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 оцінки та прогнозування рівня міського транспортного трафіку»

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи _____ **Іван КОРНІЛОВ**
(підпис)

Консультант бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Виконав _____ **Максим ГЛАДКИЙ**
(підпис)

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

Гладкому Максиму Єгоровичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи
«Програмне забезпечення оцінки та прогнозування рівня міського транспортного трафіку»
затверджена наказом від «19» грудня 2025 р. № 3204 «С»

Термін подання завершеної роботи на кафедру «_22__» травня 2026 р.

Вихідні дані до роботи: опис процесів моніторингу та прогнозування транспортного трафіку; вимоги до web-орієнтованої системи побудови маршрутів і аналізу дорожнього навантаження; структура маршрутів і геопросторових даних; технології React, Node.js, PostgreSQL/PostGIS та Mapbox GL JS.

Перелік питань, що підлягають дослідженню: аналіз предметної області та постановка задачі; проєктування структури та бази даних програмної системи; розробка програмного забезпечення для аналізу та прогнозування міського руху; тестування та впровадження програмної системи

Перелік графічних документів (за необхідності).

Дата видачі завдання «__22__» ___грудня_____ 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Іван КОРНІЛОВ**
(підпис)

Консультант бакалаврської кваліфікаційної роботи _____ **Ганна ВАЙГАНГ**
(підпис)

Завдання взяв до виконання _____ **Максим ГЛАДКИЙ**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI – Artificial Intelligence, штучний інтелект

AJAX – Asynchronous JavaScript and XML, технологія асинхронного обміну даними

API – Application Programming Interface, програмний інтерфейс застосунку

CDN – Content Delivery Network, мережа доставки контенту

CI/CD – Continuous Integration / Continuous Deployment, безперервна інтеграція та розгортання

CORS – Cross-Origin Resource Sharing, механізм контролю доступу між доменами

CPU – Central Processing Unit, центральний процесор

CRUD – Create, Read, Update, Delete, базові операції роботи з даними

DB – Database, база даних

ER-діаграма – Entity-Relationship Diagram, діаграма «сутність–зв’язок»

GIS – Geographic Information System, геоінформаційна система

GPS – Global Positioning System, глобальна система позиціонування

HTML – HyperText Markup Language, мова розмітки гіпертексту

HTTP – HyperText Transfer Protocol, протокол передачі гіпертексту

JSON – JavaScript Object Notation, формат обміну даними

JWT – JSON Web Token, токен автентифікації користувача

PaaS – Platform as a Service, хмарна платформа як сервіс

RAM – Random Access Memory, оперативна пам’ять

REST – Representational State Transfer, архітектурний стиль взаємодії компонентів вебсистем

SPA – Single Page Application, односторінковий вебзастосунок

SQL – Structured Query Language, мова структурованих запитів

SRID – Spatial Reference System Identifier, ідентифікатор системи просторових координат

TCP – Transmission Control Protocol, протокол передачі даних

TTL – Time To Live, час зберігання даних у кеші

UI – User Interface, користувацький інтерфейс

UML – Unified Modeling Language, уніфікована мова моделювання

URL – Uniform Resource Locator, уніфікований локатор ресурсу

UTC – Coordinated Universal Time, координований всесвітній час

UX – User Experience, користувацький досвід

WebGL – Web Graphics Library, бібліотека графічної візуалізації у браузері

ВСТУП

Розвиток сучасної міської інфраструктури супроводжується постійним зростанням транспортного навантаження, що суттєво впливає на ефективність функціонування дорожньої мережі великих міст. Збільшення кількості транспортних засобів, нерівномірність транспортних потоків, висока інтенсивність руху в години пік та обмежена пропускна здатність міських магістралей призводять до виникнення заторів, збільшення тривалості поїздок і зниження загального рівня мобільності населення. Додатковими наслідками перевантаження транспортної системи є підвищення витрат пального, зростання рівня шкідливих викидів та ускладнення роботи міських служб і громадського транспорту.

У таких умовах особливого значення набувають інформаційні технології, орієнтовані на оперативний аналіз дорожньої ситуації, виявлення проблемних ділянок транспортної мережі та підтримку прийняття рішень під час побудови маршрутів. Використання веборієнтованих платформ, геоінформаційних сервісів, алгоритмів аналізу графів і механізмів обробки даних у реальному часі створює можливість для формування адаптивних систем моніторингу трафіку. Такі системи забезпечують отримання актуальної інформації про стан дорожнього руху, дозволяють оцінювати рівень завантаженості окремих ділянок доріг та виконувати прогнозування зміни транспортної ситуації на основі накопичених статистичних даних.

Актуальність теми роботи полягає у необхідності створення програмного забезпечення, здатного забезпечити комплексний підхід до аналізу міського трафіку, побудови маршрутів та візуалізації транспортної ситуації із використанням сучасних вебтехнологій і картографічних сервісів. Практична потреба в подібних рішеннях обумовлена необхідністю підвищення ефективності транспортного планування, оптимізації пересування користувачів та забезпечення більш раціонального використання міської дорожньої інфраструктури.

Об’єктом дослідження є процес оцінювання та прогнозування параметрів міського дорожнього руху в реальному часі.

Предметом дослідження є методи, алгоритми та програмні засоби моніторингу транспортних потоків, аналізу завантаженості дорожньої мережі та побудови маршрутів у веборієнтованих інформаційних системах.

Метою роботи є розробка програмного забезпечення для оцінювання та прогнозування рівня транспортного навантаження міської дорожньої мережі з можливістю візуалізації трафіку, аналізу дорожньої ситуації та побудови оптимальних маршрутів.

Для досягнення поставленої мети необхідно виконати такі завдання:

- провести аналіз предметної області, сучасних підходів до моніторингу транспортних потоків та існуючих програмних рішень для аналізу дорожнього руху;
- спроєктувати архітектуру програмної системи та структуру бази даних для зберігання інформації про транспортні потоки, маршрути та користувачів;
- розробити алгоритми побудови маршрутів з урахуванням рівня завантаженості дорожніх ділянок та зміни транспортної ситуації в реальному часі;
- реалізувати серверну та клієнтську частини вебзастосунку з інтеграцією картографічних сервісів;
- забезпечити візуалізацію рівнів транспортного навантаження та реалізувати модуль прогнозування дорожньої ситуації на основі історичних даних;
- провести тестування програмного забезпечення та оцінити ефективність функціонування розробленої системи.

У процесі дослідження використано методи системного аналізу, методи проєктування інформаційних систем, принципи об’єктно-орієнтованого програмування, алгоритми теорії графів для пошуку оптимальних маршрутів, а також методи аналізу та обробки даних для оцінювання транспортного навантаження.

Практичне значення отриманих результатів полягає у створенні веборієнтованого програмного засобу, який може бути використаний для аналізу транспортної ситуації, визначення перевантажених ділянок дорожньої мережі та оптимізації маршрутів пересування користувачів. Розроблений застосунок може бути адаптований для використання як окремими користувачами, так і в межах інформаційних систем моніторингу міської транспортної інфраструктури.

Апробація результатів роботи. Основні результати дослідження та положення кваліфікаційної роботи апробовано у тезах доповіді на тему «Програмне забезпечення оцінки та прогнозування рівня міського транспортного трафіку», поданих до VII Всеукраїнської науково-практичної конференції студентів та аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем – 2026», що проводиться Національним університетом біоресурсів і природокористування України.

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі проведено аналіз предметної області, розглянуто сучасні підходи до моніторингу транспортних потоків, виконано огляд існуючих програмних рішень і технологій для аналізу міського трафіку. У другому розділі описано процес проєктування програмної системи, структуру бази даних, архітектуру вебзастосунку та алгоритми побудови маршрутів. У третьому розділі наведено особливості програмної реалізації системи, результати тестування, аналіз функціонування програмного забезпечення та оцінку ефективності розробленого рішення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз міського дорожнього руху як предметної області

Розвиток сучасних міст супроводжується зростанням транспортного навантаження, що впливає на ефективність функціонування дорожньо-транспортної інфраструктури. Збільшення кількості транспортних засобів, обсягів перевезень та нерівномірний розподіл транспортних потоків у межах міської мережі призводять до перевантаження доріг, збільшення часу пересування, витрат пального та погіршення екологічної ситуації.

У таких умовах важливого значення набувають програмні системи моніторингу та прогнозування транспортних потоків. Сучасні інформаційні технології дозволяють створювати веборієнтовані платформи з інтеграцією картографічних сервісів, обробкою геопросторових даних та візуалізацією транспортного навантаження в режимі реального часу [1], [2]. Подібні рішення є складовою концепції Smart City, у межах якої транспортна інфраструктура розглядається як адаптивна динамічна система.

Міський дорожній рух характеризується високим рівнем динамічності та значною кількістю взаємопов’язаних параметрів. Основними характеристиками транспортного потоку є інтенсивність руху, щільність транспортного потоку та середня швидкість руху транспортних засобів [4]. Інтенсивність визначає кількість транспортних засобів за одиницю часу, щільність характеризує кількість автомобілів на певній ділянці дороги, а середня швидкість дозволяє оцінити ефективність функціонування дорожньої мережі та фактичний стан транспортного потоку.

Для аналізу взаємозв’язку між основними параметрами транспортного потоку використовується фундаментальна модель транспортного руху, відповідно до якої збільшення щільності транспортного потоку призводить до поступового зменшення середньої швидкості руху. Після досягнення критичного

рівня щільності система переходить у стан перевантаження, що супроводжується різким зменшенням пропускної здатності дороги та формуванням заторів (табл. 1.1).

Таблиця 1.1

Основні параметри транспортного потоку

Параметр	Характеристика	Практичне застосування
Інтенсивність руху	Кількість транспортних засобів за одиницю часу	Аналіз завантаженості доріг
Щільність потоку	Кількість транспортних засобів на одиницю довжини дороги	Визначення рівня перевантаження
Середня швидкість	Усереднена швидкість транспортного потоку	Оцінювання ефективності руху
Час затримки	Тривалість простою або уповільнення руху	Аналіз заторів та оптимізація маршрутів

Стан транспортної мережі залежить від значної кількості факторів, які мають часовий, інфраструктурний та випадковий характер. До часових факторів належать добові цикли транспортної активності, що формують так звані години пік, а також тижневі та сезонні коливання інтенсивності руху. У робочі дні найбільше навантаження на дорожню мережу спостерігається у ранкові та вечірні години, коли значна кількість населення одночасно здійснює пересування між житловими районами та діловими центрами міста.

Інфраструктурні фактори пов’язані з особливостями організації дорожньої мережі. Конфігурація перехресть, кількість смуг руху, режими роботи світлофорів, стан дорожнього покриття та наявність ділянок із обмеженою пропускною здатністю безпосередньо впливають на швидкість транспортного потоку та рівень утворення заторів. Особливо критичними є так звані «вузькі місця» транспортної мережі, де локальне перевантаження може спричиняти поширення заторів на суміжні дорожні ділянки.

Окрему групу становлять випадкові фактори, вплив яких складно передбачити заздалегідь. До таких факторів належать дорожньо-транспортні

пригоди, ремонтні роботи, перекриття доріг, погодні умови та проведення масових заходів (рис. 1.1).

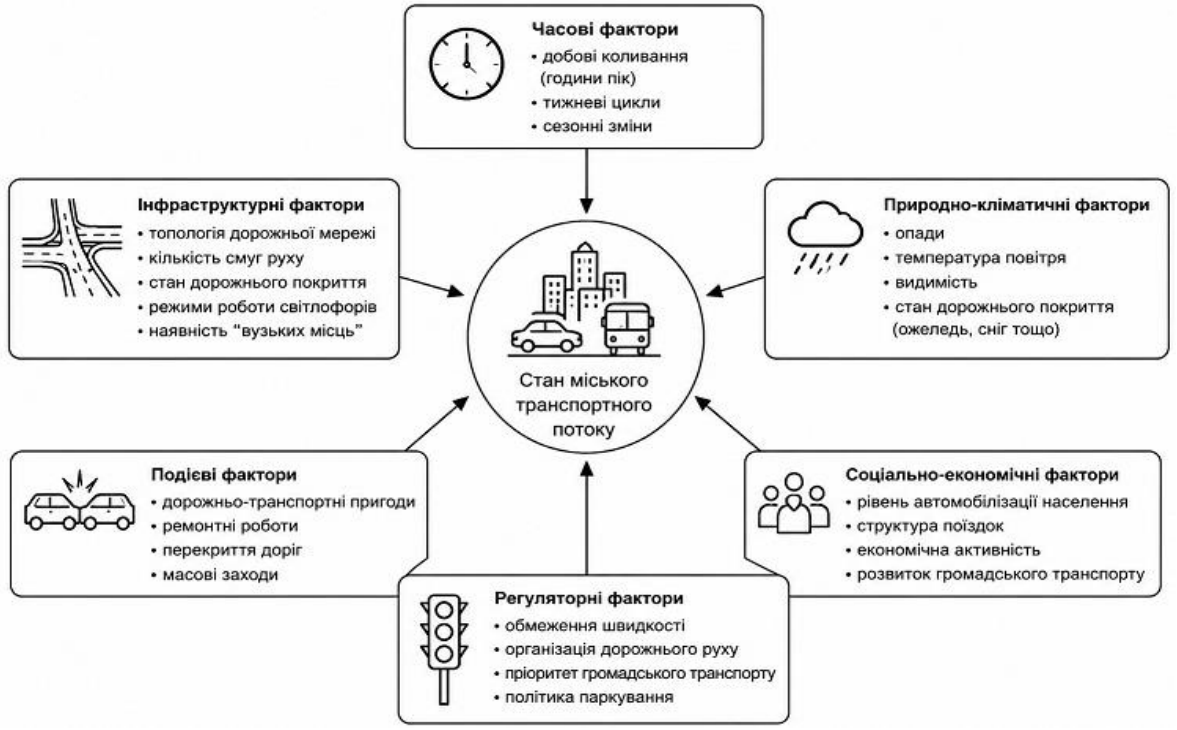

Рис. 1.1 Основні фактори впливу на стан міського транспортного потоку

Подібні події можуть суттєво змінювати транспортну ситуацію, тому системи аналізу трафіку повинні забезпечувати оперативне оновлення інформації та адаптивне коригування маршрутів.

У сучасних системах моніторингу транспортного руху важливе значення мають геоінформаційні технології та картографічні сервіси, що забезпечують візуалізацію дорожньої ситуації та побудову маршрутів у режимі реального часу [14, 15, 25, 26]. У розробленому застосунку використано інтерактивні карти з відображенням рівня завантаженості доріг за допомогою кольорового маркування.

Одним із основних джерел даних є технологія Floating Car Data, яка використовує GPS-дані зі смартфонів і навігаційних систем транспортних засобів. Додатково застосовуються дорожні камери, датчики та системи відеоспостереження, що дозволяє підвищити точність аналізу транспортної ситуації.

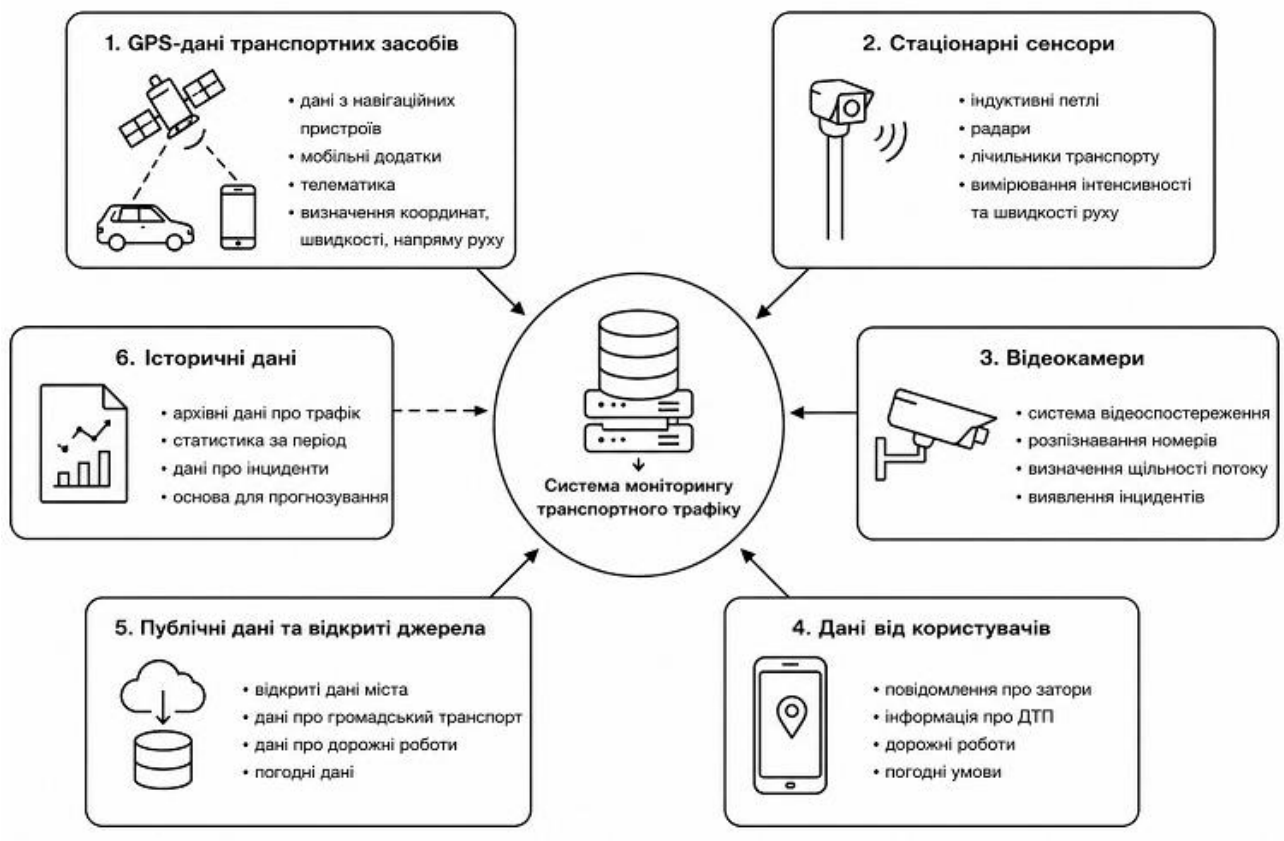

Рис. 1.2 Джерела отримання даних у системах моніторингу транспортного трафіку

Проведений аналіз предметної області дозволяє зробити висновок, що система оцінювання та прогнозування міського транспортного трафіку повинна забезпечувати виконання комплексу взаємопов’язаних функцій, серед яких збір та обробка геопросторових даних, аналіз транспортного навантаження, побудова маршрутів, візуалізація транспортної ситуації та прогнозування зміни параметрів дорожнього руху на основі історичних даних. Реалізація зазначених функцій потребує використання сучасних вебтехнологій, клієнт-серверної архітектури, засобів роботи з базами даних та алгоритмів обробки інформації у режимі реального часу [7–10].

Таким чином, міський дорожній рух є складною динамічною системою, ефективне дослідження якої потребує комплексного застосування методів аналізу даних, геоінформаційних технологій та сучасних програмних засобів веброзробки. Результати проведеного аналізу підтверджують доцільність розробки веборієнтованого програмного забезпечення для оцінювання та

прогнозування транспортного трафіку, що забезпечує можливість оперативного аналізу дорожньої ситуації та оптимізації маршрутів пересування в умовах міського середовища.

1.2 Аналіз факторів, що впливають на рівень міського трафіку

Ефективність систем аналізу та прогнозування дорожнього руху значною мірою залежить від врахування факторів, які визначають зміну параметрів транспортного потоку. Міський трафік формується під впливом сукупності взаємопов’язаних умов, частина з яких має прогнозований циклічний характер, тоді як інші виникають випадково та можуть швидко змінювати транспортну ситуацію. Для програмних систем моніторингу трафіку аналіз таких факторів є необхідним для побудови адаптивних маршрутів, оцінювання рівня завантаженості дорожньої мережі та формування прогнозних моделей.

Одним із ключових чинників є часові закономірності транспортної активності. Інтенсивність руху змінюється відповідно до добових, тижневих та сезонних циклів.

Найбільше транспортне навантаження спостерігається у ранкові та вечірні години пік, коли відбувається масове переміщення населення між житловими та діловими районами міста.

У вихідні дні транспортні потоки зміщуються у бік торговельно-розважальних і рекреаційних зон, а у святкові періоди навантаження на окремі магістралі може суттєво перевищувати середні показники [2, 5]. Врахування часових закономірностей дозволяє використовувати історичні дані для оцінювання типових сценаріїв транспортної поведінки (табл. 1.2).

Таблиця 1.2

Вплив часових факторів на транспортний потік

Фактор	Характер впливу	Наслідок для транспортної мережі
Ранкові години пік	Масове переміщення до ділових районів	Зростання щільності потоку
Вечірні години пік	Реверсний транспортний рух	Формування заторів
Вихідні дні	Переміщення до зон відпочинку	Локальні перевантаження
Святкові періоди	Підвищення транспортної активності	Перевантаження магістралей

Важливий вплив на транспортний потік мають інфраструктурні характеристики дорожньої мережі. Конфігурація перехресть, кількість смуг руху, режими роботи світлофорних систем та стан дорожнього покриття визначають пропускну здатність окремих ділянок дороги. Особливо критичними є транспортні вузли зі складною геометрією руху або обмеженою пропускною здатністю, де навіть незначне збільшення інтенсивності руху може призводити до утворення заторів. Аналіз інфраструктурних параметрів дозволяє враховувати проблемні ділянки під час побудови маршрутів та оцінювання транспортного навантаження [1, 6].

Суттєвий вплив на дорожню ситуацію мають метеорологічні умови. Атмосферні опади, ожеледиця, туман та недостатня видимість знижують середню швидкість транспортного потоку та збільшують дистанцію між транспортними засобами. У результаті зменшується пропускна здатність дорожньої мережі та підвищується ймовірність виникнення заторів. Особливо нестабільною транспортна ситуація стає під час інтенсивних опадів або різких температурних змін, коли зростає кількість дорожньо-транспортних пригод.

