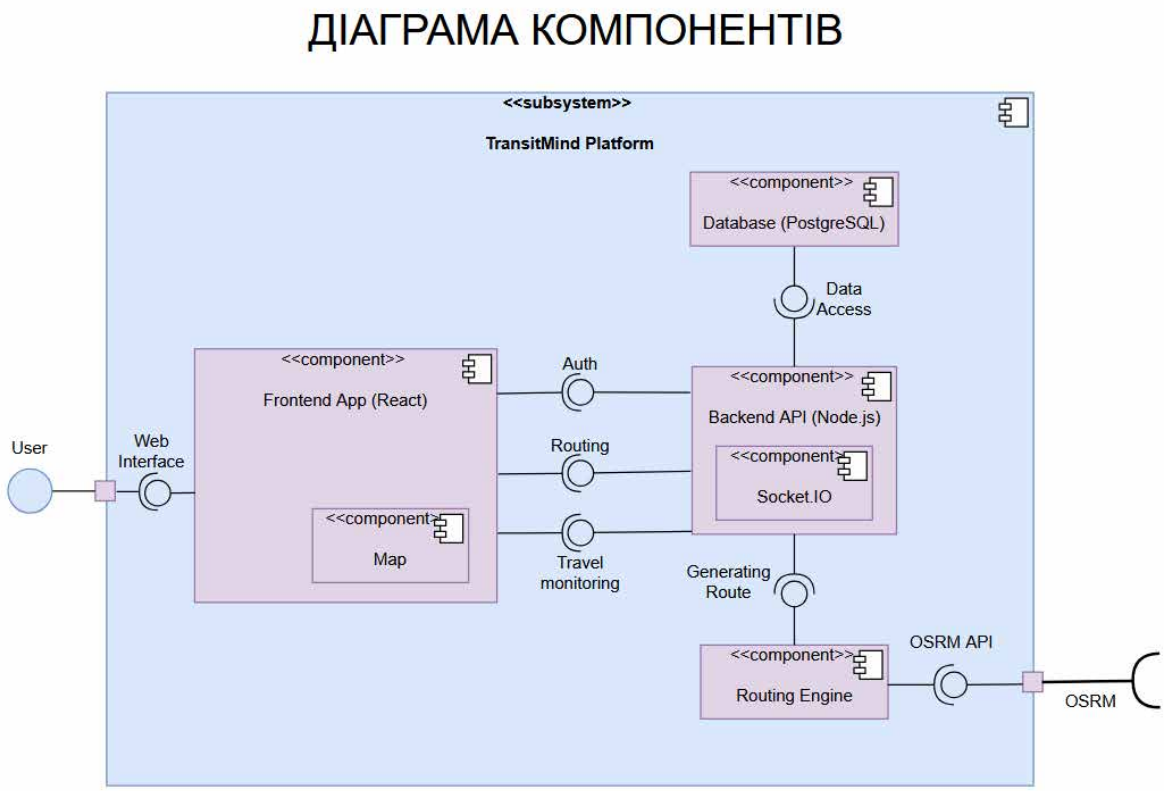


Рис. 12 Діаграма компонентів системи TransitMind

Система складається з наступних взаємодіючих компонентів:

- **SPA Client:** односторінковий вебзастосунок на базі React, який завантажується у браузері користувача. Взаємодіє з сервером виключно за протоколом HTTP REST (передаючи та отримуючи дані у форматі JSON).
- **REST API Gateway:** ядро бізнес-логіки на базі Node.js та фреймворку Express. Виступає єдиною точкою входу для клієнтських запитів та проксі-сервером для зовнішніх викликів.
- **PostgreSQL Database:** реляційна база даних, яка спілкується з сервером через TCP-з'єднання та SQL-запити.
- **External Services:** набір сторонніх компонентів, до яких бекенд робить самостійні Fetch-запити (OSRM для геометрії шляху, Open-Meteo для кліматичного контексту та TomTom Traffic для аналізу заторів).

Така компонентна архітектура робить проєкт високомасштабованим: наприклад, можна легко замінити публічний інстанс OSRM на власний ізольований сервер маршрутизації без жодних змін у клієнтському коді.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Для розробки програмного забезпечення автоматизації побудови маршрутів та фокус-сесій TransitMind як основну систему управління базами даних (СУБД) було обрано **PostgreSQL**. Це потужна об'єктно-реляційна система з відкритим вихідним кодом, яка відрізняється високою надійністю та ідеально підходить для роботи з геопросторовими даними.

Вибір саме цієї СУБД обґрунтовано наступними технічними перевагами:

- **Підтримка формату JSONB:** дозволяє зручно та ефективно зберігати масиви координат та контрольних точок маршруту (waypoints).
- **Висока точність обчислень:** використання типу numeric гарантує відсутність похибок при збереженні географічної широти та довготи.
- **Швидкодія та інтеграція:** наявність оптимізованого драйвера pg для середовища Node.js забезпечує швидку асинхронну взаємодію бекенду з інформаційною базою.

Усі транзакції від сервера до бази даних виконуються за допомогою діалекту SQL, що дозволяє миттєво витягувати ребра графа для обчислення маршрутів.

3.2 Розробка інформаційної бази

На етапі проєктування платформи TransitMind архітектура бази даних пройшла процес оптимізації (міграція 004_core_schema_trim.sql). З ядра було видалено надлишкові соціальні таблиці, щоб максимально розвантажити базу даних.

Для створення бази даних використовується ключове слово CREATE DATABASE, яке дозволяє створити нову інформаційну базу із заданою назвою. Запит створення бази даних зображено на рис. 13.

```
CREATE DATABASE transitmind
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    CONNECTION LIMIT = -1;
```

Рис. 13 Запит створення бази даних TransitMind

У фінальній інформаційній базі залишилися чотири ключові сутності (таблиці): cities (вершини графа), roads (ребра графа), routes (збережені шляхи) та sessions (фокус-сесії). Для створення таблиць використовується стандартний синтаксис CREATE TABLE. Запит створення таблиці зображено на рис. 14. Для прикладу візьмемо комплексну таблицю routes, яка використовує тип JSONB для збереження полілінії шляху.

```
CREATE TABLE routes (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    transport_type VARCHAR(50) DEFAULT 'driving',
    start_city VARCHAR(100),
    end_city VARCHAR(100),
    start_lat NUMERIC(10, 8) NOT NULL,
    start_lng NUMERIC(11, 8) NOT NULL,
    end_lat NUMERIC(10, 8) NOT NULL,
    end_lng NUMERIC(11, 8) NOT NULL,
    duration_minutes INTEGER,
    distance_km NUMERIC(10, 2),
    waypoints JSONB,
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

Рис. 14 Запит створення таблиці «routes»

Тригери – це спеціальний тип функцій у PostgreSQL, які запускаються автоматично при виконанні певних дій з таблицями бази даних. Вони забезпечують автоматизацію рутинних процесів. Наприклад, для таблиці sessions необхідно автоматично оновлювати поле updated_at кожного разу, коли змінюється статус сесії. Запит створення відповідної функції та тригера зображено на рис. 15.