Окрему групу становлять випадкові події, які складно передбачити заздалегідь. До них належать дорожньо-транспортні пригоди, ремонтні роботи, перекриття дорожніх ділянок, вихід із ладу світлофорних систем та проведення масових заходів. Такі фактори можуть суттєво змінювати структуру транспортних потоків у межах короткого проміжку часу, тому системи

моніторингу трафіку повинні підтримувати механізми оперативного оновлення інформації та динамічного коригування маршрутів.

У розробленому програмному застосунку реалізовано механізм оновлення інформації про стан дорожньої мережі в режимі реального часу, що дозволяє адаптувати результати маршрутизації відповідно до поточної транспортної ситуації. Для візуалізації транспортного навантаження використовуються картографічні сервіси та кольорове маркування дорожніх ділянок залежно від рівня завантаженості.

Важливу роль також відіграють поведінкові фактори, пов’язані зі стилем керування транспортними засобами, нерівномірністю руху в потоці та реакцією водіїв на дорожні події. Хоча такі фактори складно формалізувати математично, їх вплив частково враховується через статистичний аналіз швидкості руху та історичних показників транспортної активності (табл. 1.3).

Таблиця 1.3

Характеристика факторів впливу на міський транспортний трафік

Група факторів	Основні складові	Вплив на транспортний потік
Часові	Добові та тижневі цикли	Формування годин пік
Інфраструктурні	Перехрестя, світлофори, кількість смуг	Зміна пропускної здатності
Погодні	Дощ, сніг, туман, ожеледиця	Зниження швидкості руху
Випадкові	ДТП, ремонтні роботи	Локальні перевантаження
Поведінкові	Стиль водіння, реакція водіїв	Нерівномірність руху

Отже, транспортний потік формується під впливом комплексу взаємопов’язаних факторів, які мають різний рівень прогнозованості та інтенсивності впливу. Для забезпечення точності аналізу та прогнозування дорожньої ситуації програмна система повинна враховувати як історичні закономірності транспортної активності, так і поточні зміни стану дорожньої мережі. Використання комплексного підходу до обробки транспортних даних дозволяє підвищити ефективність побудови маршрутів та оперативного оцінювання транспортного навантаження.

1.3 Огляд існуючих систем моніторингу та прогнозування трафіку

Розвиток інтелектуальних транспортних систем та цифрових сервісів геолокації зумовив появу програмних платформ, орієнтованих на моніторинг дорожньої ситуації, аналіз транспортних потоків і прогнозування навантаження на транспортну мережу. Сучасні системи поєднують засоби обробки геопросторових даних, навігаційні сервіси та інструменти інтерактивної візуалізації, що дозволяє підвищити ефективність управління транспортними потоками [4; 7].

Одним із найбільш поширених сервісів є Google Maps, який використовує геолокаційні дані мобільних пристроїв для відображення стану дорожнього руху в реальному часі. Платформа підтримує прогнозування транспортного навантаження на основі історичних даних, побудову маршрутів для різних типів транспорту та автоматичне перепланування маршруту при зміні дорожньої ситуації [8]. Водночас система має обмеження для наукових досліджень через закритий характер платформи та обмежений доступ до статистичних даних і API [9].

Сервіс Waze функціонує на основі краудсорсингового підходу та активної взаємодії користувачів. Платформа дозволяє оперативно повідомляти про дорожньо-транспортні пригоди, ремонтні роботи, перекриття руху та інші події, що впливають на транспортний потік [10]. Ефективність Waze значною мірою залежить від кількості активних користувачів, тому в регіонах із низькою активністю точність оцінювання дорожньої ситуації знижується [11].

Професійний сегмент транспортного моніторингу представлений платформами TomTom Traffic та HERE WeGo, які орієнтовані на інтеграцію з корпоративними навігаційними системами та транспортною інфраструктурою. Дані сервіси використовують високоточні картографічні моделі, телеметричні дані транспортних засобів та алгоритми оцінювання часу прибуття [12]. Проте їх

використання в академічних або невеликих програмних проєктах обмежується комерційною моделлю поширення та складністю інтеграції [13].

Альтернативою комерційним платформам є OpenStreetMap та побудовані на його основі сервіси Open Source Routing Machine і GraphHopper. Використання відкритих геоданих забезпечує можливість реалізації власних алгоритмів прогнозування, інтеграції аналітичних модулів та модифікації функціоналу відповідно до вимог програмного продукту [14]. Перевагою такого підходу є відсутність жорстких ліцензійних обмежень, підтримка REST-архітектури та можливість подальшого масштабування системи.

Для узагальнення результатів проведеного аналізу доцільно виконати порівняння основних характеристик існуючих систем моніторингу трафіку, що наведено у табл. 1.4.

Таблиця 1.4

Порівняльна характеристика систем моніторингу та прогнозування трафіку

Критерій порівняння	Google Maps	Waze	TomTom Traffic	SmartTraffic AI
Основне джерело даних	GPS-пристрої Android/iOS	Повідомлення користувачів	Автомобільні датчики та телеметрія	API та історична база даних
Тип прогнозування	Статистичне	Обмежене	Аналітичне	Алгоритмічне
Доступність API	Частково обмежена	Обмежена	Комерційна	REST JSON
Можливість інтеграції власних алгоритмів	Низька	Відсутня	Середня	Висока
Візуалізація аналітики	Інтерактивна карта	Карта подій	Професійні навігаційні модулі	Інтерактивні графіки та карта
Облік зовнішніх факторів	Частковий	Мінімальний	Передбачений	Передбачений архітектурою
Орієнтація системи	Масовий користувач	Соціальна взаємодія	Корпоративний сектор	Аналітичний вебзастосунок

Проведений аналіз свідчить, що більшість сучасних транспортних платформ орієнтована переважно на відображення поточного стану дорожньої ситуації та навігаційну підтримку користувачів. Реалізація повноцінного прогнозування транспортного навантаження, інтерактивної аналітики та відкритої інтеграції алгоритмів машинної обробки даних у багатьох системах залишається обмеженою. Значна частина комерційних платформ функціонує за принципом «закритої екосистеми», що ускладнює використання накопичених транспортних даних у дослідницьких або навчальних проєктах.

У межах розроблюваного застосунку SmartTraffic AI основна увага приділяється поєднанню засобів моніторингу транспортних потоків, інтерактивної візуалізації та алгоритмічного прогнозування в єдиному веборієнтованому середовищі. Архітектура системи передбачає можливість інтеграції зовнішніх API, накопичення історичних даних та подальшого розширення функціоналу прогнозування, що забезпечує адаптивність програмного рішення до реальних умов транспортної інфраструктури.

1.4 Аналіз сучасних технологій оцінювання транспортного навантаження

Оцінювання транспортного навантаження є важливою складовою сучасних інтелектуальних транспортних систем, оскільки від точності аналізу дорожньої ситуації залежить ефективність прогнозування трафіку та оптимізація маршрутів руху. Сучасні технології поєднують апаратні засоби збору інформації, алгоритми аналізу даних та хмарні сервіси обробки транспортних потоків [15].

Традиційними засобами контролю дорожнього руху є стаціонарні сенсори, зокрема індуктивні петлі та радарні детектори. Індуктивні системи забезпечують точний підрахунок транспортних засобів за рахунок зміни електромагнітного поля під час руху автомобіля над датчиком. Радарні системи використовують ефект Доплера для визначення швидкості транспортного потоку та можуть одночасно відстежувати декілька об’єктів на дорожній ділянці [16].

Значного поширення набули технології комп’ютерного зору, що використовують алгоритми глибокого навчання для аналізу відеопотоку. Такі системи дозволяють виконувати класифікацію транспортних засобів, визначати швидкість руху та автоматично фіксувати дорожні інциденти [17].

Перед розглядом особливостей сучасних технологій доцільно порівняти їх основні характеристики, що наведено у табл. 1.5.

Таблиця 1.5

Порівняльна характеристика технологій оцінювання транспортного навантаження

Технологія	Основний принцип роботи	Переваги	Недоліки
Індуктивні петлі	Реєстрація зміни електромагнітного поля	Висока точність	Складний монтаж
Радарні детектори	Аналіз швидкості за ефектом Доплера	Безконтактний контроль	Висока вартість
Відеоаналітика	Обробка відеопотоку засобами AI	Класифікація транспорту	Значне навантаження на систему
Floating Car Data	GPS-дані мобільних пристроїв	Масштабованість	Наявність шумів у даних
Bluetooth/Wi-Fi Sniffing	Аналіз MAC-адрес пристроїв	Низька вартість	Обмежена точність

Найбільш перспективною технологією для веборієнтованих систем моніторингу є Floating Car Data (FCD), яка базується на використанні GPS-координат мобільних пристроїв і транспортних засобів. Аналіз зміни координат та часових міток дозволяє визначати середню швидкість руху на окремих сегментах дорожньої мережі [18]. Основною перевагою FCD є можливість збору великих обсягів даних без використання дорогої фізичної інфраструктури.

Для обробки транспортних даних сучасні системи використовують технології Big Data та хмарні сервіси. Потокова обробка інформації забезпечує оперативне оновлення даних про дорожню ситуацію, а алгоритми машинного

навчання дозволяють прогнозувати транспортне навантаження на основі історичних закономірностей [19].

На рис. 1.3 наведено загальну структуру сучасної системи оцінювання транспортного навантаження, яка поєднує модулі збору даних, аналітичної обробки та прогнозування.

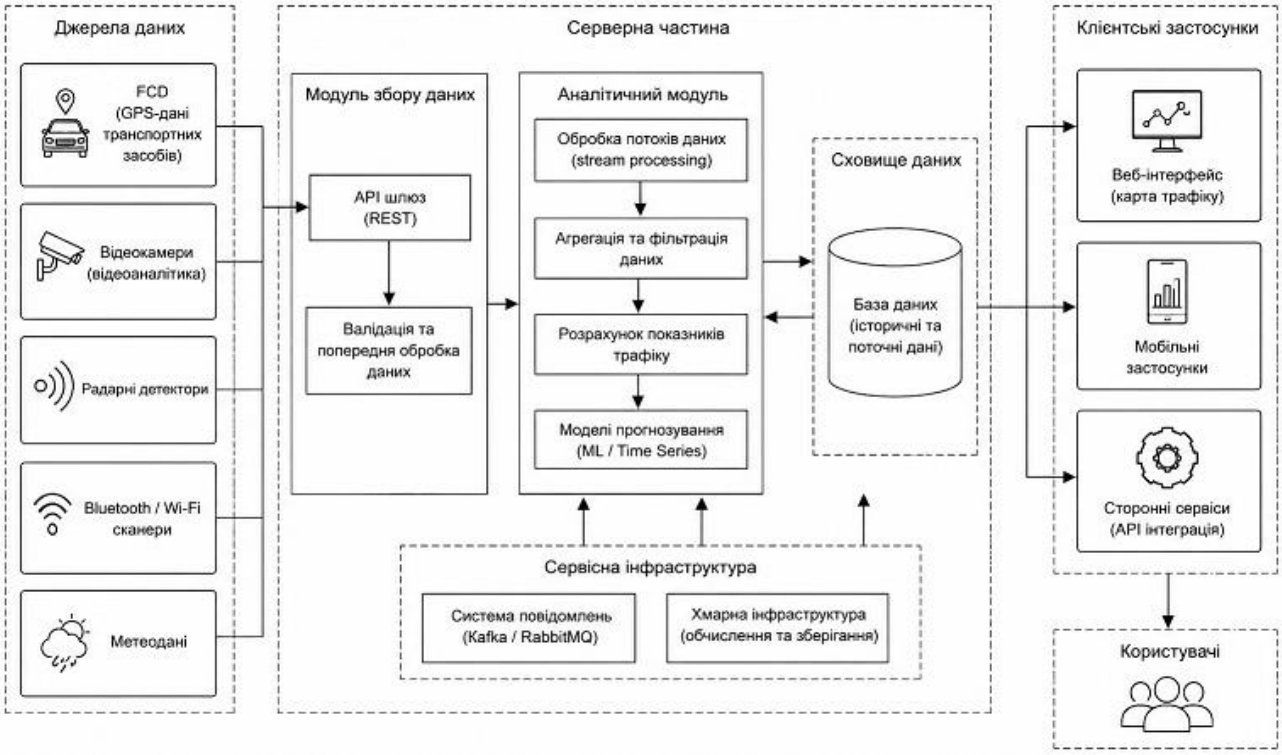

Рис. 1.3 Архітектура системи оцінювання транспортного навантаження

Проведений аналіз свідчить, що для реалізації застосунку SmartTraffic AI найбільш доцільним є використання технології Floating Car Data у поєднанні з хмарною обробкою даних та REST-архітектурою. Такий підхід забезпечує масштабованість системи, інтерактивне оновлення інформації та підтримку алгоритмів прогнозування транспортного навантаження.

1.5 Постановка задачі розробки програмного забезпечення

Результати аналізу предметної області, сучасних технологій моніторингу трафіку та існуючих програмних рішень дозволили сформувати основні вимоги до програмного забезпечення SmartTraffic AI. Розроблюваний застосунок

призначений для моніторингу транспортного навантаження, прогнозування рівня заторів та побудови маршрутів із врахуванням поточної й прогнозованої дорожньої ситуації.

Об’єктом дослідження є процес аналізу та прогнозування міського дорожнього руху в умовах змінного транспортного навантаження. Предметом дослідження виступають програмні методи обробки геоданих, алгоритми прогнозування транспортних потоків та засоби інтерактивної візуалізації інформації у веборієнтованому середовищі.

Функціонування системи передбачає інтеграцію із зовнішніми картографічними сервісами, обробку GPS-даних, формування прогнозів завантаженості доріг та відображення результатів у режимі реального часу. Для забезпечення стабільної роботи застосунку було обрано клієнт-серверну архітектуру з REST-взаємодією між компонентами системи. Серверна частина відповідає за обробку даних, формування прогнозів і роботу з базою даних, тоді як клієнтська частина забезпечує візуалізацію карти трафіку, маршрутизацію та інтерактивну взаємодію з користувачем.

Перед описом функціональних можливостей системи доцільно подати основні вимоги до програмного забезпечення, що наведені у табл. 1.6.

Таблиця 1.6

Основні вимоги до програмного забезпечення SmartTraffic AI

Категорія вимог	Характеристика
Продуктивність	Час формування маршруту не більше 500 мс
Масштабованість	Підтримка одночасної роботи декількох користувачів
Адаптивність	Коректне відображення на ПК та мобільних пристроях
Надійність	Валідація вхідних даних та обробка помилок
Безпека	Авторизація користувачів за JWT
Доступність	Робота системи через вебінтерфейс та хмарне розгортання

Основними функціями системи є візуалізація дорожнього трафіку, побудова маршрутів, прогнозування транспортного навантаження та відображення статистичної інформації у графічному вигляді. Користувач може

здійснювати пошук адрес, отримувати інформацію про поточний стан доріг та переглядати прогноз зміни трафіку залежно від обраного часу.

Для формалізації взаємодії користувача із системою було побудовано діаграму прецедентів, яка відображає основні сценарії використання програмного забезпечення. На рис. 1.4 подано взаємодію користувача із сервісами авторизації, пошуку маршрутів, прогнозування та аналітики.

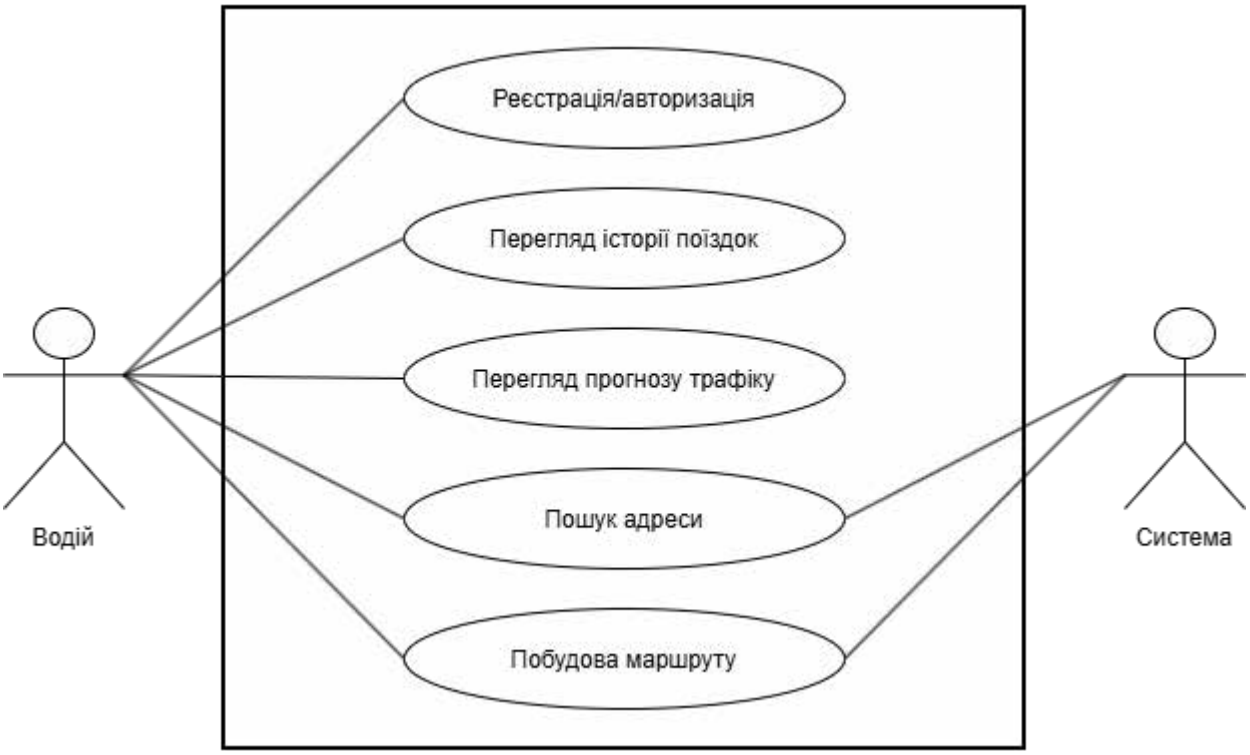

Рис. 1.4 Діаграма прецедентів системи SmartTraffic AI

Діаграма прецедентів демонструє, що центральним елементом системи є користувач, який взаємодіє з функціями побудови маршрутів, перегляду прогнозів трафіку та візуалізації статистичних даних. Для забезпечення роботи застосунку система використовує зовнішні картографічні API та сервер бази даних, де зберігається інформація про транспортні потоки та історію маршрутів.

Технологічна реалізація SmartTraffic AI базується на використанні сучасних вебтехнологій. Для серверної частини застосовано Node.js та Express, що забезпечують асинхронну обробку запитів і підтримку REST API. Клієнтська частина реалізована на React.js із використанням інтерактивних картографічних компонентів. Для зберігання просторових даних використовується PostgreSQL із

розширенням PostGIS, яке забезпечує підтримку географічних координат та операцій над ними.

Поставлена задача розробки програмного забезпечення передбачає створення масштабованої веборієнтованої системи, здатної обробляти транспортні дані у режимі реального часу, формувати прогнози дорожнього навантаження та забезпечувати користувачів актуальною інформацією про стан транспортної мережі. Такий підхід дозволяє підвищити ефективність маршрутизації та покращити інформаційну підтримку учасників дорожнього руху.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Проєктування структури програмної системи

Проєктування архітектури програмної системи SmartTraffic AI здійснювалося з урахуванням вимог до швидкодії, масштабованості, стабільності функціонування та можливості подальшого розширення функціоналу. Враховуючи специфіку обробки транспортних даних у режимі реального часу, програмне забезпечення побудоване за клієнт-серверною моделлю із розподілом функціональних обов’язків між окремими компонентами. Такий підхід забезпечує незалежність інтерфейсної частини від серверної логіки, спрощує тестування окремих модулів та дозволяє оптимізувати обмін даними між складовими системи.

Архітектура застосунку реалізована на основі компонентного підходу, за якого кожен програмний модуль виконує окрему функцію та взаємодіє з іншими компонентами через стандартизовані інтерфейси. Клієнтська частина системи створена із використанням бібліотеки React та функціонує як односторінковий вебзастосунок, що забезпечує динамічне оновлення інтерфейсу без повторного завантаження сторінки. Серверна логіка реалізована на платформі Node.js із використанням REST API для передачі даних між фронтендом і сервером. Зберігання структурованих даних організовано у СУБД PostgreSQL, яка забезпечує обробку запитів, збереження історичних даних про транспортні потоки та підтримку просторової інформації. Для побудови маршрутів і відображення дорожньої ситуації використовується інтеграція із сервісом Mapbox API.

Структура взаємодії основних компонентів програмної системи наведена на рис. 2.1.

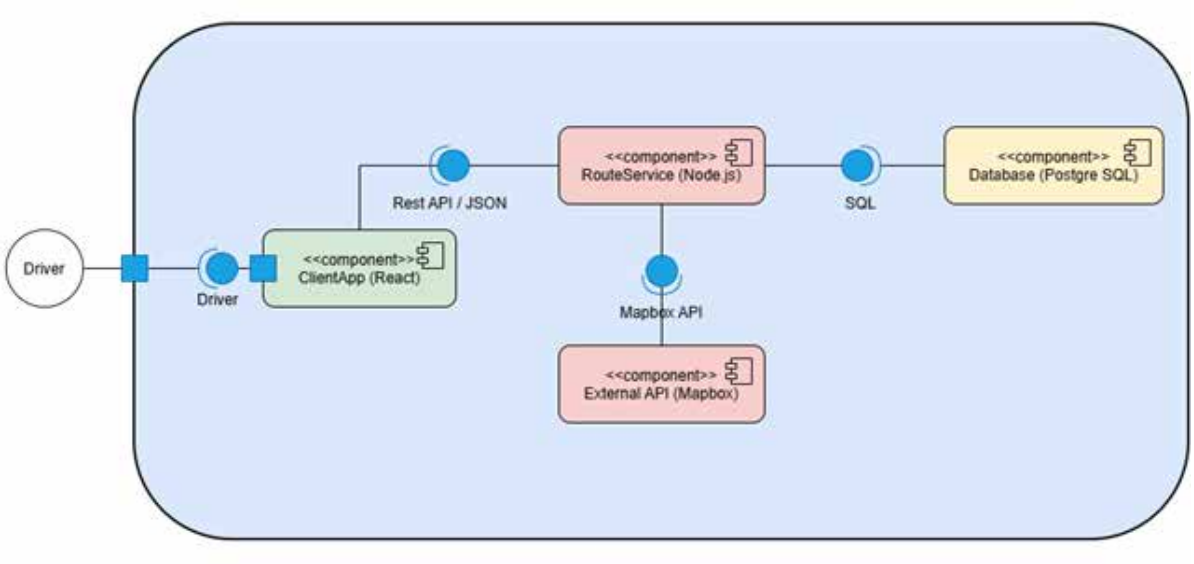

Рис. 2.1 – Архітектура програмної системи SmartTraffic AI (діаграма компонентів)

Наведена схема демонструє взаємозв'язок між клієнтським застосунком, серверною частиною, зовнішнім картографічним сервісом та системою керування базами даних. Центральним елементом архітектури є серверний модуль RouteService, який забезпечує координацію обробки запитів, обмін даними із зовнішніми сервісами та взаємодію із базою даних. Передача інформації між компонентами виконується у форматі JSON через REST API, що забезпечує універсальність інтеграції та підтримку асинхронної обробки запитів.