```sql
-- Створення функції для оновлення часу
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Створення тригера для таблиці sessions
CREATE TRIGGER set_updated_at
BEFORE UPDATE ON sessions
FOR EACH ROW
EXECUTE FUNCTION update_updated_at_column();
```

Рис. 15 Запит створення функції та тригера для таблиці «sessions»

Щоб забезпечити цілісність інформаційної бази, між усіма таблицями встановлено жорсткі зв'язки через зовнішні ключі (FOREIGN KEY). Сформований повний лістинг SQL-запитів створення всіх таблиць, зв'язків та тригерів представлений у **Додатку А**.

Після створення всіх компонентів бази даних, за допомогою засобів СУБД була сформована діаграма (схема) бази даних, яка зображена на рис. 16.

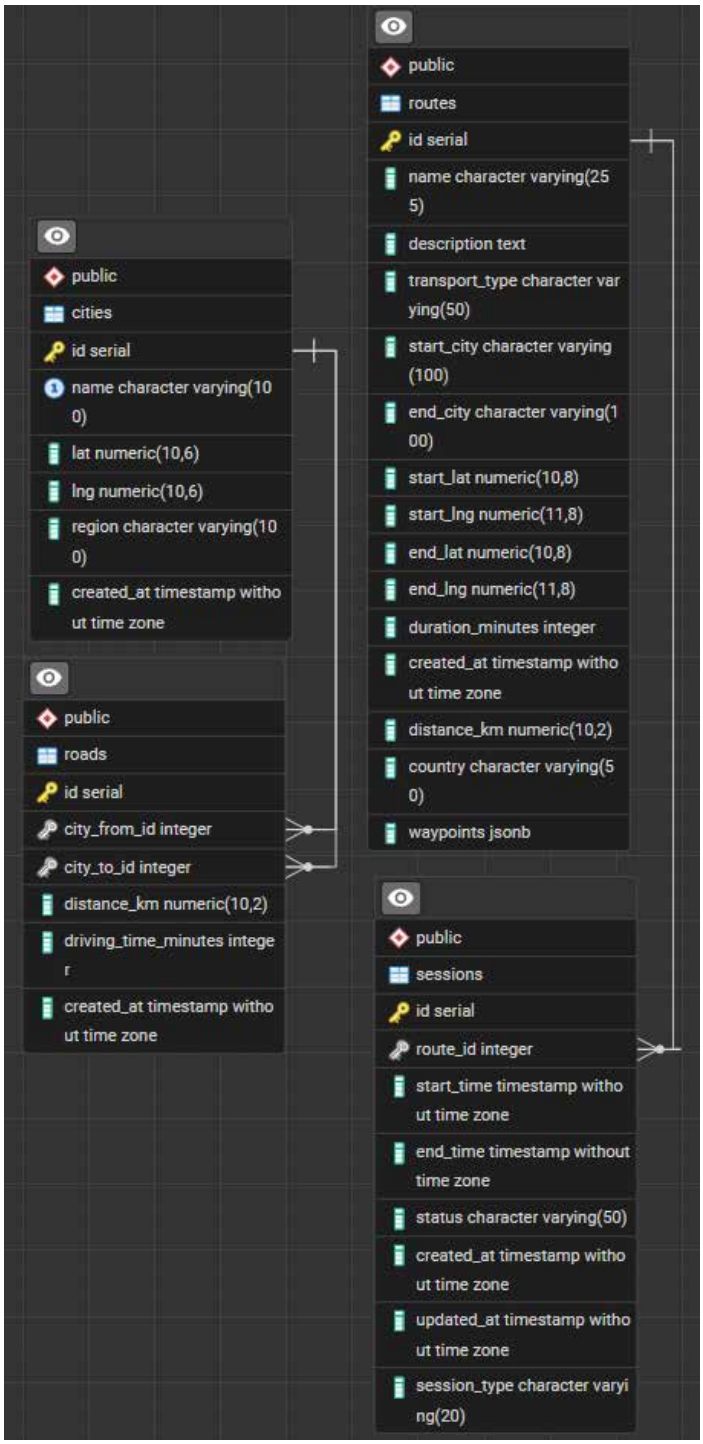


Рис. 16 Діаграма створеної бази даних

3.3 Вибір інструментарію для створення прикладного ПЗ

Для створення повноцінної архітектури картографічної платформи TransitMind було застосовано сучасний стек вебтехнологій, розділений на клієнтську та серверну частини.

Клієнтська частина (Frontend):

- **React 19:** Основна бібліотека для побудови користувацького інтерфейсу на основі компонентного підходу. Забезпечує реактивне оновлення метрик на бічній панелі.
- **Vite:** Сучасний та надзвичайно швидкий збирач модулів (bundler), який використовується для локальної розробки та збірки продакшен-версії фронтенду.
- **Leaflet та react-leaflet:** Потужний картографічний рушій з відкритим вихідним кодом. Використовується для рендерингу базового шару карти (наприклад, Esri World Dark Gray), малювання поліліній маршруту та анімації маркерів під час сесії.
- **Axios:** HTTP-клієнт для здійснення асинхронних REST-запитів до нашого бекенду.

Серверна частина (Backend):

- **Node.js та Express 5:** Надійне середовище виконання та мінімалістичний вебфреймворк, які виконують роль API-шлюзу (Gateway). Сервер приймає запити від браузера та керує бізнес-логікою.
- **luxon:** Бібліотека для складної роботи з датами та часовими поясами. Вона є критично важливою для розрахунку локального часу прибуття (наприклад, генерації профілю заторів на "завтра" у конкретному часовому поясі маршруту).

Інтеграція зовнішніх сервісів (Data Enrichment): Ключовою особливістю TransitMind є збагачення базової геометрії шляху додатковим контекстом. Для цього серверна частина звертається до наступних зовнішніх API:

1. **OSRM (Open Source Routing Machine):** Використовується для отримання точної дорожньої геометрії (GeoJSON), дистанції та базового часу в дорозі між точками маршруту.
2. **Open-Meteo:** Надає деталізований прогноз погоди (температура, опади, швидкість вітру) вздовж заданих координат полілінії.

3. **TomTom Routing API:** Опціональний сервіс, який надає професійні моделі трафіку (затримки в реальному часі, історичний трафік та вільний рух) для побудови порівняльних графіків у панелі аналітики.

Всі запити до платних або обмежених API виконуються виключно на стороні бекенда (проксіювання), що гарантує захист приватних ключів доступу (наприклад, TOMTOM_API_KEY, який зберігається у файлі .env на сервері).

3.4 Алгоритмізація та програмування програмних модулів

Алгоритмічне забезпечення, що застосовується до конкретного об’єкта, дозволяє визначити необхідну структуру і склад обчислювально-керуючого комплексу, виробити вимоги до швидкодії та надійності проєктованої системи. Алгоритмічне забезпечення TransitMind полягає в моделюванні поведінки користувача та розробці складних математичних алгоритмів для обробки геопросторових даних. Значним елементом є оптимізація алгоритмів для взаємодії з базою даних та паралельне виконання запитів до зовнішніх сервісів. Результатом створення правильних алгоритмів є вдала та безпомилкова робота системи побудови маршрутів.

У системі можна виділити чотири ключові алгоритмічні модулі, які забезпечують її функціонування.

3.4.1 Алгоритм Дейкстри для пошуку найкоротшого шляху Першим і найголовнішим модулем серверної частини є класичний алгоритм Дейкстри, імплементований для пошуку найкоротшого шляху на абстрактному графі бази даних. Вершинами графа виступають об'єкти з таблиці cities, а ребрами — roads з ваговими коефіцієнтами (відстань у кілометрах). Алгоритм використовує структуру даних «черга з пріоритетом» (Priority Queue). На кожному кроці він вибирає місто з мінімальною накопиченою відстанню та оновлює відстані до сусідів, доки не досягне цільової кінцевої точки.

Частину коду, яка відповідає за програмну реалізацію алгоритму Дейкстри на мові JavaScript (у середовищі Node.js), зображено на рис. 17.

```
function findPath(cities, roads, startId, endId, type = 'shortest') {
  if (startId === endId) {
    return {
      path: [startId],
      totalDistance: 0,
      totalTime: 0,
      waypoints: [cities.find(c => c.id === startId).name],
    };
  }


  const graph = {};
  cities.forEach(city => {
    graph[city.id] = [];
  });

  roads.forEach(road => {
    graph[road.city_from_id].push({
      to: road.city_to_id,
      distance: parseFloat(road.distance_km),
      time: parseInt(road.driving_time_minutes),
    });
  });

  if (type === 'shortest') {

    return dijkstra(cities, graph, startId, endId);
  } else {

    return longestPathDFS(cities, graph, roads, startId, endId);
  }
}
```

Рис. 17 Реалізація алгоритму Дейкстри для графа міст

3.4.2 Інтеграція алгоритму з дорожнім рушієм OSRM Після того, як алгоритм Дейкстри знаходить абстрактну послідовність міст (наприклад, Київ - Житомир - Біла Церква), система ініціює наступний крок — запит до картографічного рушія маршрутизації OSRM. Складність цього алгоритму полягає у необхідності «склеїти» декілька сегментів маршруту в єдину безперервну полілінію. Програмний модуль перебирає знайдені міста, витягує їхні координати (широту та довготу) і формує URL-запит до публічного інстансу OSRM, отримуючи у відповідь точну геометрію дорожньої мережі у форматі GeoJSON, а також реальну відстань у метрах та час у секундах.

Частину коду, яка відповідає за звернення до API OSRM та агрегацію сегментів шляху, зображено на рис. 18.

```
const getDetailedRoadLine = async (startLat, startLng, endLat, endLng) => {
  const cacheKey = getCacheKey(
    parseFloat(startLat),
    parseFloat(startLng),
    parseFloat(endLat),
    parseFloat(endLng)
  );

  if (routeCache.has(cacheKey)) {
    return routeCache.get(cacheKey);
  }

  const osrmUrl = `https://router.project-osrm.org/route/v1/driving/${startLng},${startLat};${endLng},${endLat}?overview=full&geometries=geojson`;

  try {
    const response = await fetch(osrmUrl);
    if (!response.ok) {
      const fallback = getDirectFallback(startLat, startLng, endLat, endLng);
      routeCache.set(cacheKey, fallback);
      setTimeout(() => routeCache.delete(cacheKey), 60 * 60 * 1000);
      return fallback;
    }

    const data = await response.json();
    if (!data.routes || data.routes.length === 0) {
      const fallback = getDirectFallback(startLat, startLng, endLat, endLng);
      routeCache.set(cacheKey, fallback);
      setTimeout(() => routeCache.delete(cacheKey), 60 * 60 * 1000);
      return fallback;
    }

    const route = data.routes[0];
    const result = {
      coordinates: route.geometry.coordinates.map(([lng, lat]) => ({
        lat: parseFloat(lat),
        lng: parseFloat(lng),
      })),
      distance: route.distance,
      duration: route.duration,
    };

    setCacheWithTTL(cacheKey, result);
    return result;
  } catch (error) {
    return getDirectFallback(startLat, startLng, endLat, endLng);
  }
};

const buildOsrmMultiStopRoute = async (points) => {
  let pathCoordinates = [];
  let totalDistanceMeters = 0;
  let totalDurationSeconds = 0;

  for (let i = 0; i < points.length - 1; i++) {
    const from = points[i];
    const to = points[i + 1];
    const segment = await getDetailedRoadLine(from.lat, from.lng, to.lat, to.lng);

    const coords = segment.coordinates || [];
    if (coords.length > 0) {
      if (pathCoordinates.length === 0) {
        pathCoordinates = coords;
      } else {
        pathCoordinates.push(...coords.slice(1));
      }
    }

    totalDistanceMeters += Number(segment.distance || 0);
    totalDurationSeconds += Number(segment.duration || 0);
  }

  return {
    pathCoordinates,
    totalDistanceKm: Math.round((totalDistanceMeters / 1000) * 100) / 100,
    totalTimeMinutes: Math.max(1, Math.round(totalDurationSeconds / 60)),
  };
```

Рис. 18 Алгоритм побудови геометрії шляху через OSRM

3.4.3 Алгоритм збагачення маршруту контекстними даними (Enrichment) Ключовою інновацією платформи TransitMind є агрегація контексту середовища вздовж побудованого шляху. Для оптимізації швидкодії сервер не робить запит по кожній точці, а застосовує метод рівномірного семплінгу - вибирає до 5 контрольних точок із загальної полілінії. Після цього модуль використовує конструкцію Promise.all() для паралельного асинхронного виконання запитів. Сервер одночасно звертається до сервісу Open-Meteo (для отримання погоди) та timeapi.io (для розрахунку часових поясів). Також реалізовано fallback-евристику: якщо комерційний ключ TomTom API для отримання реальних заторів відсутній, алгоритм автоматично розраховує середню швидкість на основі даних OSRM і визначає рівень завантаженості доріг.

Частину коду, яка відповідає за агрегацію та збагачення даних, зображено на рис. 19.

```
const [weatherResponse, timezoneResponse] = await Promise.all([
  fetch(weatherUrl),
  fetch(timezoneUrl),
]);

router.post('/route-enrichment', async (req, res) => {
  try {
    const { pathCoordinates = [], statistics = {} } = req.body;
    const sampled = samplePathCoordinates(pathCoordinates, 5);

    const weatherAlongRoute =
      sampled.length > 0
        ? await Promise.all(
            sampled.map((p) =>
              fetchPointContext({
                lat: p.lat,
                lng: p.lng,
                label: p.label,
                atTime: new Date().toISOString(),
              })
            )
          )
```

Рис. 19 Алгоритм паралельної агрегації контекстних даних

3.4.4 Алгоритм лінійної інтерполяції для фокус-сесії Коли користувач ініціює запуск фокус-сесії на клієнтській частині (фронтенді на базі React), маркер автомобіля має плавно рухатися картою. Оскільки система не отримує реальних GPS-координат, було розроблено алгоритм лінійної інтерполяції. Знаючи

загальну довжину маршруту, координати зламів полілінії та відсоток пройденого часу сесії (відношення elapsedTime до totalDuration), математична функція вираховує точне проміжне геодезичне положення об'єкта (lat, lng). Це дозволяє карті Leaflet плавно оновлювати позицію через метод panTo().

Частину коду клієнтського застосунку, яка виконує інтерполяцію координат, зображено на рис. 20.