Основні функціональні характеристики компонентів програмної системи наведено у таблиці 2.1.

Таблиця 2.1

Характеристика основних компонентів програмної системи SmartTraffic AI

Компонент	Технологія реалізації	Призначення
ClientApp	React	Формування користувацького інтерфейсу, взаємодія з картою, відображення маршрутів і результатів аналізу
RouteService	Node.js, Express	Обробка REST-запитів, виконання серверної логіки, маршрутизація даних
Database	PostgreSQL	Збереження історичних даних, параметрів маршрутів та інформації про транспортні потоки

External API	Mapbox API	Побудова маршрутів, отримання географічних координат і дорожньої інформації

Клієнтська частина застосунку відповідає за взаємодію користувача із системою та формування візуального представлення результатів аналізу. Інтерфейс побудований за принципом компонентної структури React, що дозволяє повторно використовувати окремі елементи та підтримувати реактивне оновлення даних. Відображення карти та маршрутів виконується через інтеграцію з Mapbox GL JS, що забезпечує підтримку географічної візуалізації та динамічного оновлення дорожньої ситуації.

Серверний рівень виконує функції проміжного програмного шару між інтерфейсом користувача, зовнішнім API та базою даних. Node.js обрано завдяки підтримці асинхронної моделі виконання, що є доцільним для обробки великої кількості одночасних HTTP-запитів. Серверний модуль виконує приймання координат маршруту, формування запитів до Mapbox API, обробку отриманих результатів, а також передачу структурованих даних до клієнтської частини застосунку.

Під час функціонування системи реалізовано послідовний механізм обміну даними між компонентами. Після введення користувачем початкової та кінцевої точки маршруту клієнтський застосунок формує HTTP-запит до серверного API. Сервер виконує перевірку коректності параметрів, після чого звертається до сервісу Mapbox для отримання оптимального маршруту та інформації про стан дорожнього трафіку. Отримані результати обробляються та передаються до бази даних для збереження історії запитів і подальшого аналізу. Після завершення обробки дані повертаються у клієнтський застосунок та відображаються на інтерактивній карті.

Послідовність взаємодії компонентів програмної системи доцільно представити окремою схемою потоків даних (рис. 2.2).

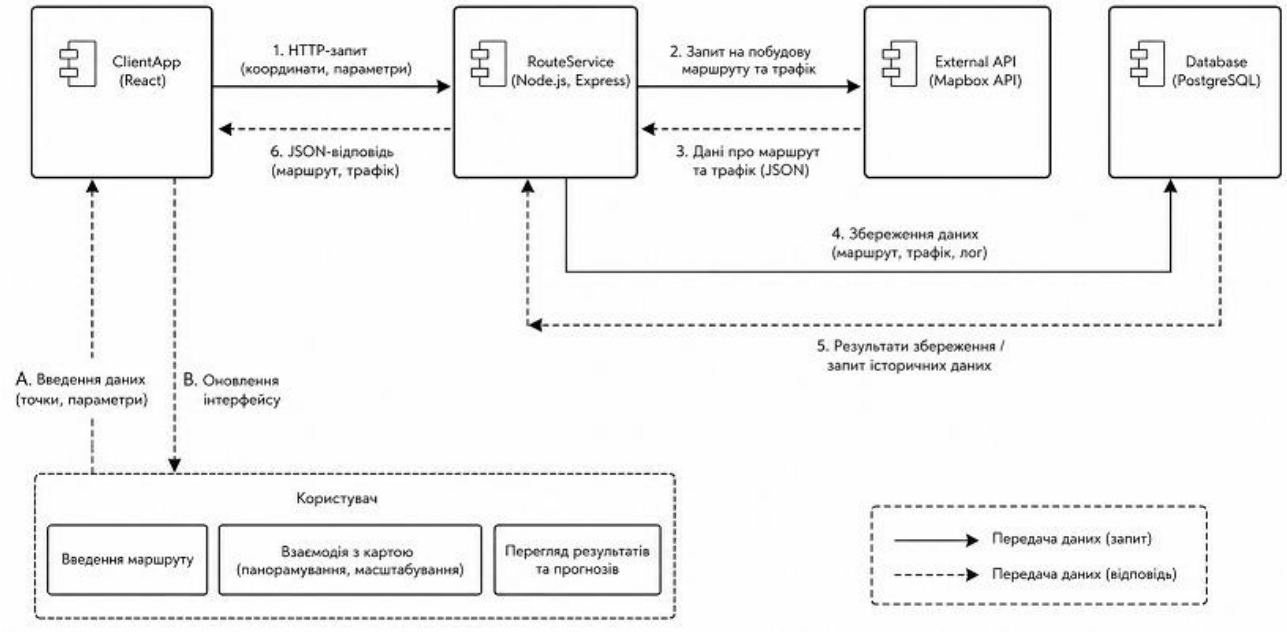

Рис. 2.2 Схема взаємодії компонентів та потоків даних у системі SmartTraffic AI

Реалізована архітектура дозволяє ізолювати функціональні модулі та забезпечити незалежне масштабування серверної та клієнтської частин системи. У разі збільшення навантаження серверні сервіси можуть бути розгорнуті окремо від інтерфейсної частини, що позитивно впливає на продуктивність застосунку. Додатковою перевагою є можливість заміни зовнішнього картографічного сервісу без суттєвого перепроєктування інших модулів системи, оскільки взаємодія реалізована через окремий програмний інтерфейс.

Важливим аспектом архітектурного проєктування є забезпечення безпеки обміну даними. Клієнтський застосунок не має прямого доступу до бази даних, а всі запити проходять через серверний API, де виконується перевірка параметрів та контроль доступу. Такий підхід знижує ризик несанкціонованого втручання у структуру даних і забезпечує централізоване керування логікою взаємодії із системою.

Отже, спроєктована архітектура SmartTraffic AI забезпечує логічний розподіл функціональних обов’язків між програмними компонентами, підтримує ефективний обмін даними між модулями та створює основу для подальшого розширення функціональних можливостей застосунку.

2.2 Логічна модель та структура бази даних

Функціонування програмної системи SmartTraffic AI потребує структурованого зберігання транспортних, географічних та аналітичних даних. Логічна модель бази даних сформована з урахуванням особливостей обробки інформації про транспортні потоки, маршрути користувачів і результати прогнозування рівня завантаженості дорожньої мережі. Основними вимогами до структури даних стали забезпечення цілісності інформації, мінімізація дублювання записів та підтримка ефективної взаємодії між серверною частиною системи й аналітичними модулями.

Логічна модель бази даних побудована за реляційним принципом та включає взаємопов’язані сутності, які описують основні об’єкти предметної області. Структура моделі забезпечує збереження інформації про користувачів системи, координати переміщення, параметри дорожнього трафіку, характеристики маршрутів і прогнозовані показники транспортного навантаження.

Базовою сутністю системи є таблиця «Водій», що використовується для збереження ідентифікаційних даних користувачів. Інформація про координати переміщення фіксується у таблиці «Місцезнаходження», яка містить значення широти, довготи та часові мітки. Показники швидкості руху та щільності транспортного потоку накопичуються у таблиці «Дані_трафіку», що дозволяє виконувати аналіз дорожньої ситуації та формувати прогнозні моделі.

Центральним елементом структури є таблиця «Ділянка_дороги», яка містить інформацію про сегменти транспортної мережі, їх довжину та рівень завантаженості. Для реалізації функцій прогнозування використовується таблиця «Прогноз», де зберігаються результати розрахунків прогнозованого транспортного навантаження.

Маршрути користувачів описуються через таблицю «Маршрут», яка містить початкову та кінцеву точки руху, а також загальний час проходження маршруту. Послідовність проходження дорожніх сегментів реалізована через

таблицю «Точка_маршруту», що забезпечує зв’язок між маршрутом та окремими ділянками дороги і дозволяє підтримувати алгоритми оптимізації транспортних шляхів.

Основні сутності логічної моделі та їх призначення наведено у табл. 2.2.

Таблиця 2.2

Основні сутності бази даних системи SmartTraffic AI

Сутність	Призначення
Водій	Збереження даних користувачів системи
Місцезнаходження	Фіксація координат та часу переміщення
Дані_трафіку	Збереження параметрів транспортного потоку
Ділянка_дороги	Опис сегментів транспортної мережі
Прогноз	Збереження результатів прогнозування
Маршрут	Формування параметрів маршруту
Точка_маршруту	Визначення послідовності проходження ділянок

Зв’язки між сутностями реалізовано за допомогою зовнішніх ключів, що забезпечує цілісність інформації та узгодженість структури даних. Між таблицями «Водій» та «Місцезнаходження» реалізовано зв’язок типу «один-до-багатьох», оскільки одному користувачу може відповідати множина координатних записів. Аналогічний тип зв’язку використовується між сутностями «Ділянка_дороги» та «Прогноз», а також між таблицями «Маршрут» і «Точка_маршруту». Така організація структури даних забезпечує можливість масштабування системи без необхідності зміни базових принципів організації бази даних.

Для візуалізації логічної структури інформаційного забезпечення розроблено ER-діаграму, яка відображає взаємозв’язки між сутностями та їх атрибутами (рис. 2.3).

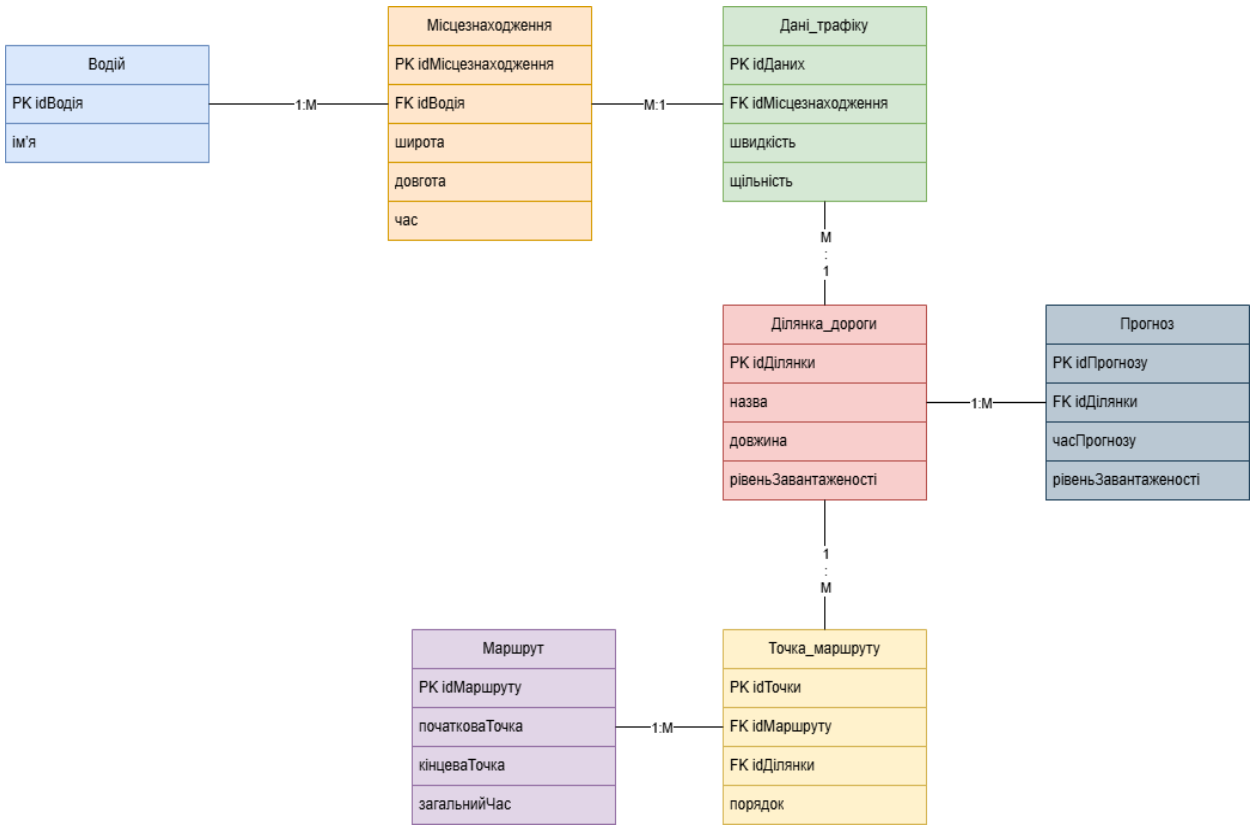

Рис. 2.3 ER-діаграма бази даних системи SmartTraffic AI

Побудована логічна модель бази даних забезпечує ефективне зберігання транспортної інформації, підтримує роботу серверної частини застосунку та створює основу для реалізації функцій аналізу й прогнозування дорожнього трафіку. Використання реляційного підходу дозволяє забезпечити структурованість даних, оптимізувати виконання запитів та підтримувати стабільну роботу програмної системи під час обробки значних обсягів інформації.

2.3 Вибір системи управління базами даних та обґрунтування інструментарію

Реалізація програмної системи SmartTraffic AI потребувала вибору системи управління базами даних, здатної забезпечити ефективне зберігання та обробку просторово-часових даних транспортної мережі. Особливістю застосунку є необхідність роботи з координатами, маршрутами, історією

переміщень та показниками дорожнього трафіку, що формує підвищені вимоги до швидкодії запитів, підтримки геопросторових операцій та стабільності роботи серверної частини.

На початковому етапі розробки розглядалися як документо-орієнтовані, так і реляційні системи управління базами даних. Одним із можливих варіантів була MongoDB, яка широко використовується у сучасних вебзастосунках завдяки підтримці JSON-подібної структури даних та простоті інтеграції з технологічним стеком Node.js і React [12]. Однак під час аналізу функціональних вимог системи було встановлено, що використання документо-орієнтованої моделі ускладнює реалізацію складних зв’язків між сутностями та виконання аналітичних запитів, пов’язаних із транспортними маршрутами й історією переміщень користувачів.

Важливим критерієм вибору СУБД стала підтримка геопросторових операцій. Для систем моніторингу транспортного трафіку необхідною є можливість швидкого визначення координатних об’єктів, обчислення відстаней між точками, аналізу маршрутів та обробки просторових запитів. Незважаючи на наявність у MongoDB базових геопросторових індексів, їх функціональність виявилася недостатньою для реалізації повноцінної роботи з дорожніми сегментами та прогнозування транспортного навантаження.

Після аналізу можливостей різних платформ було обрано реляційну систему управління базами даних PostgreSQL із розширенням PostGIS [13]. Дане рішення забезпечує підтримку географічних типів даних, просторової індексації та математичних операцій над координатами. Використання PostGIS дозволило перенести значну частину обчислень безпосередньо на рівень бази даних, що позитивно вплинуло на продуктивність серверної частини системи.

Додатковою перевагою PostgreSQL є підтримка розширених механізмів індексації та можливість роботи зі структурованими JSON-даними за допомогою типу JSONB. Це дозволяє поєднати реляційний підхід до збереження критично важливої інформації з гнучкістю обробки службових параметрів і метаданих. Використання просторових індексів GiST забезпечує швидке виконання

координатних запитів навіть за значного обсягу накопичених транспортних даних, що є важливим для систем аналізу дорожнього трафіку в реальному часі.

Порівняльну характеристику розглянутих систем управління базами даних наведено у табл. 2.3.

Таблиця 2.3

Порівняння систем управління базами даних

Характеристика	MongoDB	MySQL	PostgreSQL + PostGIS
Тип моделі даних	Документо-орієнтована	Реляційна	Об’єктно-реляційна
Підтримка геопросторових операцій	Базова	Обмежена	Розширена
Підтримка складних SQL-запитів	Часткова	Повна	Повна
Просторова індексація	2dsphere	Spatial Index	GiST, SP-GiST
Підтримка JSON-структур	Повна	Часткова	JSONB
Оптимізація роботи з маршрутами	Обмежена	Середня	Висока

Використання PostgreSQL дозволило реалізувати комбіновану структуру зберігання даних. Критично важлива інформація, зокрема дані користувачів, маршрути та параметри дорожніх сегментів, зберігається у реляційних таблицях із підтримкою зовнішніх ключів та обмежень цілісності. Для гнучких налаштувань і службових метаданих використовується тип JSONB, який дозволяє зберігати структуровані JSON-об’єкти без зміни схеми таблиць.

Однією з переваг PostgreSQL є підтримка індексів GiST, що оптимізують просторовий пошук і значно прискорюють виконання запитів під час роботи з координатами. Це особливо важливо для таблиць, які містять велику кількість записів про транспортний трафік та маршрути користувачів. Використання просторової індексації дозволяє швидко визначати найближчі дорожні сегменти та виконувати обробку географічних даних у реальному часі.

Архітектура програмної системи також потребувала вибору інструментів серверної взаємодії та розгортання застосунку. Для реалізації серверної частини використано платформу Node.js та фреймворк Express, що забезпечують підтримку асинхронної обробки HTTP-запитів і ефективну взаємодію з REST API [14]. Підключення до PostgreSQL реалізовано через бібліотеку pg, яка забезпечує створення пулу з'єднань та стабільну роботу із сервером бази даних.

Важливим фактором стала підтримка хмарного розгортання застосунку. Для розміщення серверної частини використано платформу Railway, яка забезпечує автоматизоване створення екземпляра PostgreSQL та спрощує процес конфігурації серверного середовища. Під час запуску застосунку виконується автоматична ініціалізація структури бази даних за допомогою SQL-скриптів створення таблиць та службових об'єктів.

На рис. 2.4 представлено схему взаємодії серверної частини програмної системи із СУБД та зовнішніми сервісами.

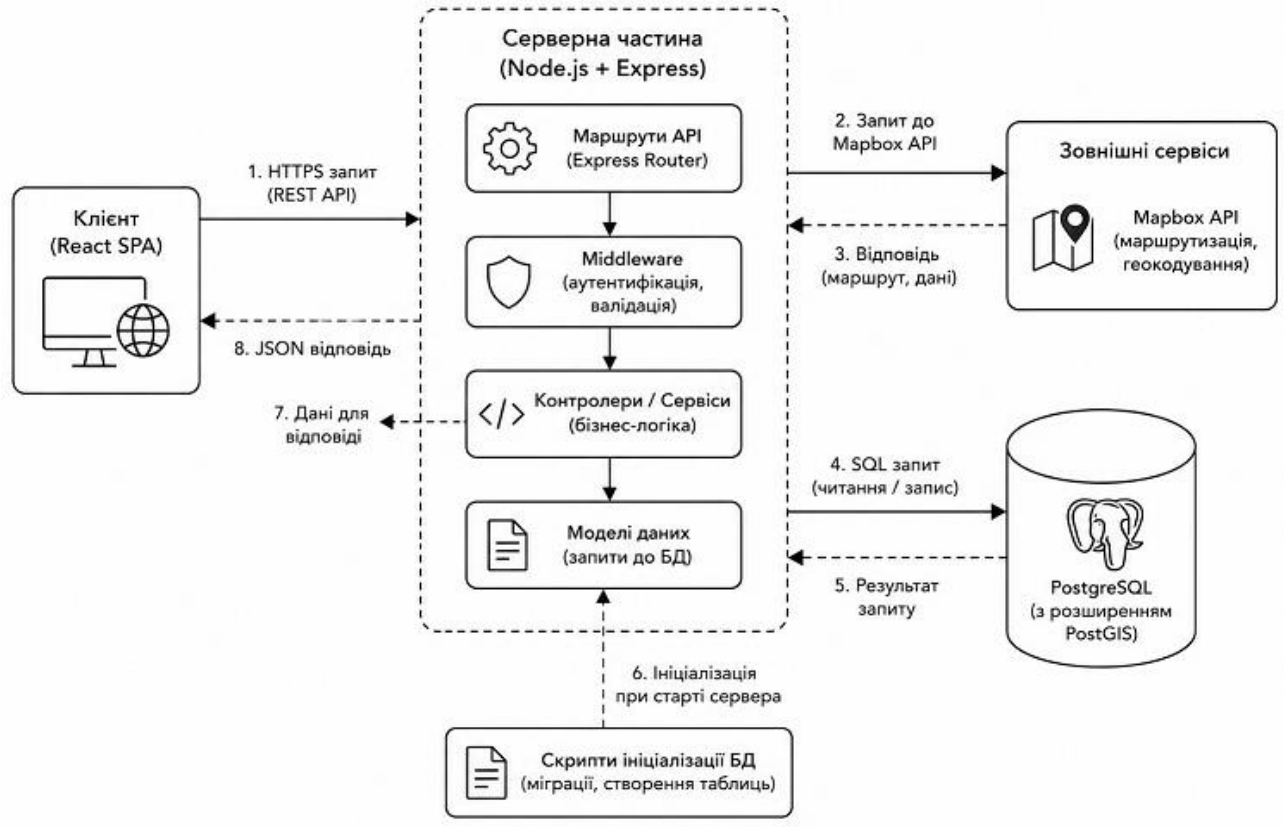

Рис. 2.4 Схема взаємодії серверної частини із PostgreSQL та зовнішніми сервісами

Проведений аналіз показав, що PostgreSQL у поєднанні з розширенням PostGIS найбільш повно відповідає вимогам системи SmartTraffic AI. Обрана СУБД забезпечує підтримку реляційної структури даних, ефективну обробку просторової інформації, високу швидкодію запитів та можливість подальшого масштабування програмного застосунку. Використання сучасного серверного інструментарію та хмарної інфраструктури дозволило реалізувати стабільну й функціонально завершену систему моніторингу та прогнозування дорожнього трафіку.