```
export const interpolateAlongPath = (path, progress) => {
  if (!Array.isArray(path) || path.length === 0) {
    return null;
  }

  if (path.length === 1) {
    return { lat: path[0].lat, lng: path[0].lng };
  }

  const clampedProgress = Math.max(0, Math.min(progress, 1));
  if (clampedProgress === 0) {
    return { lat: path[0].lat, lng: path[0].lng };
  }
  if (clampedProgress === 1) {
    const last = path[path.length - 1];
    return { lat: last.lat, lng: last.lng };
  }

  const segmentLengths = [];
  let totalLength = 0;

  for (let i = 0; i < path.length - 1; i++) {
    const from = path[i];
    const to = path[i + 1];
    const len = haversineDistanceKm(from, to);
    segmentLengths.push(len);
    totalLength += len;
  }

  if (totalLength <= 0) {
    return { lat: path[0].lat, lng: path[0].lng };
  }

  const targetDistance = totalLength * clampedProgress;
  let traversed = 0;

  for (let i = 0; i < segmentLengths.length; i++) {
    const segLen = segmentLengths[i];
    if (traversed + segLen >= targetDistance) {
      const localProgress = segLen === 0 ? 0 : (targetDistance - traversed) / segLen;
      const from = path[i];
      const to = path[i + 1];
      return {
        lat: from.lat + (to.lat - from.lat) * localProgress,
        lng: from.lng + (to.lng - from.lng) * localProgress,
      };
    }
    traversed += segLen;
  }

  const last = path[path.length - 1];
  return { lat: last.lat, lng: last.lng };
};
```

Рис. 20 Реалізація лінійної інтерполяції для анімації маркера

Для кращого розуміння послідовності виконання описаних алгоритмічних кроків (від отримання запиту до віддачі збагаченого JSON-пакета на клієнт), загальну логіку бекенду візуалізовано у вигляді блок-схеми, яка зображена на рис. 21.

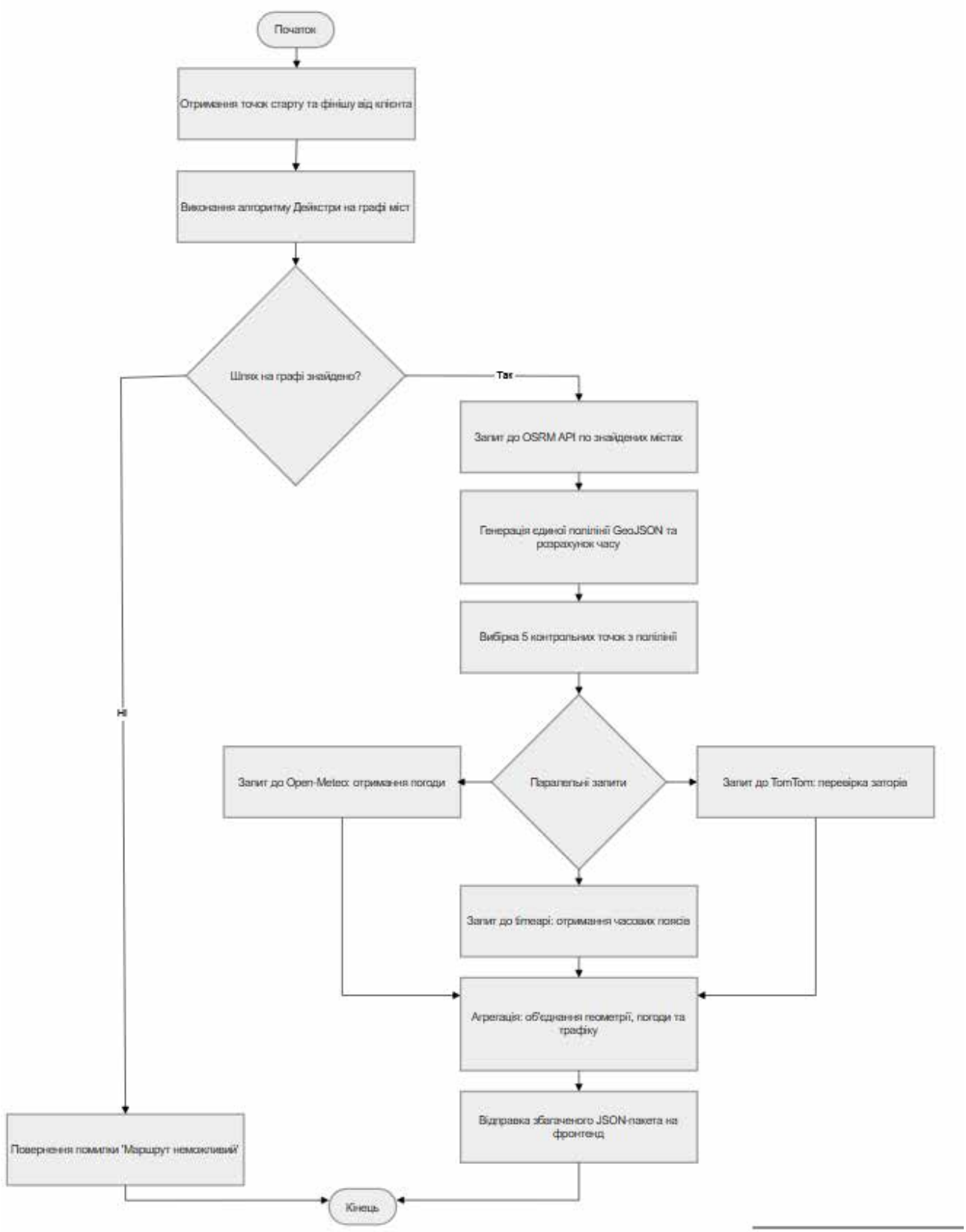


Рис. 21 Блок-схема алгоритму пошуку та збагачення маршруту

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування програмного забезпечення є процесом технічного дослідження, призначеного для отримання інформації про якість продукту у контексті його передбачуваного використання. Ця техніка включає як процес виявлення помилок або інших дефектів, так і оцінку програмних компонентів. Під час тестування можуть оцінюватися такі аспекти:

- відповідність вимогам, на які спиралися проектувальники та розробники;
- коректність відповідей на всі можливі вхідні дані;
- виконання функцій у прийнятний час;
- зручність використання та стійкість до збоїв;
- сумісність з іншими системами.

Для тестування розроблюваної картографічної платформи TransitMind було вирішено скористатися методом «чорної скриньки». Даний метод здійснюється через інтерфейс користувача програмного продукту і не вимагає прямого доступу до внутрішньої структури чи коду програми. Він полягає у взаємодії з програмним продуктом на рівні звичайного користувача. Завдяки цьому підходу можна швидко виявити логічні помилки в інтерфейсі (React) та перевірити коректність отримання даних від бекенду.

На початковому етапі було протестовано завантаження головної сторінки застосунку та ініціалізацію картографічного рушія Leaflet. Інтерфейс головного меню та інтерактивної карти зображено на рис. 22.

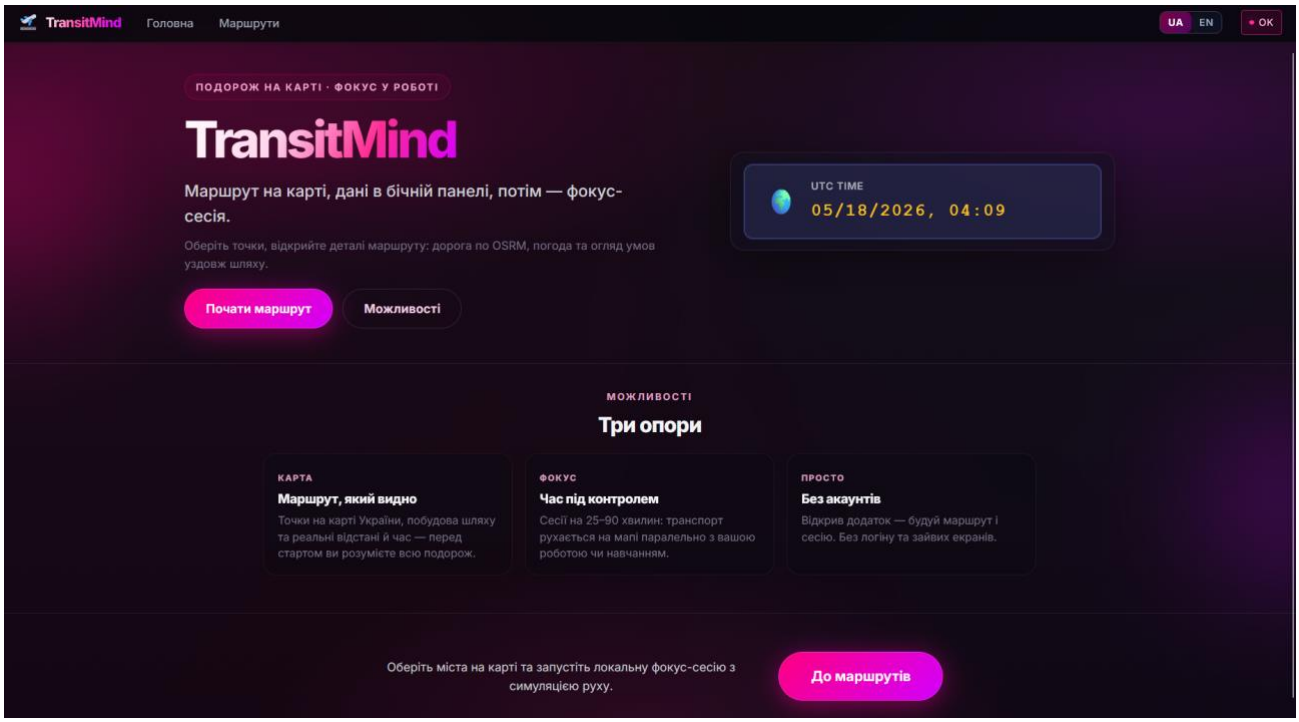


Рис. 22 Головний інтерфейс платформи TransitMind

Наступним кроком стало тестування ключового функціоналу системи — процесу «збагачення маршруту» (enrichment). Було ініційовано запит на розрахунок шляху між двома містами. Система успішно відправила HTTP-запит на бекенд, який, своєю чергою, агрегував дані з OSRM, Open-Meteo та TomTom API. Результат тестування виводу збагачених даних у бічній панелі (drawer), включаючи SVG-графік температури (sparkline) та стовпчикову діаграму заторів, зображено на рис. 23.

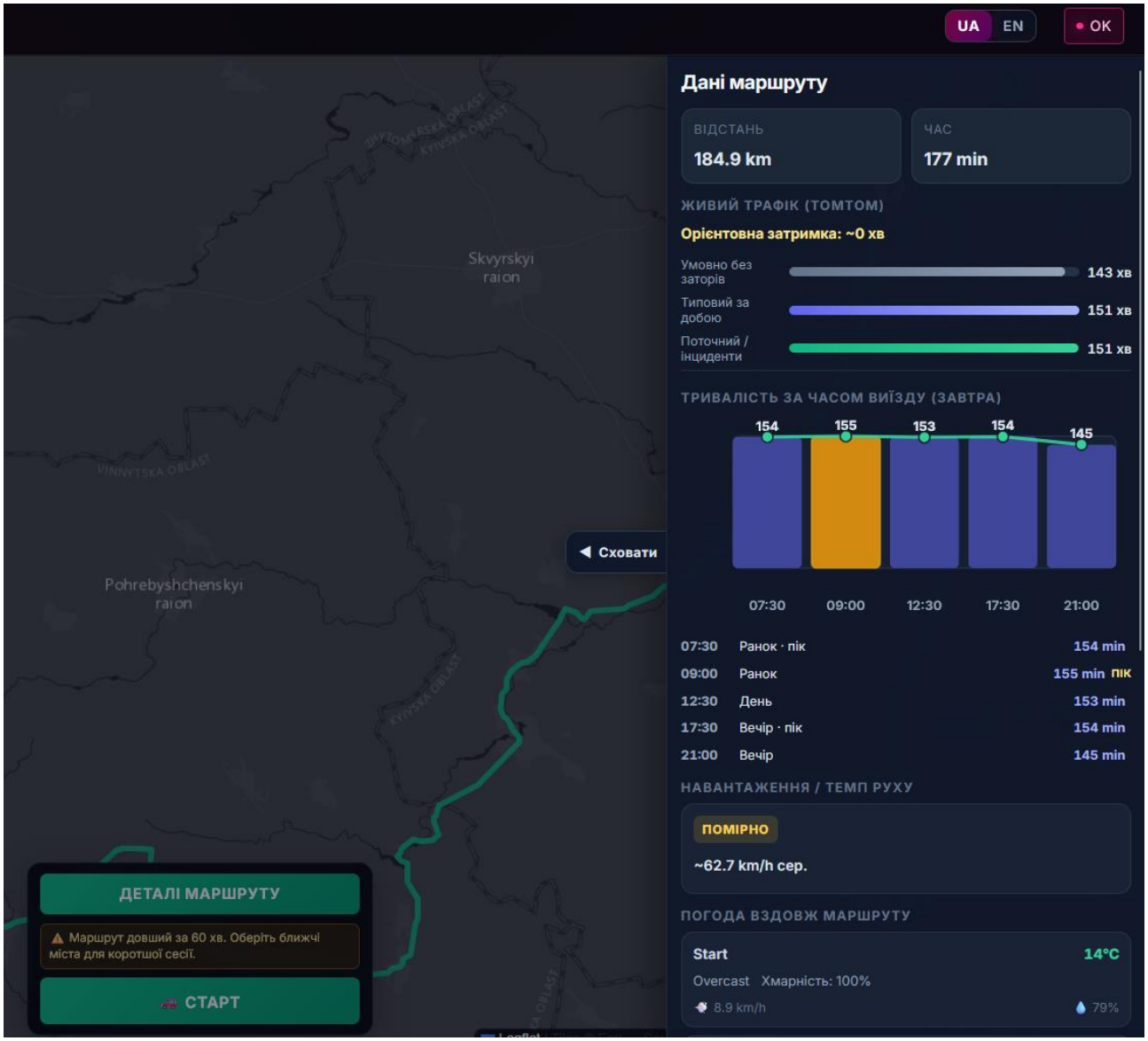


Рис. 23 Панель збагачених даних маршруту

Завершальним етапом тестування стала перевірка режиму «Фокус-сесії». Користувач натиснув кнопку «Старт», після чого система успішно створила новий запис у базі даних PostgreSQL, перевела інтерфейс у повноекранний режим та запустила таймер. Анімація руху маркера вздовж полілінії працювала плавно та без затримок завдяки коректній роботі алгоритму лінійної інтерполяції. Інтерфейс активної фокус-сесії зображено на рис. 24.

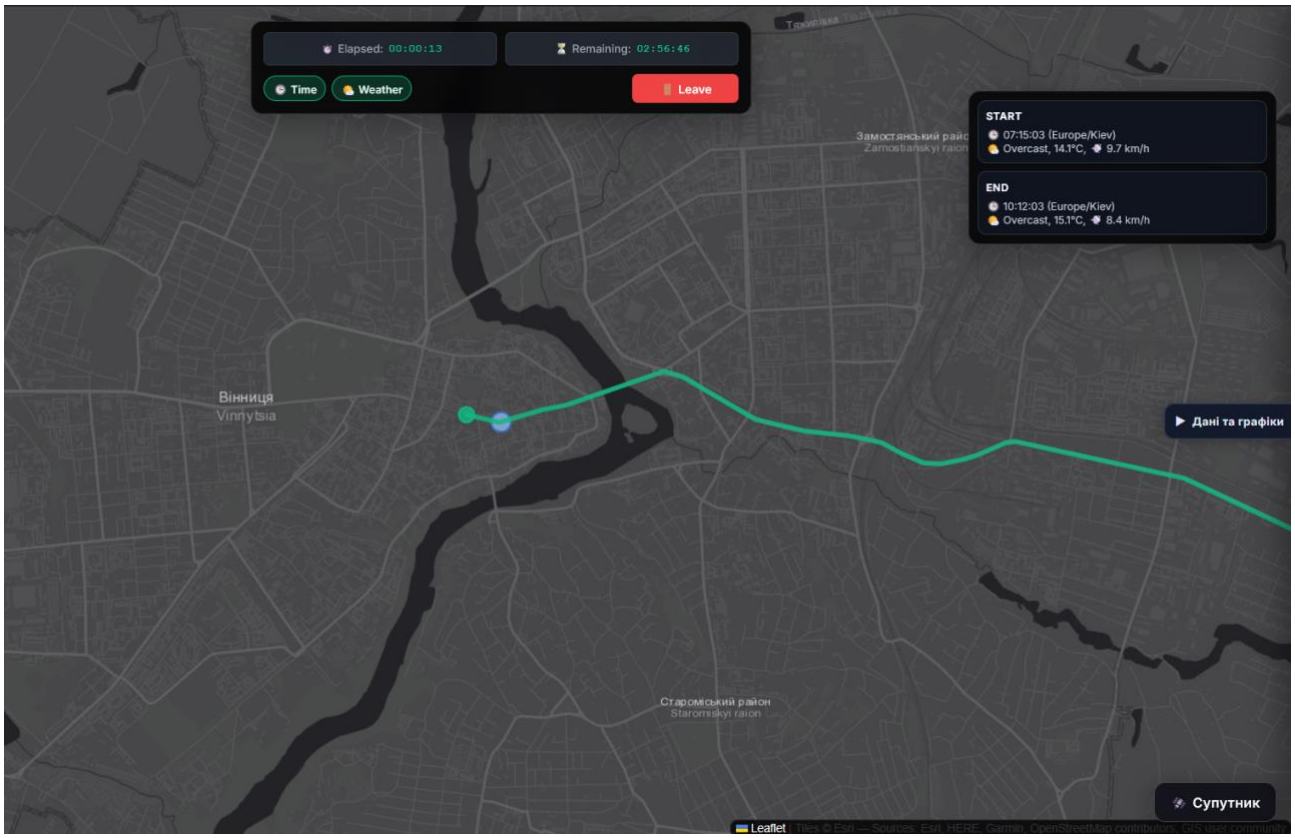


Рис. 24 Режим активної фокус-сесії

4.2 Вимоги до апаратного та програмного забезпечення

4.2.1 Діаграма розгортання. Для візуалізації розташування компонентів і їх взаємодії на різних пристроях використовують діаграму розгортання (Deployment diagram). Дана діаграма надає уявлення про архітектуру системи, відображаючи апаратні вузли (nodes), які входять до складу системи, та їхні комунікаційні взаємозв'язки.

Оскільки TransitMind є сучасним вебзастосунком (SPA), його архітектура передбачає розподіл на клієнтську машину, вебсервер та сервер бази даних, які взаємодіють через глобальну мережу. Для програмного забезпечення TransitMind було сформовано діаграму розгортання, яка зображена на рис. 25.

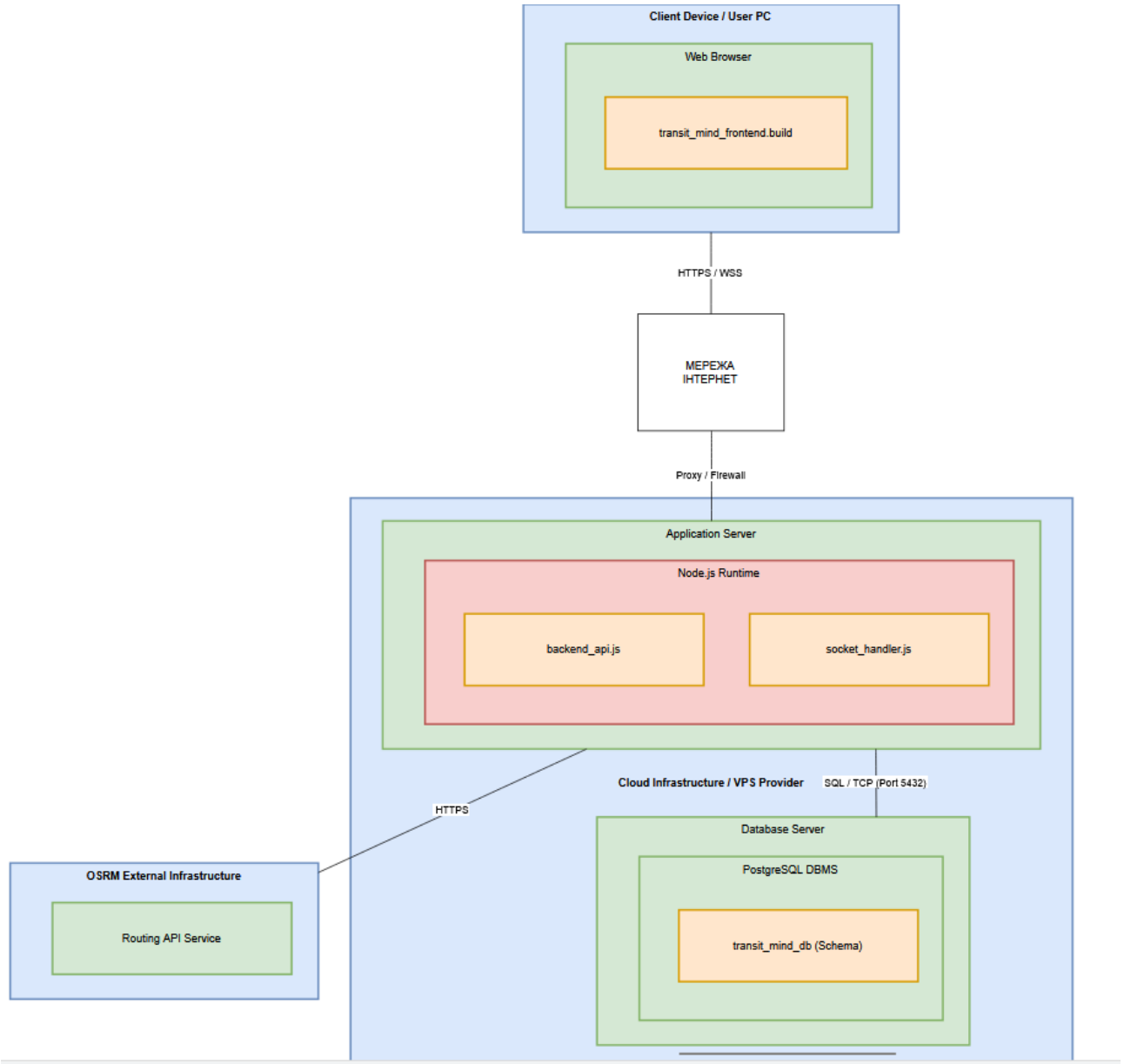


Рис. 25 Діаграма розгортання системи