2.4 Розробка фізичної структури бази даних

Після формування логічної моделі даних та вибору PostgreSQL із розширенням PostGIS було виконано фізичне проєктування структури бази даних програмної системи SmartTraffic AI. Основною метою цього етапу стала побудова структури зберігання даних, яка забезпечує високу швидкодію під час роботи з географічними координатами, маршрутами та журналами транспортного трафіку. Під час розробки фізичної моделі враховувалися особливості функціонування вебзастосунку, пов’язані з постійним надходженням координатних даних і необхідністю оперативного виконання просторових запитів.

На початковому етапі передбачалося використання повністю нормалізованої структури, сформованої відповідно до побудованої ER-діаграми. Однак тестування прототипу показало, що виконання складних SQL-запитів із великою кількістю операцій JOIN суттєво знижує продуктивність системи під час обробки транспортних даних у реальному часі. З цієї причини під час реалізації фізичної структури бази даних було застосовано часткову денормалізацію, що дозволило зменшити кількість проміжних зв’язків між таблицями та прискорити виконання операцій читання й запису.

Фізична структура бази даних реалізована у вигляді трьох основних таблиць: users, traffic_logs та routes. Такий підхід забезпечив оптимальний баланс

між структурованістю даних, продуктивністю системи та простотою взаємодії із серверною частиною застосунку.

Таблиця users використовується для збереження облікових записів користувачів та містить основні дані, необхідні для авторизації й ідентифікації користувача у системі. Як первинний ключ використовується тип UUID, що дозволяє уникнути послідовного формування ідентифікаторів та підвищує безпеку API-запитів. Паролі користувачів у базі даних не зберігаються у відкритому вигляді, а записуються як хешовані значення, що відповідає сучасним вимогам інформаційної безпеки.

Центральним елементом фізичної структури є таблиця traffic_logs, призначена для накопичення транспортних даних та координатної інформації. Саме ця таблиця містить найбільший обсяг записів, оскільки фіксує історію переміщень користувачів, показники швидкості руху та часові мітки отриманих даних. Для збереження координат використовується просторовий тип GEOMETRY(Point, 4326), який підтримується розширенням PostGIS та дозволяє виконувати математичні операції над географічними об’єктами. Використання системи координат WGS 84 забезпечує сумісність із картографічними сервісами та GPS-даними.

Для оптимізації виконання просторових запитів на поле location накладено індекс GiST, який значно прискорює операції пошуку координатних об’єктів. Наявність просторової індексації дозволяє ефективно виконувати запити визначення найближчих транспортних ділянок, аналізу дорожнього навантаження та побудови маршрутів навіть за великого обсягу накопичених даних.

Таблиця routes призначена для збереження історії побудованих маршрутів користувачів. Початкова та кінцева точки маршруту також зберігаються у форматі GEOMETRY(Point, 4326), що забезпечує підтримку географічних операцій та інтеграцію із сервісом Mapbox API. Для зберігання структури маршруту використовується поле route_data типу JSONB, у якому записуються координати маршруту, навігаційні кроки та додаткові параметри маршрутизації

у форматі GeoJSON. Використання JSONB дозволяє швидко отримувати повну структуру маршруту без необхідності звернення до додаткових таблиць.

Основні характеристики реалізованих таблиць наведено у табл. 2.4.

Таблиця 2.4

Характеристика таблиць фізичної структури бази даних

Таблиця	Основне призначення	Ключові поля
users	Збереження облікових записів користувачів	id, name, email, password_hash
traffic_logs	Накопичення координатних та транспортних даних	id, user_id, location, speed, recorded_at
routes	Збереження маршрутів користувачів	id, user_id, start_location, end_location, route_data

Для забезпечення цілісності та узгодженості інформації у структурі бази даних реалізовано систему зовнішніх ключів, що забезпечує зв’язок між таблицями users, traffic_logs та routes. Використання механізму зовнішніх обмежень дозволяє підтримувати коректність взаємозалежних даних під час виконання операцій додавання, оновлення та видалення записів. Зокрема, кожен маршрут або запис журналу трафіку обов’язково пов’язується з конкретним користувачем системи через поле user_id, що унеможливлює появу неузгоджених або ізольованих записів у базі даних.

Реалізований підхід забезпечує стабільне функціонування серверної логіки застосунку, оскільки всі операції обробки маршрутів, координатних даних та транспортної інформації виконуються з урахуванням цілісності зв’язків між сутностями. Додатково використання зовнішніх ключів спрощує виконання SQL-запитів під час формування статистики, аналізу історії переміщень користувачів та побудови маршрутів у режимі реального часу.

Для наочного представлення структури фізичної реалізації бази даних та взаємозв’язків між основними таблицями розроблено відповідну схему, наведену на рис. 2.5.

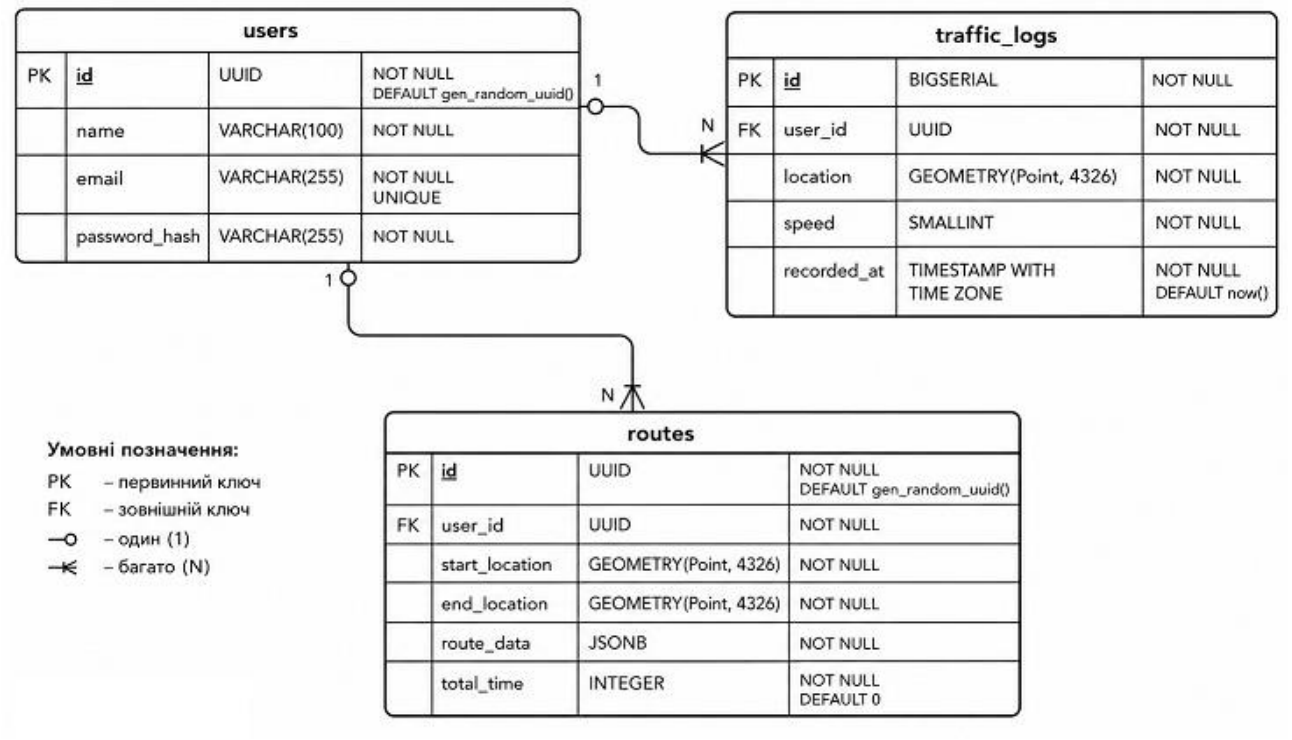

Рис. 2.5 Фізична структура бази даних системи SmartTraffic AI

Розроблена фізична модель бази даних забезпечує ефективне зберігання транспортної інформації, підтримує швидке виконання просторових SQL-запитів та відповідає вимогам систем реального часу. Використання PostgreSQL у поєднанні з розширенням PostGIS дозволило реалізувати стабільну та масштабовану структуру зберігання даних, придатну для подальшого розвитку функціональних можливостей програмної системи SmartTraffic AI.

2.5 Організація зберігання та обробки даних про трафік

Одним із ключових завдань під час розробки системи SmartTraffic AI стала організація ефективної обробки транспортних даних у режимі, наближеному до реального часу. Під час функціонування застосунку сервер постійно отримує координати користувачів, показники швидкості руху та інформацію про маршрути, що призводить до швидкого збільшення обсягу таблиці traffic_logs. За відсутності оптимізації виконання SQL-запитів це могло б суттєво знизити продуктивність системи.

Для забезпечення стабільної роботи програмного застосунку процес обробки інформації було поділено на два етапи: швидкий запис «сирих» даних та їх подальшу агрегацію для формування прогнозів транспортного трафіку. Початково розглядалася можливість виконання обчислень на стороні Node.js, однак тестування показало, що обробка великих масивів координатних даних у серверному коді створює значне навантаження на Event Loop та погіршує швидкодію застосунку.

У результаті основні операції аналізу було перенесено безпосередньо на рівень PostgreSQL із використанням можливостей PostGIS. Серверна частина формує SQL-запити для просторового групування координат, визначення дорожніх сегментів та обчислення середніх показників швидкості руху. Такий підхід дозволяє зменшити навантаження на Node.js та прискорити обробку транспортної інформації.

Під час аналізу даних система автоматично фільтрує аномальні значення, зокрема записи з дуже низькою швидкістю руху, які не відповідають параметрам транспортного потоку. Для оцінювання рівня завантаженості дороги використовується порівняння поточної швидкості із середньою швидкістю вільного руху, розрахованою на основі історичних даних нічного періоду.

Принцип визначення рівня транспортного навантаження наведено у табл. 2.5.

Таблиця 2.5

Класифікація рівня транспортного навантаження

Співвідношення швидкостей	Рівень завантаженості	Відображення на карті
Понад 75 % від еталонної швидкості	Вільний рух	Зелений
Від 40 % до 75 %	Щільний потік	Жовтий
Менше 40 %	Значний затор	Червоний

Для оптимізації взаємодії між React-інтерфейсом та сервером у системі реалізовано механізм кешування даних із використанням бібліотеки node-cache. Після першого формування прогнозу сервер зберігає агреговані результати в

оперативній пам’яті, що дозволяє уникнути повторних звернень до бази даних під час зміни часових параметрів прогнозу. Використання кешування суттєво зменшило навантаження на PostgreSQL та забезпечило плавне оновлення інформації на карті без затримок.

Схему організації зберігання, обробки та кешування транспортних даних наведено на рис. 2.6.

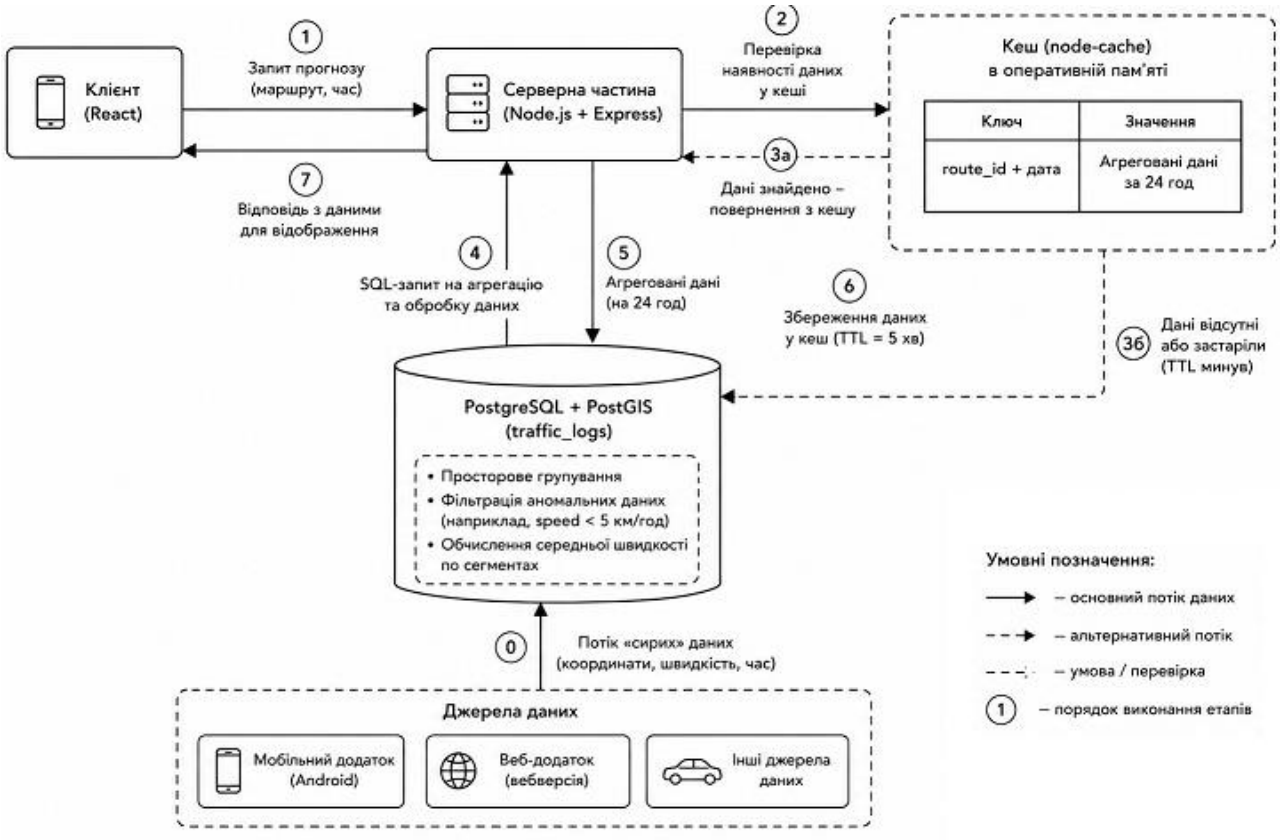

Рис. 2.6 Схема організації обробки та кешування даних транспортного трафіку

Реалізований механізм обробки транспортної інформації забезпечує ефективну роботу системи SmartTraffic AI під час обробки значного обсягу координатних даних та підтримує стабільне функціонування застосунку в режимі реального часу.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНКИ ТА ПРОГНОЗУВАННЯ МІСЬКОГО РУХУ

3.1 Реалізація архітектури та компонентів програмної системи

Після завершення етапів логічного та фізичного проєктування було виконано практичну реалізацію архітектури програмної системи SmartTraffic AI. Основною метою цього етапу стало створення гнучкої структури програмного забезпечення, здатної забезпечити стабільну взаємодію між клієнтською частиною, серверним API та базою даних під час обробки транспортної інформації у режимі реального часу. Архітектура застосунку реалізована за клієнт-серверним принципом із чітким розподілом функціональних обов'язків між компонентами системи.

Під час реалізації програмного забезпечення було прийнято рішення про фізичне розділення клієнтської та серверної частин застосунку. Такий підхід дозволив ізолювати процеси розробки, тестування та розгортання окремих модулів системи. Клієнтська частина функціонує як незалежний вебзастосунок на базі React, тоді як серверна логіка реалізована у вигляді REST API на платформі Node.js та Express. Взаємодія між компонентами здійснюється через HTTP-запити із передачею даних у форматі JSON.

Клієнтський застосунок реалізовано за моделлю Single Page Application, що забезпечує динамічне оновлення інтерфейсу без повторного завантаження сторінок. Для збірки проєкту використано Vite, який забезпечує швидке оновлення модулів під час розробки та оптимізацію фінальної збірки застосунку. Структура фронтенду організована за функціональним принципом, де окремо реалізовано компоненти інтерфейсу, сторінки маршрутизації, сервіси взаємодії з API та глобальний стан застосунку.

Серверна частина програмної системи побудована за принципом монолітної REST API архітектури. Використання єдиного серверного застосунку дозволило уникнути надмірного ускладнення інфраструктури та забезпечити централізовану обробку транспортних даних. Внутрішня структура бекенду реалізована із застосуванням багатошарової моделі, що включає маршрутизатори HTTP-запитів, контролери обробки даних та сервісний рівень бізнес-логіки. Такий підхід дозволяє ізолювати алгоритми аналізу трафіку від механізмів взаємодії з API та базою даних.

Основні компоненти програмної системи та їх функціональне призначення наведено у табл. 3.1.

Таблиця 3.1

Основні компоненти програмної системи SmartTraffic AI

Компонент	Технологія реалізації	Призначення
Frontend	React, Vite	Формування інтерфейсу користувача та візуалізація транспортних даних
Backend API	Node.js, Express	Обробка HTTP-запитів та реалізація серверної логіки
Database	PostgreSQL, PostGIS	Збереження та обробка просторових даних
Cache Layer	node-cache	Тимчасове збереження агрегованих прогнозних даних
External API	Mapbox API	Побудова маршрутів та робота з картографічними даними

Важливим аспектом реалізації архітектури стала організація безпечної взаємодії між компонентами системи. Для цього використано механізм змінних оточення, що дозволяє ізолювати конфігураційні параметри, ключі доступу та рядки підключення до бази даних від основної кодової бази застосунку. Такий підхід забезпечує безпечне розгортання програмного забезпечення у хмарному середовищі та спрощує перенесення серверної інфраструктури між різними платформами.

Під час реалізації серверної логіки особливу увагу приділено модульності компонентів. Логіку аналізу транспортного потоку та механізми маршрутизації було розділено на незалежні сервіси, що дозволило ізолювати алгоритми обробки даних від функцій побудови маршрутів. Сервіс аналізу трафіку відповідає за обчислення середньої швидкості руху, оцінювання рівня завантаженості дорожніх сегментів та агрегацію транспортних даних. Сервіс маршрутизації використовує підготовлені показники навантаження для визначення оптимального шляху пересування користувача.

Для опису структури програмних компонентів та взаємозв’язків між ними розроблено UML-діаграму класів системи управління трафіком, наведену на рис. 3.1.

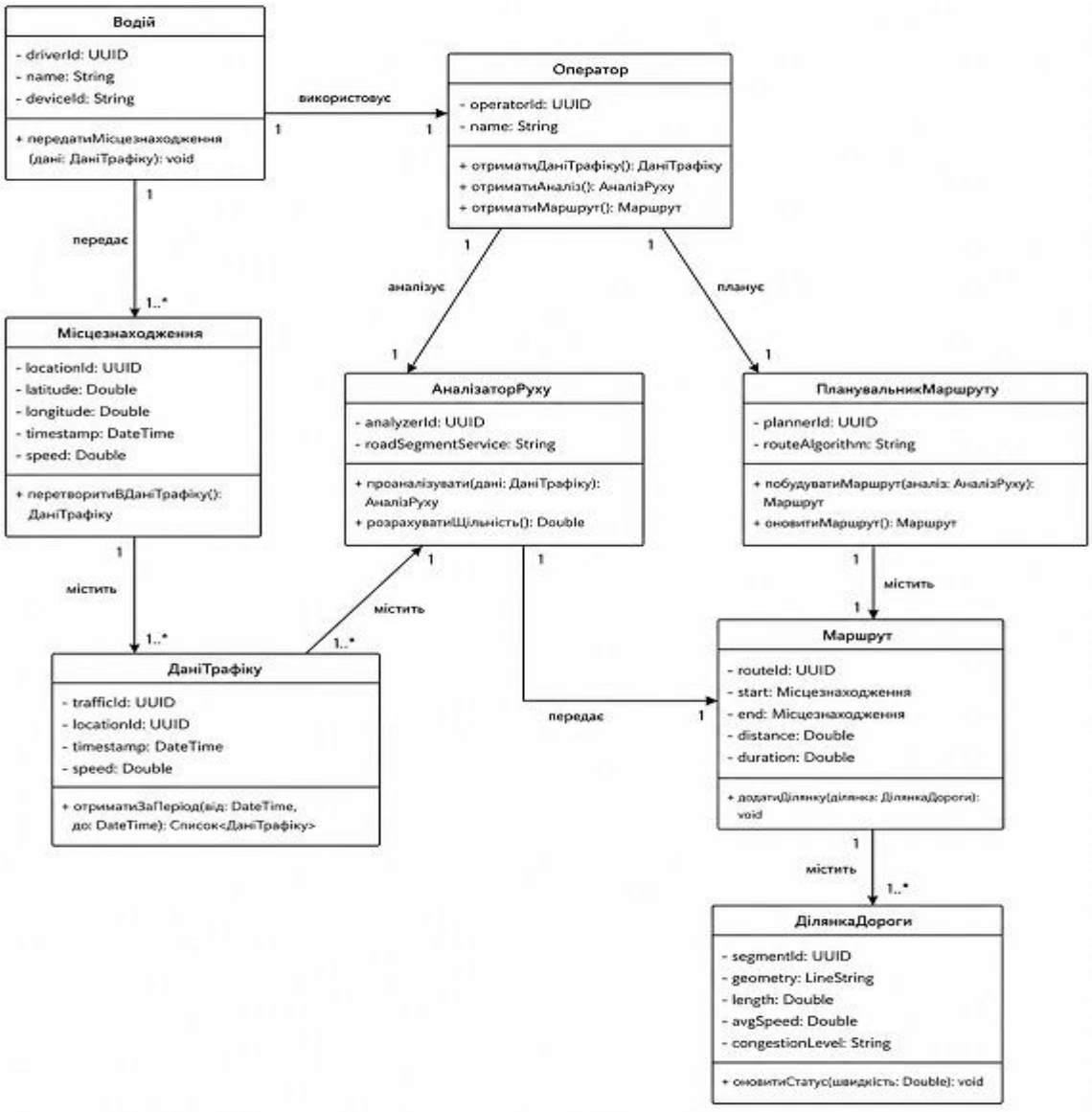

Рис. 3.1 Діаграма класів системи управління трафіком

Діаграма відображає логічний розподіл відповідальності між основними модулями застосунку та демонструє взаємодію між сервісами аналізу, маршрутизації та обробки транспортних даних. На рис. 3.1 представлено взаємозв'язки між класами «Водій», «Оператор», «Місцезнаходження», «ДаніТрафіку», «АналізаторРуху», «ПланувальникМаршруту», «Маршрут» та «ДілянкаДороги». Центральним елементом архітектури виступає клас «Оператор», який виконує роль координатора взаємодії між клієнтською частиною та сервісними модулями системи. Клас «АналізаторРуху» забезпечує обробку транспортних показників та оцінювання рівня завантаженості дорожніх сегментів, тоді як «ПланувальникМаршруту» реалізує алгоритми побудови маршрутів на основі актуальних параметрів трафіку.