На діаграмі зображено залежність компонентів системи. Робоча станція користувача (Client Device) взаємодіє з основним сервером додатків (Node.js Application Server) через захищений протокол HTTPS. Сервер додатків виконує роль проксі-шлюзу, який:

1. За допомогою протоколу TCP/IP (порт 5432) звертається до локального або віддаленого сервера бази даних (PostgreSQL Server) для збереження і зчитування маршрутів та сесій.

2. За допомогою REST/HTTP запитів звертається до вузлів сторонніх постачальників послуг (OSRM, Open-Meteo, TomTom) для отримання погодних та транспортних даних.

4.2.2 Вимоги до апаратного забезпечення. Для забезпечення коректної роботи розробленої вебплатформи необхідно враховувати апаратні та програмні вимоги як для клієнтської, так і для серверної частини.

Вимоги до клієнтської частини (пристрою користувача):

- **Процесор:** Будь-який сучасний процесор (рекомендується тактова частота від 1.8 ГГц) для плавної відмальовки картографічних тайлів.
- **Оперативна пам'ять:** Не менше 4 ГБ (для комфортної роботи браузера з графікою).
- **Програмне забезпечення:** Сучасний веббраузер із підтримкою HTML5, CSS3 та сучасного стандарту JavaScript (ES6+), наприклад Google Chrome, Mozilla Firefox або Safari. Наявність стабільного підключення до мережі Інтернет.

Вимоги до серверної частини (сервера розгортання):

- **Процесор:** 2-ядерний віртуальний процесор (vCPU) для обробки паралельних Promise-запитів до зовнішніх API.
- **Оперативна пам'ять:** Не менше 2 ГБ RAM для середовища Node.js та кешування даних у пам'яті.
- **Дисковий простір:** Від 20 ГБ SSD для операційної системи, файлів проєкту та зберігання даних СУБД.
- **Програмне забезпечення:** Встановлене середовище виконання Node.js (версії 18 або вище), пакетний менеджер npm, система управління базами даних PostgreSQL (версії 14 або вище).

4.3 Склад інсталяційного пакета

Оскільки розроблене програмне забезпечення використовує клієнт-серверну веб-архітектуру, інсталяційний пакет (архів проєкту) містить вихідні коди та конфігураційні файли для подальшого розгортання на сервері.

Для адміністратора системи інсталяційний пакет складається з наступних елементів:

1. **Директорія frontend/**: містить вихідний код клієнтського React-застосунку, файли стилів, SVG-ресурси та конфігурацію збирача Vite (vite.config.js).
2. **Директорія backend/**: містить вихідний код сервера на базі Express, контролери, алгоритми (зокрема, dijkstra.js), файли інтеграцій з API та DDL-скрипти міграції бази даних (каталог migrations/).
3. **Конфігураційні файли середовища (.env)**: шаблони файлів для вказування системних портів, URL-адрес підключення до бази даних (DB_HOST, DB_NAME, DB_USER) та приватних ключів API (наприклад, TOMTOM_API_KEY).
4. **Файли залежностей (package.json)**: містять перелік усіх необхідних NPM-модулів (axios, luxon, pg, leaflet) для автоматичного завантаження під час інсталяції.

Для розгортання системи адміністратору достатньо розгорнути базу даних PostgreSQL, виконати команду npm install для встановлення залежностей та запустити сервер командою npm start.

ВИСНОВКИ

Дана бакалаврська кваліфікаційна робота присвячена розробці сучасного програмного забезпечення картографічної платформи TransitMind, призначеної для планування маршрутів та проведення фокус-сесій. Під час виконання роботи було проведено глибокий системний аналіз предметної області, досліджено існуючі аналоги (Google Maps, Waze, OSRM) та визначено функціональні вимоги до системи, головною з яких стала необхідність агрегації контексту середовища без перевантаження інтерфейсу користувача.

У процесі проєктування було розроблено логічну модель реляційної бази даних, яка пройшла оптимізацію та містить лише критично важливі сутності: міста,