Після реалізації статичної структури програмного забезпечення було виконано моделювання динамічної взаємодії між компонентами системи. Для цього побудовано UML-діаграму послідовності, яка відображає процес формування прогнозу транспортного трафіку.

На рис. 3.2 наведено діаграму послідовності для сценарію «Запит прогнозу трафіку».

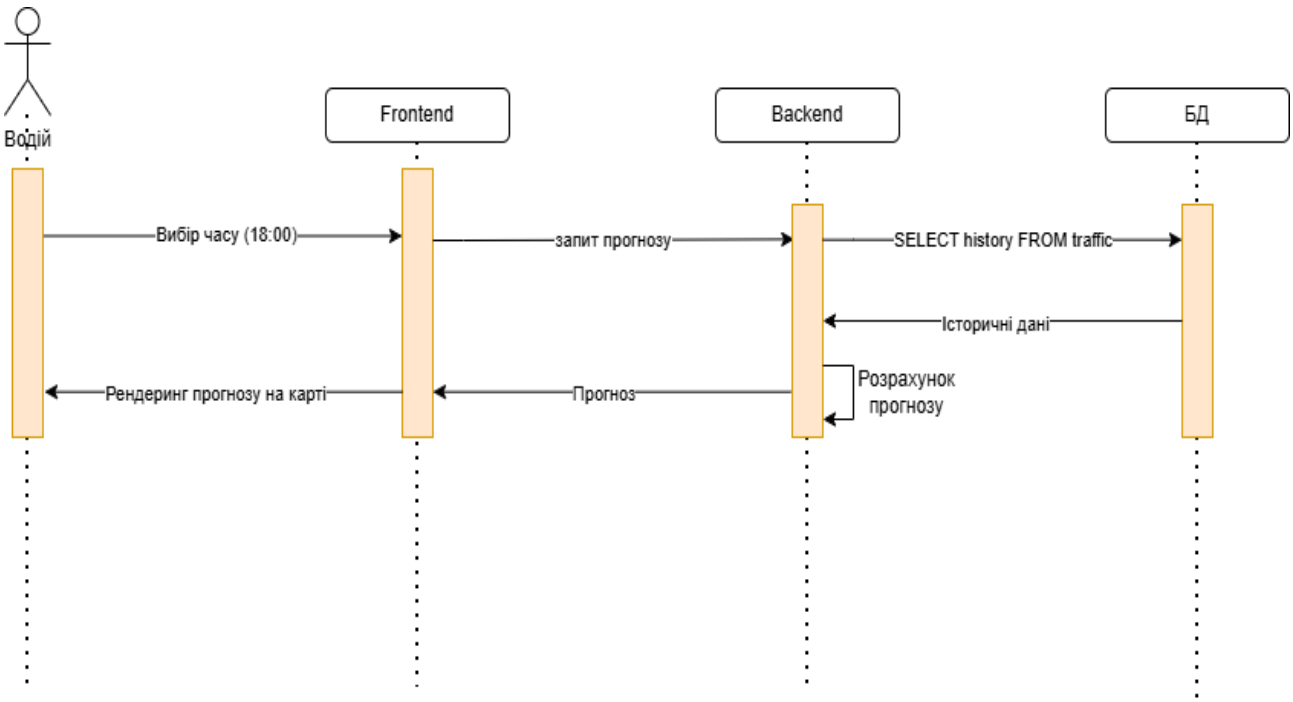

Рис. 3.2 Діаграма послідовності для сценарію "Запит прогнозу трафіку"

Процес формування прогнозу розпочинається із вибору користувачем часових параметрів у клієнтському інтерфейсі. Після цього React-застосунок формує HTTP-запит до серверного API, який передається на обробку серверному модулю. Серверна частина виконує SQL-запит до PostgreSQL для отримання історичних транспортних даних за відповідний часовий інтервал. Отримані координати та показники швидкості передаються сервісу аналізу трафіку, де виконується обробка даних та формування прогнозованого рівня завантаженості дорожніх сегментів.

Після завершення обчислень сервер формує компактний JSON-масив, що містить координати дорожніх ділянок та відповідні показники транспортного навантаження. Клієнтська частина отримує сформований прогноз та виконує оновлення інтерактивної карти із відображенням дорожньої ситуації за допомогою кольорової індикації транспортного потоку.

Реалізована архітектура програмної системи забезпечує ізоляцію функціональних компонентів, підтримує масштабованість серверної інфраструктури та дозволяє ефективно обробляти транспортні дані у режимі реального часу. Використання модульного підходу спрощує подальший розвиток програмного забезпечення та створює основу для інтеграції складніших алгоритмів прогнозування транспортних потоків.

3.2 Технологічне забезпечення та програмна реалізація системи

Розробка програмної системи SmartTraffic AI вимагала формування технологічного стеку, здатного забезпечити стабільну роботу із транспортними даними, підтримку інтерактивної картографічної візуалізації та швидку взаємодію між клієнтською і серверною частинами застосунку. Під час вибору технологій основна увага приділялася сумісності компонентів, швидкодії системи, простоті інтеграції із зовнішніми сервісами та можливості подальшого масштабування програмного забезпечення.

Для реалізації застосунку було використано стек PERN, що поєднує PostgreSQL, Express.js, React та Node.js. Використання єдиної мови програмування JavaScript як для клієнтської, так і для серверної частини дозволило спростити розробку програмного забезпечення та забезпечити узгодженість між компонентами системи.

Основні технології та інструментальні засоби, використані під час реалізації SmartTraffic AI, наведено у табл. 3.2.

Таблиця 3.2

Технологічне забезпечення програмної системи SmartTraffic AI

Технологія	Призначення
React	Розробка клієнтського інтерфейсу
Vite	Збірка та оптимізація фронтенду
Tailwind CSS	Стилізація інтерфейсу
Node.js	Серверне середовище виконання
Express.js	Реалізація REST API
PostgreSQL	Система управління базами даних
PostGIS	Обробка геопросторових даних
Mapbox GL JS	Картографічна візуалізація
Axios	Виконання HTTP-запитів
JWT	Реалізація авторизації
bcryptjs	Хешування паролів
node-cache	Кешування даних
Railway	Хмарне розгортання серверної частини
DBeaver	Адміністрування бази даних
Postman	Тестування REST API
Visual Studio Code	Середовище розробки

Клієнтська частина системи реалізована на базі React із використанням компонентного підходу до побудови інтерфейсу. Такий підхід дозволив розділити функціональні елементи застосунку на незалежні модулі, що спростило підтримку та подальше розширення системи. Для маршрутизації між

сторінками використано бібліотеку react-router-dom, яка забезпечує навігацію без повторного завантаження сторінок.

Під час реалізації інтерфейсу особливу увагу приділено роботі з інтерактивною картою. Для інтеграції картографічного сервісу використано бібліотеку Mapbox GL JS, яка забезпечує відображення дорожньої мережі, побудову маршрутів та динамічну візуалізацію транспортного навантаження. Використання технології WebGL дозволило забезпечити плавне масштабування карти та стабільне відображення великої кількості географічних об’єктів.

На рис. 3.3 наведено структуру взаємодії клієнтської частини із картографічним сервісом та серверним API.

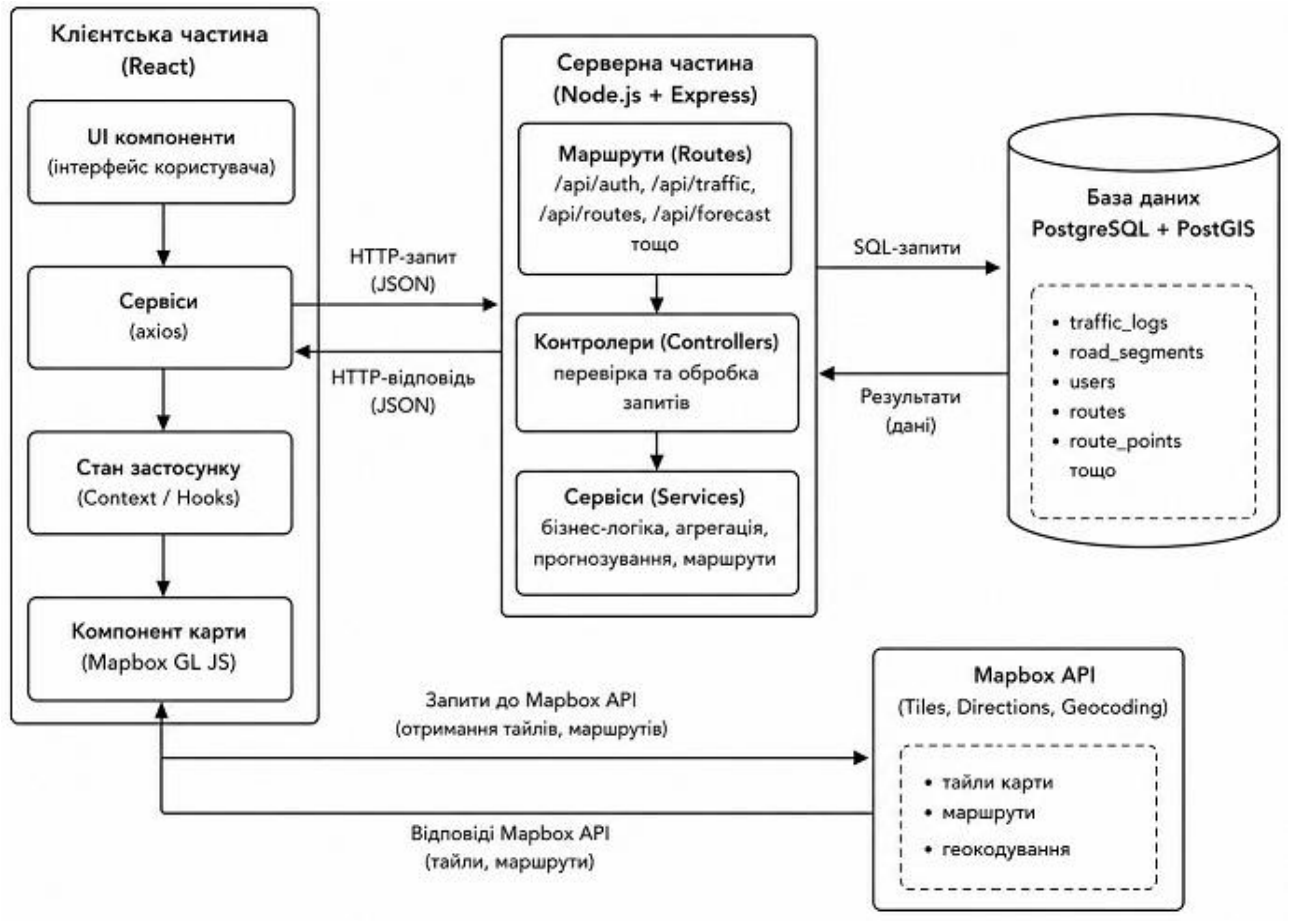

Рис. 3.3 Схема взаємодії клієнтської частини із сервером та Mapbox API

Під час реалізації React-компонентів було оптимізовано процес оновлення карти та візуалізації транспортних даних. Для уникнення повторної ініціалізації об’єкта карти використано хук useRef, який дозволяє зберігати посилання на

екземпляр Mapbox незалежно від циклу повторного рендерингу React-компонентів. Це дозволило усунути перевантаження інтерфейсу та забезпечити плавну роботу застосунку під час зміни параметрів прогнозу.

Транспортні дані, отримані від серверної частини, перетворюються у формат GeoJSON, який використовується Mapbox для побудови векторних ліній маршруту та відображення дорожнього навантаження. Кольорове маркування сегментів маршруту формується відповідно до рівня транспортного трафіку та оновлюється в реальному часі після отримання нових прогнозних даних.

Для стилізації інтерфейсу використано Tailwind CSS, який забезпечив реалізацію адаптивного користувацького інтерфейсу за принципом Mobile First. Використання utility-підходу дозволило значно скоротити обсяг окремих CSS-файлів та спростити підтримку стилів компонентів. Завдяки цьому інтерфейс застосунку коректно відображається як на настільних комп’ютерах, так і на мобільних пристроях.

Серверна частина програмної системи реалізована на платформі Node.js із використанням Express.js як каркасу для побудови REST API. Початково серверна логіка розміщувалася у єдиному файлі, однак зі збільшенням кількості функцій було виконано реорганізацію структури бекенду відповідно до модульного підходу. Серверний застосунок було розділено на маршрутизатори, контролери та модулі роботи з базою даних.

Структуру серверної частини програмної системи наведено у табл. 3.3.

Таблиця 3.3

Основні модулі серверної частини SmartTraffic AI

Модуль	Призначення
routes	Обробка HTTP-маршрутів
controllers	Валідація та обробка запитів
services	Реалізація бізнес-логіки
db.js	Підключення до PostgreSQL
middleware	Авторизація та обробка помилок
cache	Кешування прогнозних даних

Для взаємодії із PostgreSQL використано драйвер pg, який забезпечує підтримку пулу з’єднань, виконання параметризованих SQL-запитів та стабільну роботу серверної частини під час одночасного обслуговування великої кількості HTTP-запитів. Використання пулу з’єднань дозволило мінімізувати витрати часу на повторне відкриття TCP-з’єднань із сервером бази даних та підвищити продуктивність системи під час роботи із транспортними даними в режимі реального часу. Під час реалізації серверної логіки активно використовувалися нативні можливості PostgreSQL та PostGIS для виконання просторових SQL-запитів, агрегації координатних даних і формування транспортної статистики.

Від використання ORM-рішень було відмовлено через обмежену підтримку геопросторових функцій PostGIS та недостатню гнучкість під час роботи із складними SQL-конструкціями. Реалізація запитів безпосередньо мовою SQL дозволила забезпечити повний контроль над виконанням просторових операцій, оптимізувати швидкодію системи та зменшити накладні витрати під час обробки координатної інформації. Крім цього, використання параметризованих SQL-запитів дозволило підвищити рівень захисту системи від SQL-ін’єкцій та забезпечити коректну обробку користувацьких даних.

Важливим етапом реалізації серверної частини стала організація механізмів безпеки, контролю доступу та обробки помилок. Для реалізації авторизації використано JWT-токени, які дозволяють виконувати перевірку автентичності користувача без постійного збереження активних сесій на сервері. Паролі користувачів зберігаються виключно у вигляді хешованих значень із використанням бібліотеки bcryptjs, що відповідає сучасним вимогам захисту облікових даних. Додатково у серверній частині реалізовано глобальний механізм обробки помилок, який перехоплює виняткові ситуації, фіксує інформацію про помилки у журналі сервера та формує структуровані JSON-відповіді без аварійного завершення роботи застосунку.

Для тестування REST API та перевірки коректності роботи серверних ендпоінтів використовувалась програма Postman, яка дозволила виконати моделювання HTTP-запитів різних типів та перевірити обробку відповідей

сервером. Адміністрування бази даних, аналіз структури таблиць та перегляд геопросторових даних виконувалися за допомогою DBeaver. Основним середовищем розробки програмного забезпечення було Visual Studio Code із підключенням модулів ESLint та Prettier для автоматизованого контролю стилю програмного коду та підтримки єдиної структури оформлення файлів проєкту.

На рис. 3.4 наведено загальну структуру програмної реалізації системи SmartTraffic AI, яка відображає взаємодію клієнтської частини, серверного API, бази даних PostgreSQL/PostGIS та зовнішніх сервісів картографічної обробки даних.

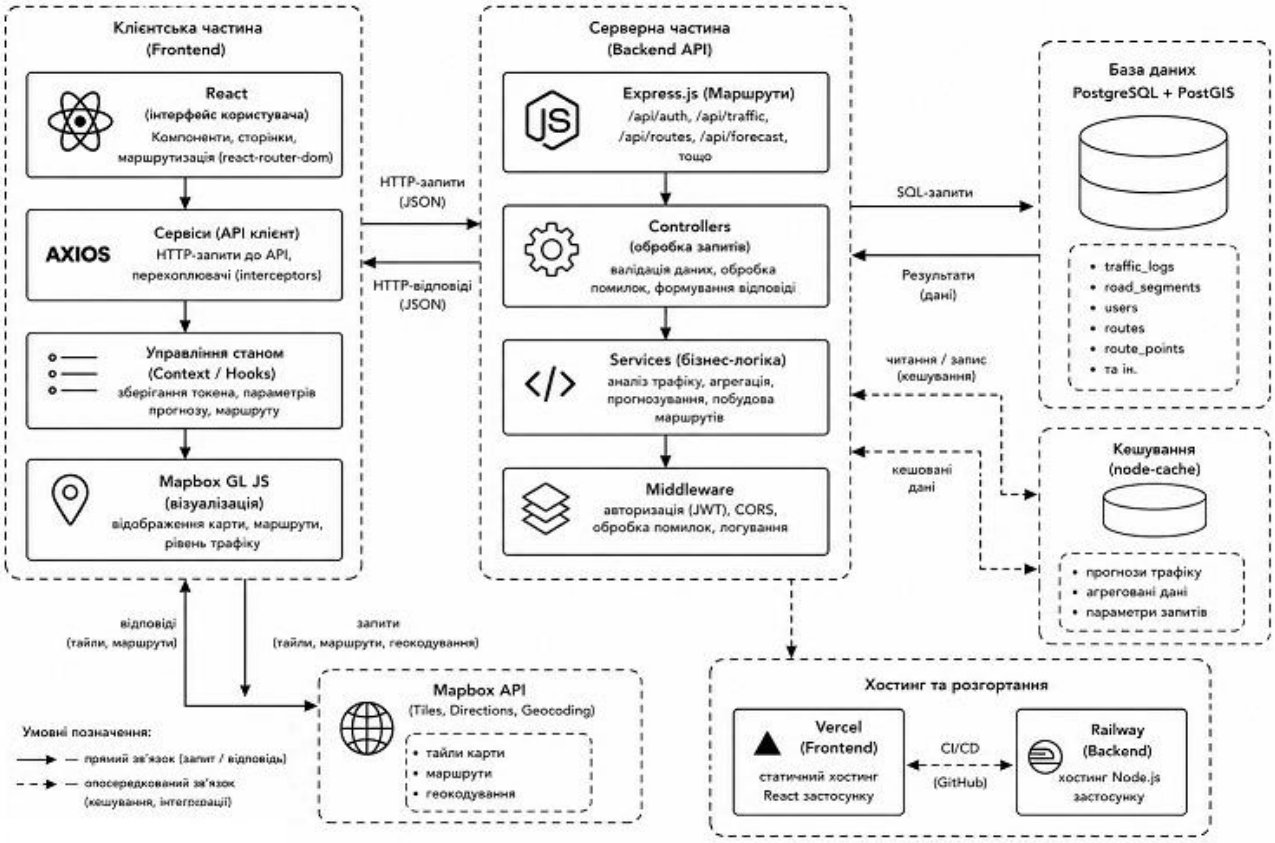

Рис. 3.4 Структура програмної реалізації системи SmartTraffic AI

Реалізоване технологічне забезпечення дозволило створити функціонально завершену систему моніторингу та прогнозування транспортного трафіку, здатну стабільно працювати із просторовими даними та підтримувати інтерактивну взаємодію із користувачем у режимі реального часу.

3.3 Реалізація модулів аналізу та прогнозування транспортного трафіку

Одним із ключових функціональних елементів системи SmartTraffic AI є модуль аналізу та прогнозування транспортного трафіку, який забезпечує обробку координатних даних, оцінювання рівня завантаженості дорожньої мережі та формування прогнозів транспортного руху на основі історичних показників. Практична реалізація цього модуля вимагала врахування низки факторів, пов’язаних із нестабільністю GPS-координат, нерівномірністю транспортних даних та необхідністю забезпечення швидкої реакції системи під час роботи в режимі реального часу.

Під час тестування застосунку на реальних маршрутах було встановлено, що координати, отримані від мобільних пристроїв, містять значну похибку позиціонування. Особливо це проявляється у щільній міській забудові, де GPS-модулі можуть зміщувати координати транспортного засобу на десятки метрів від фактичного положення. Для усунення цієї проблеми у серверній частині системи реалізовано механізм просторової фільтрації та прив’язки координат до найближчих дорожніх сегментів.

Обробка координат виконується за допомогою геопросторових функцій PostGIS, зокрема ST_DWithin та ST_ClosestPoint. Під час отримання нової координати система визначає найближчий сегмент дорожньої мережі у визначеному радіусі пошуку. Якщо відповідний сегмент дороги знайдено, координата автоматично прив’язується до нього, а записи, які не відповідають транспортному руху, виключаються з подальшої обробки. Такий підхід дозволив мінімізувати вплив GPS-дрифту та підвищити точність оцінювання дорожньої ситуації.

Додатково у модулі аналізу реалізовано механізм фільтрації аномальних значень. Із транспортного потоку автоматично виключаються записи зі швидкістю, яка не відповідає реальним умовам міського руху, а також координати нерухомих об’єктів, що не беруть участі у транспортному потоці. Це

дозволило зменшити вплив шумових даних на результати аналізу та забезпечити стабільність алгоритмів прогнозування.