ребра дорожнього графа, збережені маршрути та сесії. За допомогою інструментарію UML було створено діаграми варіантів використання, послідовності, класів та розгортання, що забезпечило чітке розуміння архітектури системи.

Практичним результатом дослідження стала працююча вебплатформа, що використовує сучасний стек технологій. Для клієнтської частини застосовано бібліотеку React, картографічний рушій Leaflet та збирач Vite. Серверна частина, яка виступає єдиною точкою входу та агрегатором зовнішніх даних, реалізована на платформі Node.js із використанням фреймворку Express. Збереження та обробку даних забезпечує СУБД PostgreSQL.

Особливу увагу в роботі приділено алгоритмічному забезпеченню: реалізовано алгоритм Дейкстри для побудови найкоротших шляхів на графі міст, інтегровано технологію маршрутизації OSRM, та розроблено механізм паралельної агрегації даних (збагачення) з використанням сервісів Open-Meteo та TomTom API. Додатково було імплементовано алгоритм лінійної інтерполяції для візуалізації руху маркера під час проведення віртуальної фокус-сесії.

У завершальній частині роботи проведено тестування системи методом «чорної скриньки», яке підтвердило її стабільність та відповідність поставленим вимогам. Сформовано рекомендації щодо апаратного та програмного забезпечення для успішного впровадження розробленого продукту в експлуатацію.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Каграманова Ю. К. Як будувати UML-діаграми. URL: https://dou.ua/forums/topic/40575/ (дата звернення: 01.04.2026).
2. Зосим М. Діаграма послідовності (sequence diagrams). URL: https://www.maxzosim.com/sequence-diagrams/ (дата звернення: 01.04.2026).
3. Вокер А. Діаграма діяльності в UML: символ, компоненти та приклад. URL: https://www.guru99.com/uk/uml-activity-diagram.html (дата звернення: 01.04.2026).
4. Google Maps Platform. Документація та огляд можливостей картографічного сервісу. URL: https://mapsplatform.google.com/ (дата звернення: 01.04.2026).
5. Waze. Платформа соціальної навігації та краудсорсингу трафіку. URL: https://www.waze.com/ (дата звернення: 01.04.2026).
6. Open Source Routing Machine (OSRM). Офіційна документація базового вебклієнта та рушія маршрутизації. URL: http://project-osrm.org/ (дата звернення: 01.04.2026).
7. Петерсон Р. Модель діаграми зв’язків сутностей (ER) із прикладом СУБД. URL: https://www.guru99.com/uk/er-diagram-tutorial-dbms.html (дата звернення: 01.04.2026).
8. Фаулер М. UML. Основи. Короткий посібник зі стандартної мови об'єктного моделювання. Київ: Видавництво, 2020. 192 с.
9. Діаграма кооперації (Collaboration diagram) в UML. Структура та призначення. URL: https://uk.wikipedia.org/wiki/Діаграма_кооперації (дата звернення: 01.04.2026).
10. Офіційна документація збирача Vite та екосистеми React. URL: https://vitejs.dev/guide/ (дата звернення: 01.04.2026).

11. Архітектура компонентів у сучасних вебзастосунках на базі Node.js. URL: https://nodejs.org/uk/docs/guides/ (дата звернення: 01.04.2026).
12. Офіційна документація PostgreSQL. URL: https://www.postgresql.org/docs/ (дата звернення: 01.04.2026).
13. Офіційна документація фреймворку Express.js. URL: https://expressjs.com/ (дата звернення: 01.04.2026).
14. Фленеган Д. JavaScript. Докладний посібник. 7-ме вид. Київ: Альфа, 2021. 720 с.
15. Документація картографічної бібліотеки Leaflet. URL: https://leafletjs.com/reference.html (дата звернення: 01.04.2026).
16. Куликов К. Тестування програмного забезпечення. Базовий курс. Київ: ВД «Професіонал», 2021. 296 с.
17. Басс Л., Клементс П., Кацман Р. Архітектура програмного забезпечення на практиці. 3-тє вид. Київ: Пітер, 2019. 608 с.
18. Рекомендації з розгортання клієнт-серверних веб-додатків. URL: https://developer.mozilla.org/uk/ (дата звернення: 01.04.2026).

Додаток А

Код SQL-запитів для створення БД

Сторінок - 5

Київ – 2026