Під час реалізації системи аналізу транспортного трафіку значну увагу приділено оптимізації продуктивності серверної частини та зменшенню часу обробки транспортних даних. Початково розрахунок рівня завантаженості дорожніх сегментів виконувався безпосередньо під час звернення клієнта до API, однак зі збільшенням кількості записів у таблиці traffic_logs час виконання SQL-запитів почав суттєво зростати. Це негативно впливало на швидкодію системи та призводило до затримок під час оновлення транспортної інформації на карті. Для вирішення цієї проблеми було реалізовано механізм фонової агрегації даних із використанням бібліотеки node-cron.

Фонові задачі запускаються автоматично через заданий інтервал часу та виконують обробку нових записів, накопичених у таблиці traffic_logs. Під час агрегації система виконує групування координатних даних за дорожніми сегментами, обчислює середню швидкість транспортного потоку та формує актуальні показники завантаженості для окремих ділянок дорожньої мережі. Отримані результати записуються до таблиці прогнозних показників, що дозволяє використовувати вже підготовлені дані без повторного виконання складних обчислень під час кожного звернення користувача до системи. Такий підхід суттєво зменшив навантаження на серверну інфраструктуру та підвищив швидкість формування відповідей API.

Для оцінювання рівня транспортного навантаження реалізовано окремий сервіс TrafficEvaluationService, який виконує порівняння поточної середньої швидкості руху із еталонною швидкістю вільного транспортного потоку. Додатково під час реалізації алгоритму було враховано проблему недостатньої репрезентативності вибірки. Якщо транспортний сегмент містить лише поодинокі записи за короткий проміжок часу, система не змінює статус дорожньої ділянки, оскільки такі дані можуть мати випадковий характер та не відображати реальний стан транспортного потоку. Це дозволило уникнути

хибних спрацьовувань алгоритму та підвищити достовірність прогнозованих результатів.

Принцип визначення рівня транспортного навантаження наведено у табл. 3.4.

Таблиця 3.4

Критерії оцінювання рівня транспортного навантаження

Співвідношення поточної швидкості до еталонної	Рівень трафіку	Відображення
Понад 75 %	Вільний рух	Зелений
Від 40 % до 75 %	Помірне навантаження	Жовтий
Менше 40 %	Значний затор	Червоний

Після реалізації модуля аналізу транспортного потоку було виконано розробку алгоритму прогнозування рівня міського руху. На початковому етапі розглядалася можливість використання моделей машинного навчання, однак для повноцінного навчання таких моделей необхідні значні обсяги розмічених транспортних даних за тривалий часовий період. Крім цього, інтеграція окремого середовища машинного навчання у серверну інфраструктуру значно ускладнювала б архітектуру програмної системи.

У результаті було реалізовано алгоритм прогнозування на основі статистичної агрегації часових рядів. Основою алгоритму є циклічність міського транспортного потоку, оскільки транспортне навантаження має повторюваний характер залежно від дня тижня та часу доби. Для реалізації цього підходу весь часовий інтервал тижня поділено на 168 окремих часових слотів, сформованих за комбінацією дня тижня та години доби.

Під час формування прогнозу серверна частина виконує SQL-запити із використанням функцій EXTRACT(DOW FROM timestamp) та EXTRACT(HOUR FROM timestamp), які дозволяють групувати історичні записи за відповідними часовими параметрами. На основі агрегованих даних PostgreSQL обчислює середню швидкість транспортного потоку для кожної дорожньої ділянки та формує прогнозовані показники навантаження.

Під час тестування алгоритму було виявлено проблему відсутності історичних даних для окремих дорожніх сегментів. Для усунення цієї проблеми у сервісі ForecastService реалізовано механізм каскадного резервного пошуку прогнозних значень. Якщо дані для конкретного часовго слота відсутні, система послідовно використовує усереднені показники за день тижня, за робочі дні або еталонну швидкість вільного руху. Такий підхід дозволив забезпечити безперервність формування прогнозів та уникнути появи порожніх ділянок на карті.

Для підвищення швидкодії системи у PostgreSQL реалізовано механізм матеріалізованих представлень. Створене матеріалізоване представлення traffic_forecast_profiles зберігає попередньо агреговані результати прогнозування, що дозволяє уникнути повторного виконання складних SQL-запитів під час кожного звернення користувача до системи.

Оновлення матеріалізованих представлень виконується автоматично у фоновому режимі за допомогою Cron-задач у період мінімального навантаження на сервер. Під час оновлення система виконує повторну агрегацію історичних транспортних даних та формує актуальні профілі прогнозування для окремих часових інтервалів. Використання такого підходу дозволяє уникнути повторного виконання складних SQL-обчислень під час кожного звернення користувача до системи та суттєво зменшує навантаження на серверну інфраструктуру.

Додатковою перевагою матеріалізованих представлень є можливість збереження попередньо обчислених статистичних показників без втрати швидкодії під час роботи з великими обсягами транспортних даних. Завдяки цьому система підтримує актуальність прогнозної інформації та забезпечує швидке формування відповідей API навіть за значної кількості одночасних запитів від користувачів.

На рис. 3.5 наведено схему роботи модулів аналізу та прогнозування транспортного трафіку.

Рис. 3.5 Схема роботи модулів аналізу та прогнозування транспортного трафіку

Подана схема (рис. 3.5) відображає послідовність обробки транспортних даних, починаючи від отримання GPS-координат та їх просторової фільтрації до етапів агрегації, оцінювання рівня транспортного навантаження та формування прогнозів на основі історичних даних. На схемі показано взаємодію між модулями збору координат, сервісом аналізу трафіку, системою агрегації статистичних показників та модулем прогнозування транспортного потоку.

Особливістю реалізованого підходу є поєднання фонової обробки транспортних даних із механізмами попередньої агрегації результатів. Це дозволяє мінімізувати кількість ресурсоємних SQL-запитів під час одночасної роботи великої кількості користувачів та забезпечити стабільну швидкодію серверної частини системи. Значна частина обчислень виконується асинхронно за допомогою Cron-задач, що дозволяє рівномірно розподілити навантаження на серверну інфраструктуру.

Додатково під час реалізації аналітичного модуля було враховано проблему неповноти транспортних даних для окремих дорожніх сегментів. Для уникнення появи порожніх або некоректно визначених ділянок на карті у системі реалізовано механізм резервного прогнозування, який використовує історичні статистичні показники для суміжних часових інтервалів або еталонні значення швидкості вільного транспортного потоку. Такий підхід забезпечує цілісність прогнозної моделі та підвищує достовірність відображення транспортної ситуації.

Для детальнішого відображення логіки функціонування алгоритму прогнозування було побудовано діаграму активності, наведену на рис. 3.6. Вона демонструє послідовність виконання операцій під час аналізу транспортних даних, перевірки коректності координат, визначення дорожнього сегмента, обчислення середньої швидкості руху та формування прогнозного рівня завантаженості дорожньої мережі.

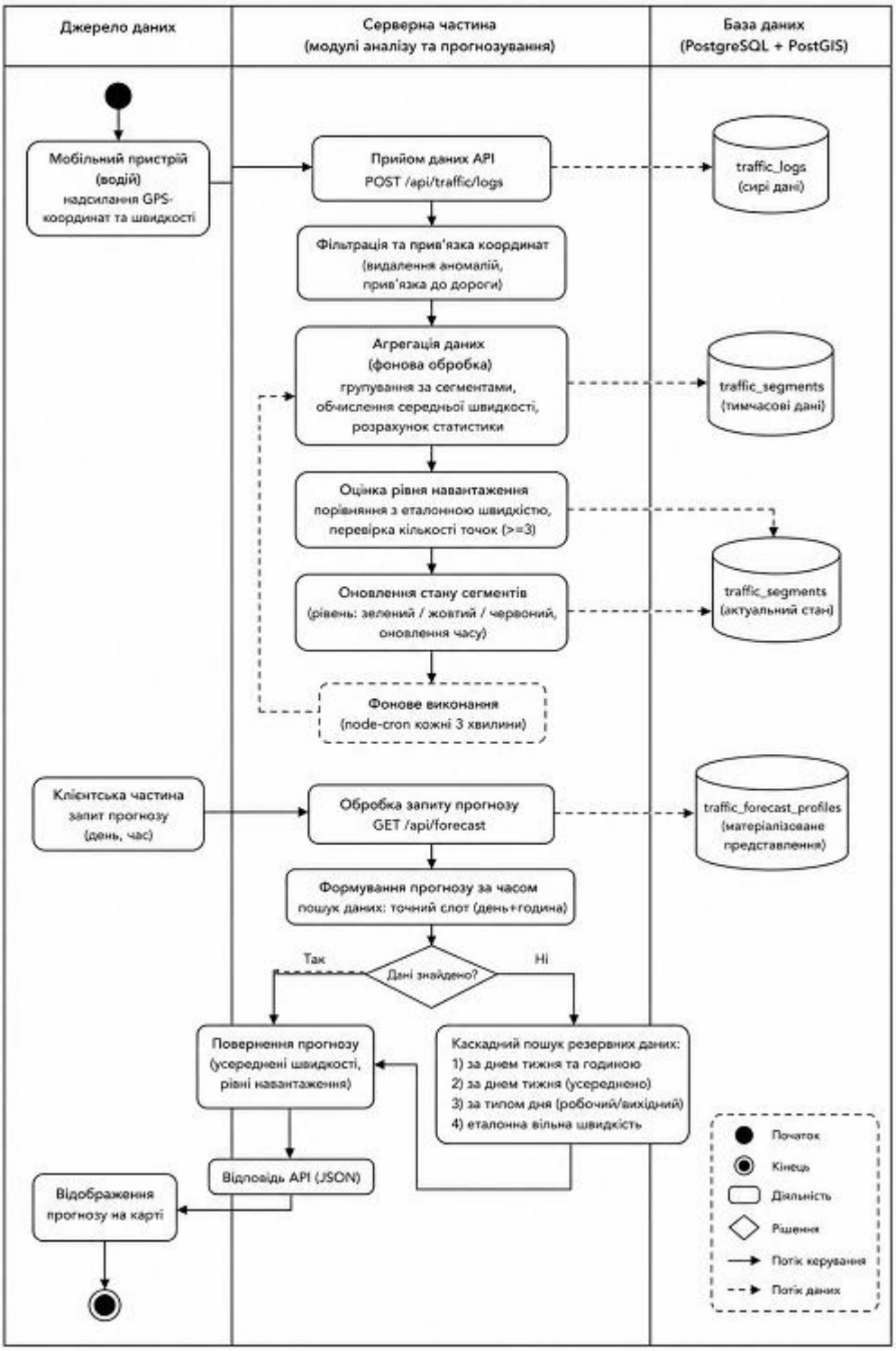

Рис. 3.6 Діаграма активності роботи модулів аналізу та прогнозування транспортного трафіку

Подана діаграма активності дозволяє простежити внутрішню логіку функціонування аналітичного модуля системи, включаючи перевірку наявності історичних даних, механізми резервного прогнозування та передачу

сформованих результатів до клієнтської частини застосунку. Додатково діаграма демонструє використання фонової агрегації та оновлення матеріалізованих представлень для підтримки актуальності прогнозних показників.

Реалізовані модулі аналізу та прогнозування забезпечують ефективну обробку координатних даних, фільтрацію GPS-похибок, оцінювання рівня транспортного навантаження та формування прогнозів дорожнього руху на основі статистичної агрегації історичних даних. Використання геопросторових функцій PostGIS, фонової агрегації та матеріалізованих представлень дозволило забезпечити високу швидкодію системи та стабільну роботу програмного застосунку в режимі реального часу.

3.4 Візуалізація результатів роботи програмної системи

Одним із ключових етапів реалізації програмної системи SmartTraffic AI стало створення засобів візуалізації транспортної інформації та результатів прогнозування. Оскільки транспортні дані мають значний обсяг і складну просторово-часову структуру, ефективність роботи системи значною мірою залежить від способу їх графічного представлення користувачу. Основним завданням модуля візуалізації стало забезпечення швидкого сприйняття поточної дорожньої ситуації, відображення рівня транспортного навантаження та представлення прогнозних показників у зрозумілому інтерактивному вигляді.

Основою інтерфейсу програмної системи є інтерактивна карта, реалізована із використанням бібліотеки Mapbox GL JS. Даний інструмент забезпечує підтримку векторної графіки, апаратного прискорення через WebGL та динамічного оновлення географічних об’єктів у режимі реального часу. Для передачі просторових даних між серверною та клієнтською частинами використовується формат GeoJSON, який забезпечує сумісність із картографічними сервісами та спрощує обробку координатних даних у браузері користувача.

Під час реалізації модуля візуалізації було застосовано механізм Data-Driven Styling, який дозволяє динамічно змінювати стиль відображення дорожніх сегментів залежно від показників транспортного навантаження. Серверна частина передає до клієнтського застосунку числовий індекс трафіку, після чого Mapbox GL JS автоматично формує плавний градієнт кольорів від зеленого до червоного залежно від рівня завантаженості дороги. Такий підхід дозволив уникнути жорсткого закріплення кольорових статусів на серверному рівні та забезпечив більш плавну й інформативну візуалізацію дорожньої ситуації.

Для відображення статистичних та прогнозних показників у системі використано бібліотеку Recharts, яка працює на базі SVG-графіки та інтегрується із компонентною структурою React. Використання SVG-візуалізації дозволило забезпечити високу якість графічного відображення, адаптивність елементів інтерфейсу та коректну стилізацію графіків відповідно до загального дизайну застосунку.

На рис. 3.7 наведено приклад візуалізації прогнозованого рівня транспортного навантаження на інтерактивній карті.

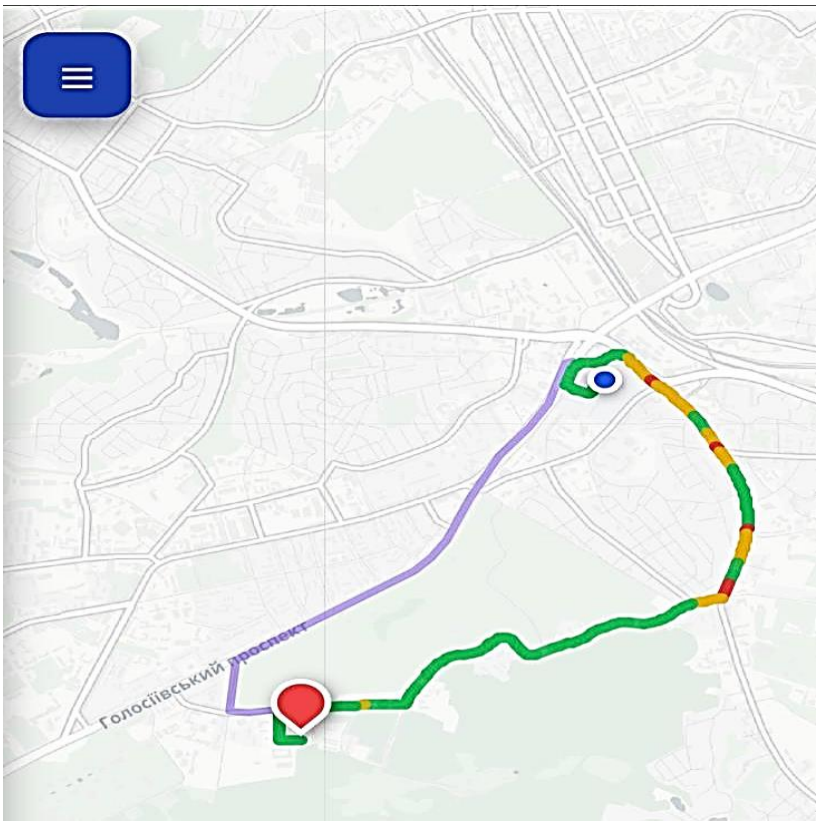

Рис. 3.7 Візуалізація транспортного трафіку на інтерактивній карті

На рис. 3.7 представлено приклад візуалізації транспортного маршруту із динамічним відображенням рівня дорожнього навантаження. Кольорова індикація дорожніх сегментів формується автоматично на основі прогнозованих показників транспортного трафіку та дозволяє швидко оцінити стан дорожнього руху на окремих ділянках маршруту.

Під час побудови графіків прогнозування було враховано проблему синхронізації часових поясів між серверною та клієнтською частинами системи. Для коректного відображення часових інтервалів використано бібліотеку date-fns та механізми автоматичного визначення локального часового поясу пристрою користувача.

Особливу увагу приділено адаптивності інтерфейсу для мобільних пристроїв. Для уникнення перевантаження інтерфейсу на невеликих екранах реалізовано механізм динамічного масштабування графіків та адаптивного відображення підписів осей залежно від ширини екрана.

Основні засоби візуалізації, реалізовані у програмній системі SmartTraffic AI, наведено у табл. 3.5.

Таблиця 3.5

Засоби візуалізації транспортних даних у системі SmartTraffic AI

Елемент візуалізації	Технологія	Призначення
Інтерактивна карта	Mapbox GL JS	Відображення дорожньої мережі та маршрутів
Графіки прогнозування	Recharts	Візуалізація статистичних і прогнозних показників
Геопросторові дані	GeoJSON	Передача координатної інформації
Адаптивний інтерфейс	Tailwind CSS	Підтримка мобільних пристроїв
Форматування часу	date-fns	Синхронізація часових поясів

Для підвищення зручності роботи користувача таблиці статистичних показників реалізовано із підтримкою горизонтального прокручування та адаптивного масштабування елементів інтерфейсу. Це дозволило уникнути

порушення структури сторінки під час відображення великого обсягу аналітичної інформації на мобільних пристроях.

На рис. 3.8 наведено приклад візуалізації прогнозних статистичних показників транспортного трафіку.

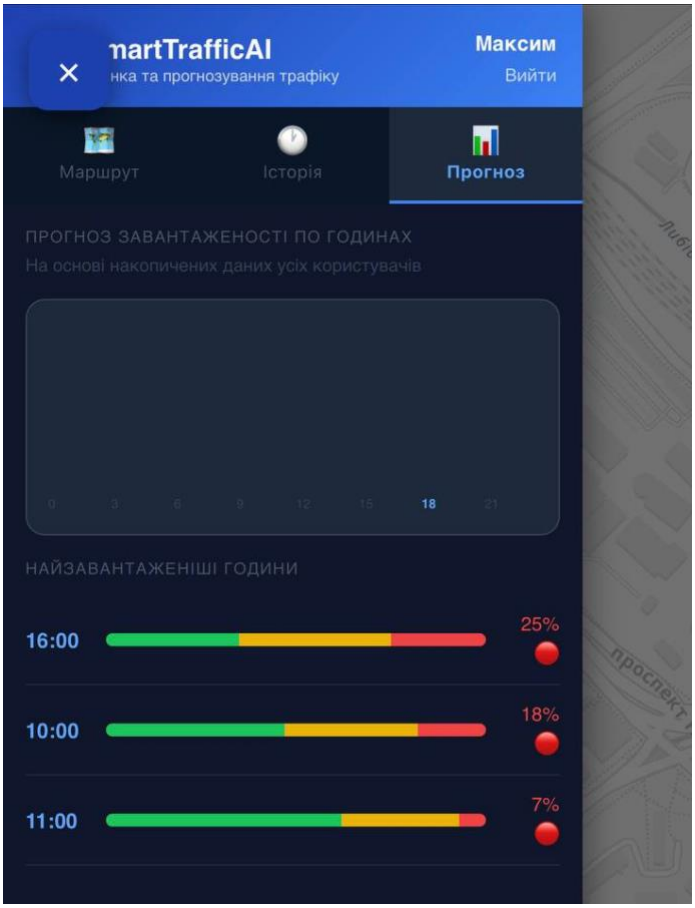

Рис. 3.8 Візуалізація прогнозних статистичних даних транспортного трафіку

На рис. 3.8 представлено інтерфейс модуля прогнозування транспортного навантаження, який відображає погодинну статистику рівня трафіку та найбільш завантажені часові інтервали. Використання кольорової індикації дозволяє швидко оцінити стан дорожнього руху та спрощує сприйняття аналітичної інформації користувачем.

Реалізовані засоби візуалізації забезпечують інтерактивне відображення транспортної інформації, підтримують адаптивну взаємодію із користувачем та дозволяють оперативно оцінювати поточний і прогнозований стан дорожнього руху. Інтеграція сучасних вебтехнологій та картографічних сервісів забезпечила створення зручного інструменту моніторингу транспортного трафіку.

4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОЇ СИСТЕМИ

4.1 Організація та результати тестування програмної системи

Після завершення основних етапів програмної реалізації системи SmartTraffic AI було проведено комплексне тестування функціональних можливостей, стабільності роботи серверної частини та коректності взаємодії між клієнтськими й серверними компонентами. Основною метою тестування стала перевірка працездатності програмного забезпечення в умовах, наближених до реальної експлуатації, а також оцінювання правильності роботи модулів аналізу транспортного трафіку, прогнозування та побудови маршрутів.

Тестування виконувалося поетапно із використанням сценаріїв, що моделюють типовий цикл взаємодії користувача із системою. Перевірка охоплювала функціонування інтерфейсу користувача, коректність REST API-запитів, роботу механізмів авторизації, обробку геопросторових даних та стабільність відображення транспортної інформації на інтерактивній карті. Окрему увагу було приділено тестуванню адаптивності інтерфейсу на мобільних пристроях, перевірці швидкодії під час побудови маршрутів та коректності відображення прогнозних показників транспортного навантаження.

Під час першого запуску вебзастосунку система автоматично ініціалізує картографічний інтерфейс та виконує запит доступу до поточної геолокації користувача через браузерний Geolocation API. Після підтвердження доступу карта фокусується на поточних координатах пристрою, що підтверджує коректність інтеграції із геолокаційними сервісами браузера. Отримані координати використовуються для автоматичного визначення стартової точки маршруту та подальшого аналізу транспортної ситуації поблизу користувача.

На рис. 4.1 наведено початковий етап роботи системи після отримання координат користувача.