```
-- Створення основної бази даних платформи
CREATE DATABASE transitmind_db
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    CONNECTION LIMIT = -1;

-- Перехід до створеної бази даних (виконується в консолі psql)
\c transitmind_db;


------------------- ОЧИЩЕННЯ ІСНУЮЧИХ СУТНОСТЕЙ -------------------

-- Видалення таблиць, якщо вони вже існують (каскадне видалення)
DROP TABLE IF EXISTS sessions CASCADE;
DROP TABLE IF EXISTS routes CASCADE;
DROP TABLE IF EXISTS roads CASCADE;
DROP TABLE IF EXISTS cities CASCADE;

-- Видалення функцій та уявлень
DROP FUNCTION IF EXISTS update_updated_at_column CASCADE;
DROP VIEW IF EXISTS route_details_view CASCADE;
DROP VIEW IF EXISTS active_sessions_view CASCADE;


------------------- СТВОРЕННЯ ТАБЛИЦЬ -------------------

-- 1. Створення таблиці міст (вершин графа)
CREATE TABLE cities (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    lat NUMERIC(10, 6) NOT NULL,
    lng NUMERIC(10, 6) NOT NULL,
```

```
    region VARCHAR(100),
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- 2. Створення таблиці доріг (ребер графа)
CREATE TABLE roads (
    id SERIAL PRIMARY KEY,
    city_from_id INTEGER NOT NULL,
    city_to_id INTEGER NOT NULL,
    distance_km NUMERIC(10, 2) NOT NULL,
    driving_time_minutes INTEGER NOT NULL,
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (city_from_id) REFERENCES cities (id) ON DELETE CASCADE,
    FOREIGN KEY (city_to_id) REFERENCES cities (id) ON DELETE CASCADE
);

-- 3. Створення комплексної таблиці маршрутів
CREATE TABLE routes (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    transport_type VARCHAR(50) DEFAULT 'driving',
    start_city VARCHAR(100),
    end_city VARCHAR(100),
    start_lat NUMERIC(10, 8) NOT NULL,
    start_lng NUMERIC(11, 8) NOT NULL,
    end_lat NUMERIC(10, 8) NOT NULL,
    end_lng NUMERIC(11, 8) NOT NULL,
    duration_minutes INTEGER,
    distance_km NUMERIC(10, 2),
    country VARCHAR(50),
    waypoints JSONB,
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

```
-- 4. Створення таблиці фокус-сесій
CREATE TABLE sessions (
    id SERIAL PRIMARY KEY,
    route_id INTEGER NOT NULL,
    start_time TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    end_time TIMESTAMP WITHOUT TIME ZONE,
    status VARCHAR(50) DEFAULT 'active',
    session_type VARCHAR(20) DEFAULT 'focus',
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (route_id) REFERENCES routes (id) ON DELETE CASCADE
);


------------------- СТВОРЕННЯ ФУНКЦІЙ ТА ТРИГЕРІВ -------------------

-- Створення функції для автоматичного оновлення часу (updated_at)
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Створення тригера для таблиці sessions
CREATE TRIGGER set_updated_at
BEFORE UPDATE ON sessions
FOR EACH ROW
EXECUTE FUNCTION update_updated_at_column();


------------------- СТВОРЕННЯ УЯВЛЕНЬ (VIEWS) -------------------

-- Уявлення 1: Детальна статистика по створених маршрутах та кількості сесій
```

```
CREATE OR REPLACE VIEW route_details_view AS
SELECT
    r.id AS route_id,
    r.name AS route_name,
    r.start_city,
    r.end_city,
    r.distance_km,
    r.duration_minutes,
    COUNT(s.id) AS total_sessions
FROM routes r
LEFT JOIN sessions s ON r.id = s.route_id
GROUP BY r.id;

-- Уявлення 2: Моніторинг активних фокус-сесій в реальному часі
CREATE OR REPLACE VIEW active_sessions_view AS
SELECT
    s.id AS session_id,
    r.name AS route_name,
    s.start_time,
    s.status,
    s.session_type
FROM sessions s
JOIN routes r ON s.route_id = r.id
WHERE s.status = 'active';


---------- СТВОРЕННЯ РОЛЕЙ ТА ОБЛІКОВИХ ЗАПИСІВ ----------

-- Створення ролі для підключення серверної частини (Node.js/Express)
DROP ROLE IF EXISTS backend_app;
CREATE ROLE backend_app WITH LOGIN PASSWORD 'transit_secure_password';

-- Надання прав доступу до бази даних та схеми
GRANT CONNECT ON DATABASE transitmind_db TO backend_app;
GRANT USAGE ON SCHEMA public TO backend_app;
```

```
-- Надання прав на виконання DML-операцій з таблицями
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO
backend_app;

-- Надання прав на використання послідовностей (для роботи SERIAL id)
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO backend_app;
```

Додаток Б

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Ковальчук Ігор Русланович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «_____» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [✓] інструменти генеративного ШІ не використовувалися.
    - [ ] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
        - [ ] перекладу іншомовних джерел;
        - [ ] стилістичного редагування та перевірки граматики;
        - [ ] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___» _____ 2026 р. _____ **Ігор Ковальчук**
(підпис)