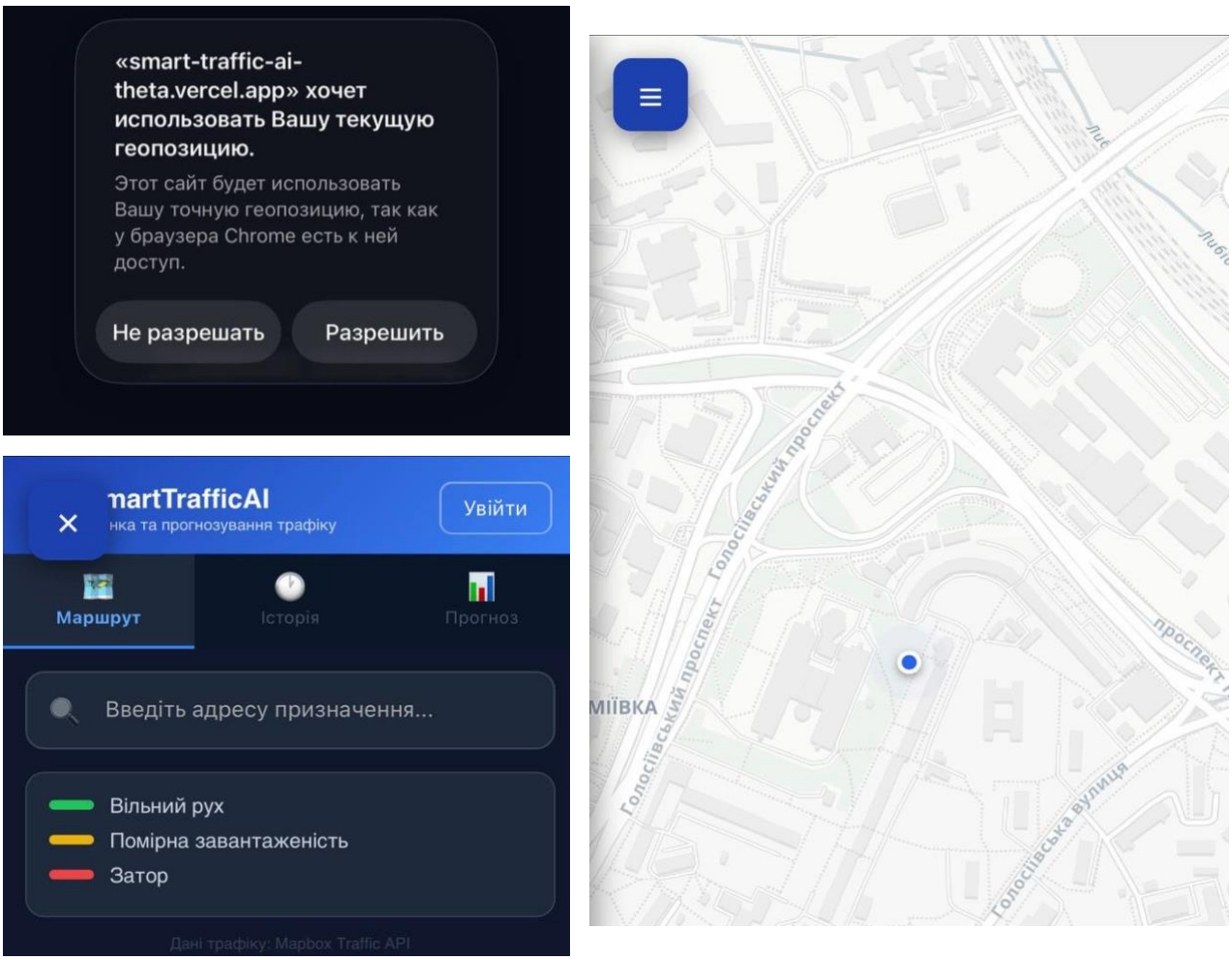

Рис. 4.1 Ініціалізація картографічного інтерфейсу та визначення геолокації користувача

Подані скріншоти демонструють процес надання дозволу на використання геолокації та результат успішного визначення поточного місцезнаходження користувача на інтерактивній карті. Коректна робота даного механізму є важливою умовою функціонування модулів побудови маршрутів і прогнозування транспортного трафіку.

Наступним етапом тестування стала перевірка роботи механізмів реєстрації та авторизації користувачів. Для цього було протестовано коректність роботи форм введення даних, перевірку формату електронної пошти, мінімальної довжини пароля та процедуру генерації JWT-токенів після успішної авторизації. Під час реєстрації серверна частина створює новий запис у таблиці

users бази даних PostgreSQL та автоматично формує токен доступу для подальшої взаємодії із системою.

Результат тестування механізмів авторизації наведено на рис. 4.2.

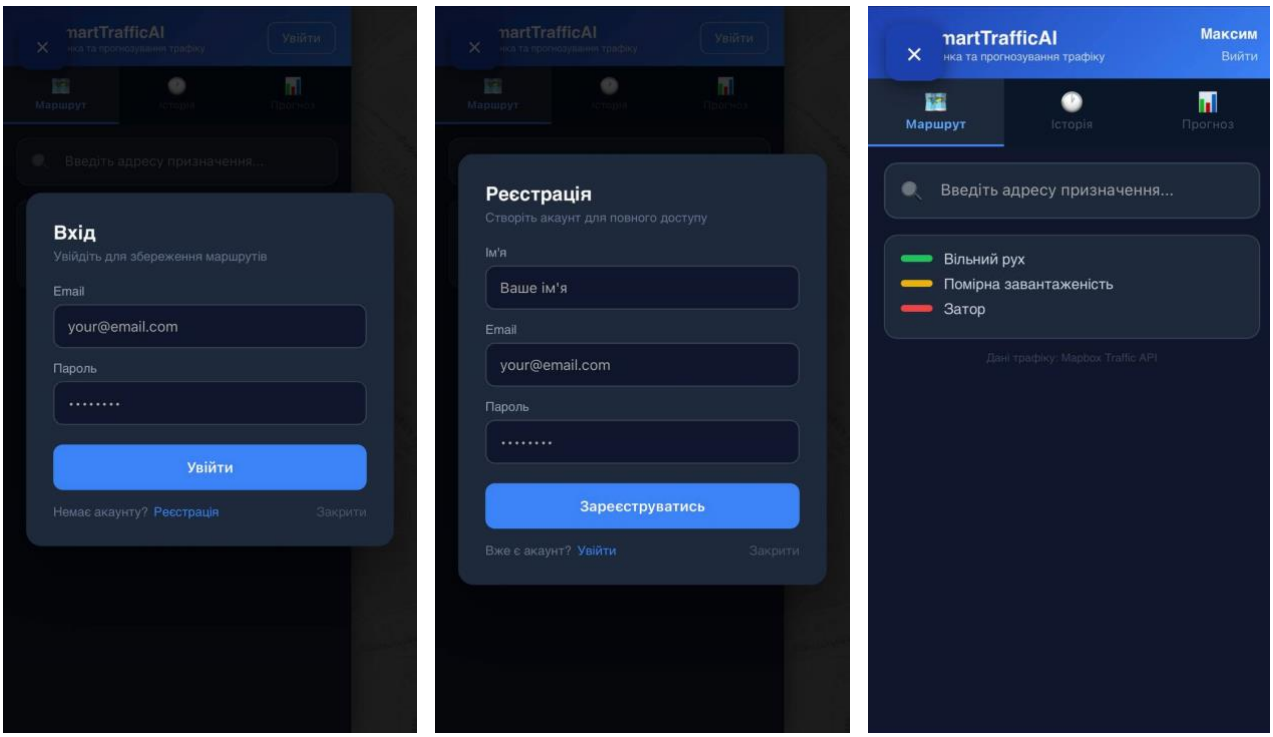

Рис. 4.2 Робота модуля авторизації та реєстрації користувачів

Одним із ключових етапів перевірки працездатності системи стало тестування модуля прогнозування транспортного навантаження. Після переходу до вкладки прогнозування система виконує аналіз історичних транспортних даних та відображає погодинні показники завантаженості дорожньої мережі. Під час тестування перевірялася правильність відображення часових інтервалів, коректність роботи графіків та відповідність кольорової індикації фактичному рівню транспортного навантаження. Додатково оцінювалася стабільність оновлення прогнозних даних під час зміни часових інтервалів та швидкість завантаження статистичної інформації на мобільних пристроях.

Окрему увагу було приділено тестуванню модуля побудови маршрутів. Після введення користувачем адреси призначення серверна частина виконує обробку координат, формує запит до Mapbox API та будує оптимальний маршрут із врахуванням поточного транспортного навантаження. Під час тестування

перевірялася коректність пошуку адрес, точність визначення координат пункту призначення, стабільність побудови маршруту та правильність відображення дорожніх сегментів із різним рівнем транспортного навантаження.

Отриманий маршрут відображається на інтерактивній карті у вигляді кольорових сегментів, що характеризують рівень трафіку на окремих дорожніх ділянках. Зелений колір відповідає вільному руху транспорту, жовтий відображає помірне навантаження, а червоний сигналізує про високий рівень заторів. Такий підхід дозволяє користувачу швидко оцінити транспортну ситуацію та обрати найбільш оптимальний маршрут пересування.

Приклад результату побудови маршруту наведено на рис. 4.3.

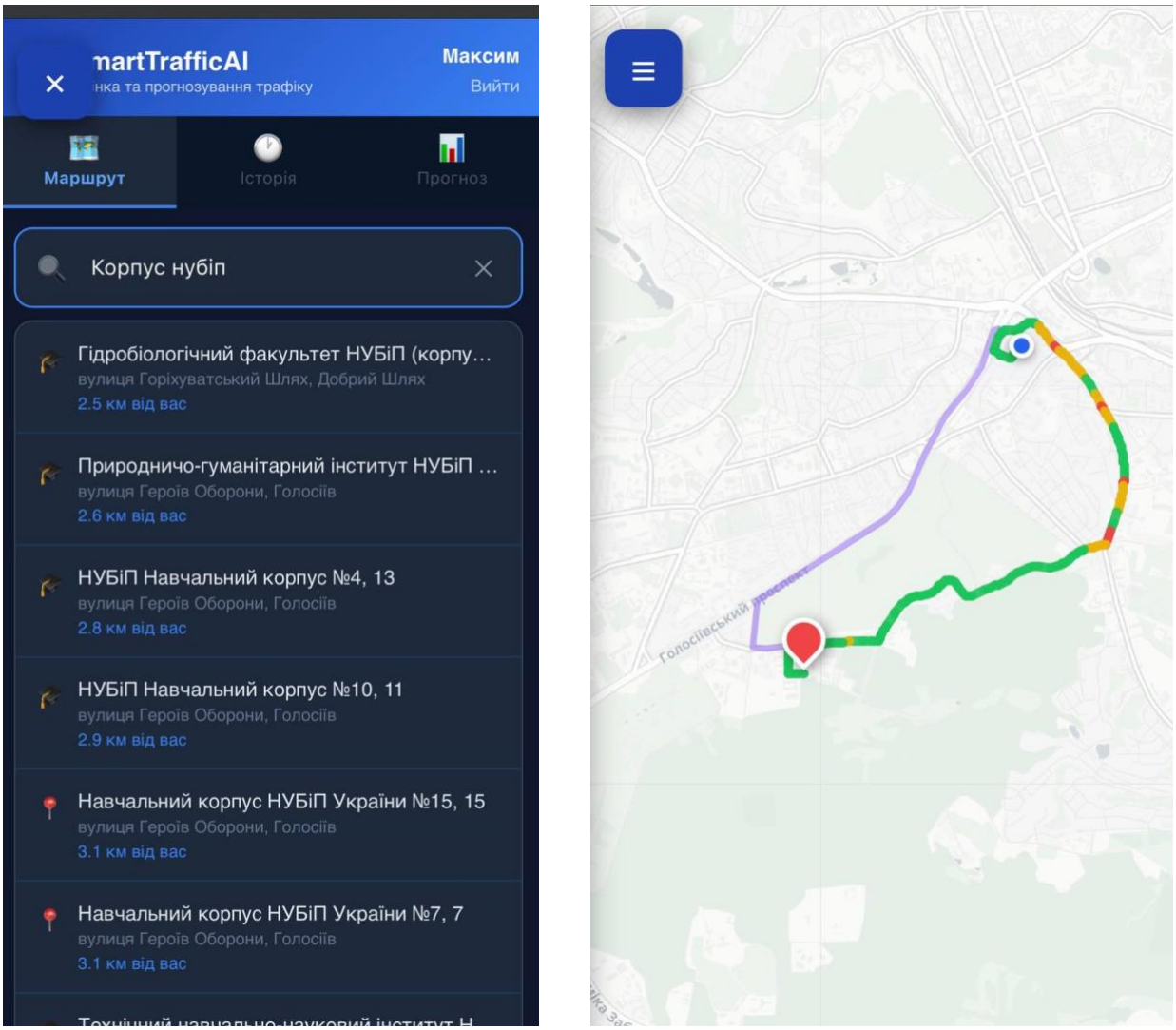

Рис. 4.3 Результат побудови маршруту із врахуванням транспортного трафіку

Подані скріншоти демонструють процес пошуку пункту призначення та результат побудови маршруту на інтерактивній карті із візуалізацією рівня транспортного навантаження на окремих дорожніх сегментах.

Під час функціонального тестування також було перевірено коректність роботи серверних ендпоінтів, механізмів обмеження CORS-запитів, автоматичне створення таблиць бази даних та стабільність роботи системи після оновлення програмного коду через CI/CD-інтеграцію GitHub, Railway та Vercel.

Результати основних функціональних випробувань наведено у табл. 4.1.

Таблиця 4.1

Результати функціонального тестування програмної системи

ID	Опис тестування	Очікуваний результат	Результат
1	Ініціалізація бази даних	Автоматичне створення таблиць системи	Успішно
2	Реєстрація користувача	Створення нового облікового запису	Успішно
3	Авторизація користувача	Генерація JWT-токена	Успішно
4	Побудова маршруту	Відображення маршруту на карті	Успішно
5	Прогнозування трафіку	Формування погодинного прогнозу	Успішно
6	Робота CORS-захисту	Блокування неавторизованих запитів	Успішно
7	Автоматичне розгортання	Оновлення застосунку після Git push	Успішно

Під час тестування продуктивності було встановлено, що середній час відповіді серверної частини під час побудови маршруту та формування прогнозу становить приблизно 150–300 мс, що є прийнятним показником для веборієнтованих геоінформаційних систем. Використання кешування, фонової агрегації транспортних даних та матеріалізованих представлень PostgreSQL дозволило зменшити навантаження на серверну інфраструктуру та забезпечити стабільну роботу програмної системи навіть під час одночасної обробки декількох запитів.

Додатково було проведено тестування адаптивності інтерфейсу на мобільних пристроях із різними розмірами екранів. Результати перевірки підтвердили коректне масштабування графічних елементів, стабільне

відображення інтерактивної карти та працездатність навігаційного інтерфейсу у мобільному режимі.

Проведене тестування підтвердило працездатність програмної системи SmartTraffic AI, коректність функціонування модулів аналізу транспортного трафіку, побудови маршрутів та прогнозування дорожнього навантаження. Отримані результати засвідчили стабільність роботи серверної та клієнтської частин застосунку, а також відповідність реалізованої системи поставленим функціональним вимогам.

4.2 Розгортання та технічне забезпечення програмної системи

Програмна система SmartTraffic AI реалізована за клієнт-серверною архітектурою із використанням хмарних сервісів для розгортання серверної частини, бази даних та клієнтського застосунку. Такий підхід дозволив забезпечити стабільний доступ до системи через мережу Інтернет, автоматизувати процес оновлення програмного забезпечення та спростити адміністрування інфраструктури. Під час вибору середовища розгортання враховувалися вимоги до швидкодії, підтримки PostgreSQL/PostGIS, інтеграції із GitHub та можливості автоматичного деплою після оновлення програмного коду.

Для реалізації серверної частини було використано платформу Railway, яка забезпечує підтримку Node.js-застосунків, автоматичне створення контейнерів та інтеграцію із PostgreSQL. Клієнтська частина системи розгорнута на платформі Vercel, що дозволило оптимізувати доставку статичних файлів через глобальну CDN-мережу та забезпечити швидке завантаження інтерфейсу користувача. Використання хмарних сервісів дозволило відмовитися від ручного налаштування фізичних серверів та суттєво спростило процес підтримки програмної системи.

Оскільки клієнтська частина системи реалізована як Single Page Application, основне навантаження припадає на сервер обробки даних та базу даних PostgreSQL. Водночас вимоги до апаратного забезпечення користувача

залишаються мінімальними, що дозволяє використовувати систему як на персональних комп’ютерах, так і на мобільних пристроях. Для коректної роботи достатньо сучасного веббраузера із підтримкою JavaScript, WebGL та стабільного підключення до мережі Інтернет.

Додатково під час розгортання було реалізовано механізми автоматичного оновлення застосунку через CI/CD-інтеграцію із GitHub. Після внесення змін до репозиторію системи платформи Railway та Vercel автоматично виконують повторну збірку проєкту та публікацію оновленої версії застосунку. Це дозволило забезпечити безперервність процесу розробки, швидке виправлення помилок та стабільне оновлення функціональних можливостей системи.

Основні вимоги до програмного та апаратного забезпечення наведено у табл. 4.2.

Таблиця 4.2

Вимоги до програмного та апаратного забезпечення

Компонент	Мінімальні вимоги
Клієнтський пристрій	2 ГБ RAM, доступ до Інтернету
Веббраузер	Google Chrome 90+, Firefox 85+, Safari 14+
Серверна платформа	Node.js v18+
Система керування БД	PostgreSQL 14+ із PostGIS
Середовище розгортання	Railway, Vercel
Інструменти розробки	Visual Studio Code, GitHub

Для коректного відображення інтерактивної карти браузер користувача повинен підтримувати WebGL та сучасні JavaScript API. Під час тестування було підтверджено коректну роботу системи на настільних комп’ютерах, планшетах та мобільних пристроях із різними розмірами екранів.

Розгортання серверної частини виконувалося із використанням платформи Railway, яка забезпечує підтримку Node.js-застосунків та автоматичне створення PostgreSQL-кластера. Серверна частина системи підключена до GitHub-репозиторію, що дозволяє автоматично виконувати оновлення застосунку після внесення змін до програмного коду. Під час розгортання було налаштовано

змінні середовища для безпечного зберігання токенів доступу та конфігураційних параметрів системи.

Для зберігання транспортних даних та просторової інформації використовується PostgreSQL із розширенням PostGIS. Сервер бази даних працює окремо від клієнтської частини та взаємодіє із Node.js-застосунком через захищене TCP-з'єднання. Для оптимізації роботи із координатними даними використано просторові індекси та параметризовані SQL-запити.

Клієнтська частина системи розгорнута на платформі Vercel, яка автоматично виконує збірку React-застосунку та публікацію статичних файлів через глобальну CDN-мережу. Такий підхід забезпечує швидке завантаження інтерфейсу користувача та стабільний доступ до системи незалежно від місця підключення користувача.

На рис. 4.4 наведено діаграму розгортання програмної системи SmartTraffic AI.

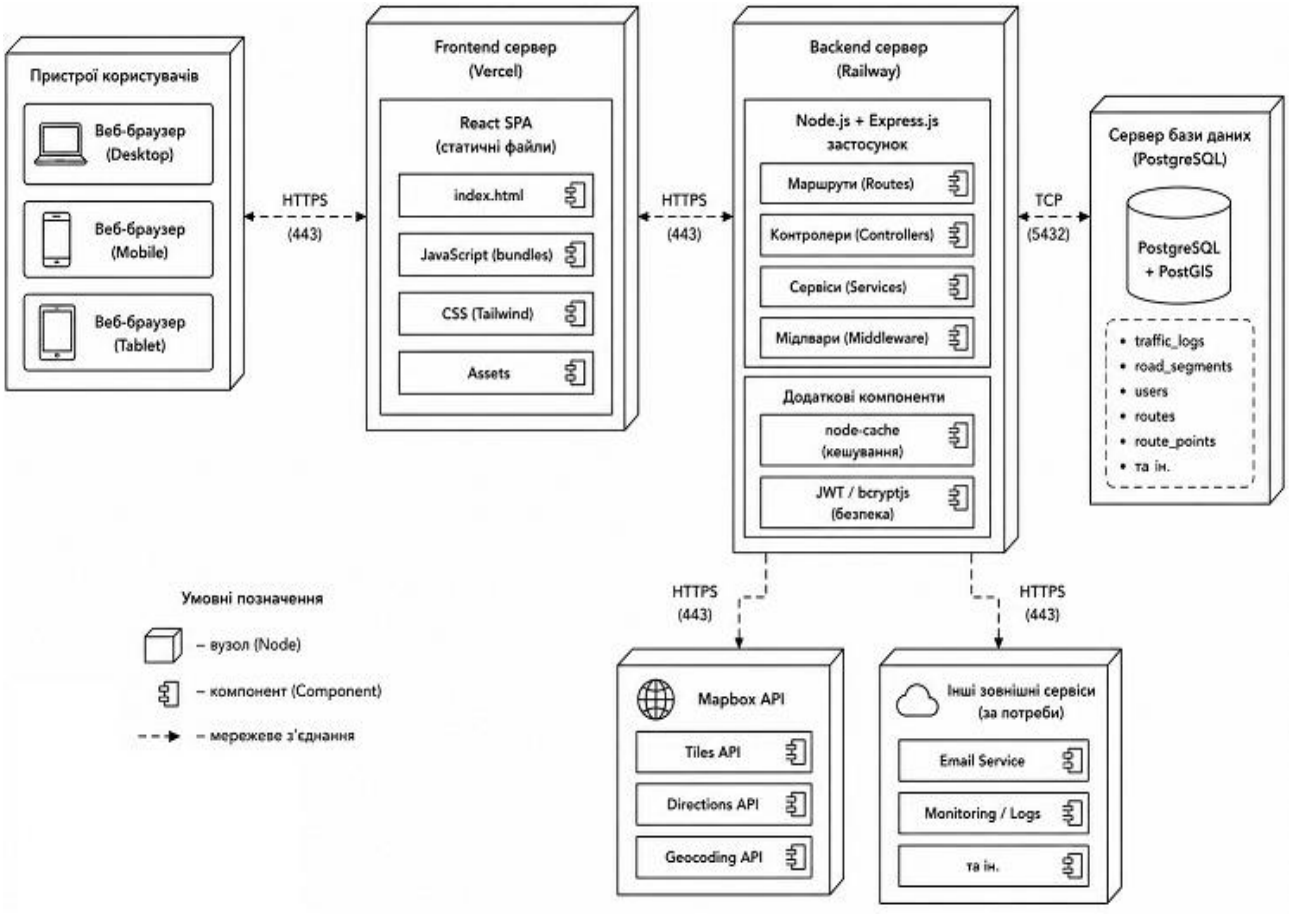

Рис. 4.4 Діаграма розгортання системи SmartTraffic AI

Подана діаграма (рис. 4.4) відображає структуру взаємодії між клієнтськими пристроями, frontend- та backend-серверами, сервером бази даних PostgreSQL, а також зовнішніми API-сервісами. Діаграма демонструє використання захищених HTTPS-з'єднань, організацію обробки запитів у Node.js-застосунку та взаємодію із картографічними сервісами Mapbox API.

У процесі розгортання також було реалізовано автоматичне оновлення програмного забезпечення через механізми CI/CD. Після внесення змін до GitHub-репозиторію платформи Railway та Vercel автоматично виконують повторну збірку та публікацію оновленої версії системи. Це дозволило значно спростити процес підтримки застосунку та забезпечити швидке внесення змін без необхідності ручного налаштування серверної інфраструктури.

Проведене тестування підтвердило стабільність роботи програмної системи у хмарному середовищі, коректність взаємодії між усіма компонентами архітектури та відповідність технічної інфраструктури вимогам веборієнтованих геоінформаційних систем реального часу.

4.3 Інструкція користувача та демонстрація роботи системи

Програмна система SmartTraffic AI орієнтована на використання через веббраузер без необхідності встановлення додаткового програмного забезпечення на пристрій користувача. Доступ до функціональних можливостей системи здійснюється через клієнтський вебінтерфейс, розгорнутий на платформі Vercel. Для початку роботи користувачу необхідно перейти за вебадресою застосунку та надати доступ до поточної геолокації пристрою. Отримання координат використовується для автоматичного визначення стартової точки маршруту та подальшого аналізу транспортної ситуації.

Після відкриття застосунку користувач отримує доступ до інтерактивної карти, яка відображає поточне місцезнаходження та дорожню мережу. Навігація по системі реалізована через адаптивне меню, яке забезпечує швидкий перехід між режимами побудови маршруту, перегляду прогнозів та історії поїздок. Для

доступу до персоналізованих функцій передбачено механізм авторизації та реєстрації користувачів із використанням JWT-аутентифікації.

Для побудови маршруту користувач вводить адресу пункту призначення у рядок пошуку або обирає необхідну точку безпосередньо на інтерактивній карті. Після обробки координат серверна частина виконує запит до Mapbox API та формує оптимальний маршрут із врахуванням поточного транспортного навантаження. Побудований маршрут автоматично відображається на карті у вигляді кольорових сегментів, які характеризують рівень транспортного трафіку на окремих ділянках дороги.

Окремий режим роботи системи призначений для перегляду прогнозних статистичних показників транспортного трафіку. Після переходу до вкладки прогнозування система виконує аналіз історичних даних та відображає погодинний прогноз рівня завантаженості дорожньої мережі. Візуалізація результатів реалізована із використанням кольорової індикації та графічних елементів, що дозволяє швидко оцінити прогнозований стан дорожнього руху.

Основні функціональні можливості програмної системи наведено у табл. 4.3.

Таблиця 4.3

Основні функціональні можливості системи SmartTraffic AI

Функціональна можливість	Призначення
Авторизація користувачів	Доступ до персоналізованих функцій системи
Геолокація	Визначення поточного місцезнаходження
Побудова маршруту	Формування оптимального маршруту руху
Аналіз трафіку	Оцінювання рівня транспортного навантаження
Прогнозування	Відображення погодинного прогнозу трафіку
Історія маршрутів	Збереження інформації про поїздки користувача

Наведені у табл. 4.3 функціональні можливості охоплюють основні сценарії взаємодії користувача із програмною системою SmartTraffic AI. Реалізований набір функцій забезпечує не лише побудову маршрутів та відображення транспортної ситуації, але й підтримує аналіз історичних даних,

прогнозування рівня навантаження та персоналізацію роботи користувача через механізми авторизації та збереження маршрутів. Поєднання геоінформаційних сервісів, модулів прогнозування та інтерактивної візуалізації дозволило створити комплексну систему моніторингу транспортного трафіку.

Під час роботи із системою результати побудови маршрутів та інформація про транспортне навантаження оновлюються автоматично без необхідності перезавантаження сторінки. Це забезпечується використанням SPA-архітектури та асинхронної взаємодії клієнтської частини із сервером через REST API. Такий підхід дозволяє мінімізувати затримки під час отримання нових транспортних даних та забезпечує плавну взаємодію користувача із картографічним інтерфейсом.

Додатковою перевагою реалізованого підходу є можливість оперативного оновлення інформації про дорожню ситуацію під час зміни маршруту або переміщення користувача. Після отримання нових координат серверна частина автоматично виконує повторний аналіз транспортного навантаження та за потреби формує оновлений маршрут із врахуванням актуального стану дорожньої мережі. Це дозволяє системі адаптуватися до динамічних змін транспортної ситуації у режимі реального часу.

Реалізований вебінтерфейс забезпечує зручну взаємодію користувача із системою та підтримує стабільну роботу як на персональних комп’ютерах, так і на мобільних пристроях. Під час тестування підтверджено коректність відображення інтерактивної карти, графічних елементів прогнозування та адаптивного інтерфейсу на різних типах екранів. Проведена демонстрація роботи підтвердила коректність функціонування модулів побудови маршрутів, аналізу транспортного трафіку та прогнозування дорожнього навантаження.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено програмну систему SmartTraffic AI, призначену для аналізу, моніторингу та прогнозування міського транспортного трафіку із використанням сучасних вебтехнологій, геоінформаційних сервісів та інструментів обробки просторових даних. Реалізована система забезпечує побудову маршрутів, оцінювання рівня транспортного навантаження та візуалізацію прогнозних показників дорожнього руху в режимі реального часу.

Під час виконання роботи було проведено аналіз сучасних підходів до побудови інтелектуальних транспортних систем, геоінформаційних сервісів та методів прогнозування транспортного навантаження. За результатами аналізу обрано клієнт-серверну архітектуру із розподілом функцій між frontend- та backend-компонентами системи.

Для реалізації серверної частини використано Node.js та Express, а клієнтський вебінтерфейс розроблено на базі React. Для роботи із просторовими даними застосовано PostgreSQL із розширенням PostGIS, що забезпечило підтримку географічних типів даних, просторової індексації та виконання координатних обчислень. Це дозволило підвищити продуктивність системи під час роботи із великими обсягами транспортних даних.

У процесі розробки сформовано логічну та фізичну структуру бази даних для зберігання інформації про користувачів, маршрути та показники транспортного трафіку. Для оптимізації швидкодії використано просторові індекси GiST, матеріалізовані представлення та механізми кешування даних.

Під час реалізації модулів аналізу та прогнозування розроблено алгоритми агрегації транспортних даних, оцінювання рівня дорожнього навантаження та формування прогнозів на основі статистичної обробки історичних записів. Для зменшення навантаження на сервер впроваджено механізми фонової обробки даних із використанням Cron-задач та попередньої агрегації результатів.

Значну увагу приділено реалізації інтерактивного користувацького інтерфейсу та візуалізації транспортної інформації. Для відображення дорожнього навантаження використано інтеграцію із Mapbox GL JS, а для побудови прогнозних графіків – бібліотеку Recharts. Реалізовано адаптивний інтерфейс, який забезпечує коректну роботу системи як на персональних комп’ютерах, так і на мобільних пристроях.

Проведене тестування підтвердило працездатність програмної системи, коректність функціонування модулів авторизації, побудови маршрутів, аналізу транспортного трафіку та прогнозування дорожнього навантаження. Результати тестування показали стабільну роботу клієнтської та серверної частин системи, а також прийнятний час відповіді під час обробки транспортних даних і побудови маршрутів.

Розгортання системи виконано із використанням хмарних платформ Railway та Vercel, що дозволило реалізувати автоматичне оновлення програмного забезпечення через CI/CD-інтеграцію із GitHub та забезпечити стабільний доступ до системи через мережу Інтернет.

Узагальнюючи результати виконаної роботи, можна зробити такі висновки:

- розроблено веборієнтовану систему аналізу та прогнозування міського транспортного трафіку;
- реалізовано клієнт-серверну архітектуру на базі React, Node.js та PostgreSQL/PostGIS;
- забезпечено підтримку обробки й аналізу геопросторових даних;
- реалізовано модулі побудови маршрутів та прогнозування транспортного навантаження;
- впроваджено механізми кешування, фонової агрегації та оптимізації SQL-запитів;
- створено адаптивний інтерфейс із картографічною візуалізацією;
- підтверджено працездатність системи шляхом функціонального тестування та практичної експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алексєєв О. П. Проєктування інформаційних систем : навч. посіб. Київ : Кондор, 2021. 336 с.
2. Биков В. Ю., Спірін О. М. Інформаційні технології та цифрова трансформація суспільства : монографія. Київ : ФОП Ямчинський О. В., 2022. 248 с.
3. Глушаков С. В., Євсєєв Є. А. Web-програмування : навч. курс. Харків : Фоліо, 2021. 512 с.
4. Коротєєва Т. О. Алгоритми та структури даних : навч. посіб. Львів : Видавництво Львівської політехніки, 2021. 304 с.
5. Мельник І. В., Ткаченко С. П. Аналіз сучасних систем візуалізації даних у вебсередовищі // Вісник Хмельницького національного університету. Технічні науки. 2022. № 6. С. 214–220.
6. Ніколенко А. В. Проєктування клієнт-серверних вебзастосунків : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2023. 256 с.
7. Learning React / Alex Banks, Eve Porcello. Sebastopol : O’Reilly Media, 2023. 310 p.
8. Node.js Design Patterns / Mario Casciaro, Luciano Mammino. Birmingham : Packt Publishing, 2022. 824 p.
9. PostGIS in Action / Regina Obe, Leo Hsu. Shelter Island : Manning Publications, 2021. 600 p.
10. Software Architecture with Node.js / Thomas Hunter II, Amy Rondeau. Birmingham : Packt Publishing, 2022. 406 p.
11. Sharma R., Gupta P. Modern Approaches to Full-Stack Web Development Using React and Node.js // International Journal of Advanced Computer Science and Applications. 2023. Vol. 14, No. 7. P. 455–462.
12. Wilson G., Brown T. Secure Authentication Mechanisms in Modern Web Applications //Journal of Cybersecurity and Privacy. 2024. Vol. 4, No. 1. P. 67–81.
13. Zhang Y., Li X., Wang H. Intelligent Web-Based Traffic Monitoring Systems Using GIS Technologies // IEEE Access. 2022. Vol. 10. P. 88421–88435.

14. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс]. URL: PostgreSQL Documentation
15. React – The library for web and native user interfaces [Електронний ресурс]. URL: React Documentation
16. Node.js – Run JavaScript Everywhere [Електронний ресурс]. URL: Node.js Documentation
17. Express – Fast, unopinionated, minimalist web framework for Node.js [Електронний ресурс]. URL: Express.js Documentation
18. TypeScript: JavaScript With Syntax for Types [Електронний ресурс]. URL: TypeScript Documentation
19. Tailwind CSS – Rapidly build modern websites without ever leaving your HTML [Електронний ресурс]. URL: Tailwind CSS Documentation
20. Redux Toolkit – The official, opinionated, batteries-included toolset for efficient Redux development [Електронний ресурс]. URL: Redux Toolkit Documentation
21. Docker Documentation [Електронний ресурс]. URL: Docker Documentation
22. Vite Documentation [Електронний ресурс]. URL: Vite Documentation
23. Postman API Platform [Електронний ресурс]. URL: Postman API Platform
24. JSON Web Tokens [Електронний ресурс]. URL: JWT Documentation
25. Mapbox GL JS Documentation [Електронний ресурс]. URL: Mapbox GL JS Documentation
26. OpenStreetMap Wiki [Електронний ресурс]. URL: OpenStreetMap Wiki
27. REST API Tutorial – REST Resource Naming Guide [Електронний ресурс]. URL: REST API Tutorial
28. MDN Web Docs – JavaScript Guide [Електронний ресурс]. URL: MDN JavaScript Guide
29. MDN Web Docs – CSS: Cascading Style Sheets [Електронний ресурс]. URL: MDN CSS Guide
30. MDN Web Docs – HTTP Overview [Електронний ресурс]. URL: MDN HTTP Overview

ДОДАТОК А

Програмний код застосунку

Лістинг index.html

```
<!DOCTYPE html>
<html lang="uk">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>SmartTrafficAI</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Лістинг App.js

```
import MapView from "./MapView";

function App() {
  return <MapView />;
}

export default App;
```

Лістинг MapView.jsx

```
import { MapContainer, TileLayer, Marker, Polyline, ZoomControl, useMap } from "react-leaflet";
import { useEffect, useState, useRef } from "react";
import axios from "axios";
import L from "leaflet";
import markerIcon from "leaflet/dist/images/marker-icon.png";
import markerShadow from "leaflet/dist/images/marker-shadow.png";
import "leaflet/dist/leaflet.css";

delete L.Icon.Default.prototype._getIconUrl;
L.Icon.Default.mergeOptions({ iconUrl: markerIcon, shadowUrl: markerShadow });

const userIcon = L.divIcon({
  className: "",
  html: `<div class="user-marker"><div class="user-marker-pulse"></div><div class="user-marker-
dot"></div></div>`,
  iconSize: [20, 20],
  iconAnchor: [10, 10],
});

const destIcon = L.divIcon({
  className: "",
  html: `<div style="display:flex;flex-direction:column;align-items:center;">
    <div style="width:28px;height:28px;background:#ef4444;border:3px solid white;border-
radius:50% 50% 50% 0;transform:rotate(-45deg);box-shadow:0 3px 8px rgba(0,0,0,0.4);"></div>
  </div>`,
```

```
  iconSize: [28, 34],
  iconAnchor: [14, 34],
});

const API = process.env.REACT_APP_API_URL || "http://localhost:5000";
const COLOR_MAP = { green: "#22c55e", yellow: "#eab308", red: "#ef4444" };
const ALT_COLORS = ["#8b5cf6", "#06b6d4"];
const TABS = [
  { key: "route",   label: "Маршрут", icon: "" },
  { key: "history", label: "Історія", icon: "" },
  { key: "forecast",label: "Прогноз", icon: "" },
];
const DAYS = ["Нд","Пн","Вт","Ср","Чт","Пт","Сб"];
const REFRESH_INTERVAL = 2 * 60 * 1000;

const haversine = (lat1, lon1, lat2, lon2) => {
  const R = 6371, d2r = Math.PI / 180;
  const dLat = (lat2-lat1)*d2r, dLon = (lon2-lon1)*d2r;
  const a = Math.sin(dLat/2)**2 + Math.cos(lat1*d2r)*Math.cos(lat2*d2r)*Math.sin(dLon/2)**2;
  return R * 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
};
const formatDist = (km) => km < 1 ? `${(km*1000).toFixed(0)} м` : `${km.toFixed(1)} км`;
const formatTime = (iso) => {
  const d = new Date(iso);
  return `${DAYS[d.getDay()]}, ${d.toLocaleDateString("uk-UA")} ${d.toLocaleTimeString("uk-
UA",{hour:"2-digit",minute:"2-digit"})}`;
};
const arrivalTime = (sec) => new Date(Date.now() + sec * 1000).toLocaleTimeString("uk-UA", {
hour:"2-digit", minute:"2-digit" });

const CATEGORY_ICONS = {
  restaurant:"", cafe:"", bar:"", fast_food:"",
  supermarket:"", shop:"", hospital:"", pharmacy:"",
  school:"", university:"", hotel:"", fuel:"",
  parking:"P", bank:"", park:"", subway_entrance:"",
  bus_stop:"", train_station:"", airport:"",
};
const getCategoryIcon = (item) =>
  CATEGORY_ICONS[item.type?.toLowerCase()] ||
CATEGORY_ICONS[item.class?.toLowerCase()] || "";

function MapUpdater({ position, hasRoute }) {
  const map = useMap();
  useEffect(() => { if (position && !hasRoute) map.flyTo(position, 16); }, [position, map]);
  return null;
}

function TrafficBar({ g, y, r, height = 6 }) {
  return (
    <div style={{ display:"flex", borderRadius:4, overflow:"hidden", height }}>
      {g > 0 && <div style={{ flex:g, background:"#22c55e" }} />}
      {y > 0 && <div style={{ flex:y, background:"#eab308" }} />}
```

```
      {r > 0 && <div style={{ flex:r, background:"#ef4444" }} />}
    </div>
  );
}

export default function MapView() {
  const [position, setPosition]         = useState(null);
  const [destination, setDestination]   = useState(null);
  const [routes, setRoutes]             = useState([]);
  const [selectedIdx, setSelectedIdx]   = useState(0);
  const [trafficLoading, setTrafficLoading] = useState(false);
  const [lastUpdated, setLastUpdated]   = useState(null);
  const routeParamsRef                  = useRef(null);

  const [query, setQuery]               = useState("");
  const [suggestions, setSuggestions]   = useState([]);
  const [inputFocused, setInputFocused] = useState(false);
  const [activeIndex, setActiveIndex]   = useState(-1);
  const [searchHistory, setSearchHistory] = useState(() => {
    try { return JSON.parse(localStorage.getItem("search_history") || "[]"); } catch { return []; }
  });

  const [activeTab, setActiveTab]       = useState("route");
  const [sidebarOpen, setSidebarOpen]   = useState(window.innerWidth >= 768);

  const [user, setUser]                 = useState(() => {
    try { return JSON.parse(localStorage.getItem("user") || "null"); } catch { return null; }
  });
  const [token, setToken]               = useState(() => localStorage.getItem("token") || null);
  const [authModal, setAuthModal]       = useState(null);
  const [authForm, setAuthForm]         = useState({ name: "", email: "", password: "" });
  const [authError, setAuthError]       = useState("");
  const [authLoading, setAuthLoading]   = useState(false);
  const [routeHistory, setRouteHistory] = useState([]);
  const [forecastData, setForecastData] = useState([]);

  const selected = routes[selectedIdx] || null;

  // ——— Геолокація ———
  useEffect(() => {
    const id = navigator.geolocation.watchPosition(
      (pos) => setPosition([pos.coords.latitude, pos.coords.longitude]),
      () => alert("Дозволь геолокацію"),
      { enableHighAccuracy: true, maximumAge: 2000, timeout: 10000 }
    );
    return () => navigator.geolocation.clearWatch(id);
  }, []);

  // ——— Пошук
  useEffect(() => {
    setActiveIndex(-1);
    if (!position || query.length < 2) { setSuggestions([]); return; }
```

```
    const [lat, lon] = position;
    const delta = 0.3;
    const t = setTimeout(() => {
      axios.get("https://nominatim.openstreetmap.org/search", {
        params: { q: query, format: "json", limit: 7, countrycodes: "ua",
          viewbox: `${lon-delta},${lat+delta},${lon+delta},${lat-delta}`, bounded: 0, addressdetails: 1
},
        headers: { "Accept-Language": "uk" }
      }).then(res => {
        setSuggestions(res.data.map(item => ({
          ...item, dist: haversine(lat, lon, parseFloat(item.lat), parseFloat(item.lon))
        })).sort((a,b) => a.dist - b.dist));
      }).catch(console.error);
    }, 280);
    return () => clearTimeout(t);
  }, [query, position]);

  // ——— Автооновлення
  useEffect(() => {
    if (!routeParamsRef.current) return;
    const interval = setInterval(() => fetchRoutes(routeParamsRef.current, false),
REFRESH_INTERVAL);
    return () => clearInterval(interval);
  }, [routes.length]);

  // ——— Вкладки
  useEffect(() => {
    if (activeTab === "history" && user) loadHistory();
    if (activeTab === "forecast") loadForecast();
  }, [activeTab]);

  const loadHistory = async () => {
    try {
      const res = await axios.get(`${API}/api/routes/history`, { headers: { Authorization: `Bearer
${token}` } });
      setRouteHistory(res.data);
    } catch (err) { console.error(err); }
  };

  const loadForecast = async () => {
    try {
      const res = await axios.get(`${API}/api/forecast`);
      setForecastData(res.data);
    } catch (err) { console.error(err); }
  };

  // ——— Авторизація
  const handleAuth = async (e) => {
    e.preventDefault();
    setAuthError(""); setAuthLoading(true);
    try {
      const endpoint = authModal === "login" ? "/api/auth/login" : "/api/auth/register";
```

```
      const res = await axios.post(`${API}${endpoint}`, authForm);
      localStorage.setItem("token", res.data.token);
      localStorage.setItem("user", JSON.stringify(res.data.user));
      setToken(res.data.token); setUser(res.data.user);
      setAuthModal(null); setAuthForm({ name: "", email: "", password: "" });
    } catch (err) {
      setAuthError(err.response?.data?.error || "Помилка");
    } finally { setAuthLoading(false); }
  };

  const logout = () => {
    localStorage.removeItem("token"); localStorage.removeItem("user");
    setToken(null); setUser(null);
  };

  // ——— Маршрути
  const fetchRoutes = async ({ start, end, address, tok }, showLoading = true) => {
    if (showLoading) { setTrafficLoading(true); setRoutes([]); setSelectedIdx(0); }
    try {
      const res = await axios.get(`${API}/api/route`, {
        params: { start, end, address },
        headers: tok ? { Authorization: `Bearer ${tok}` } : {}
      });
      setRoutes(res.data.routes);
      setLastUpdated(new Date().toLocaleTimeString("uk-UA", { hour: "2-digit", minute: "2-digit"
}));
      if (showLoading && window.innerWidth < 768) setSidebarOpen(false);
    } catch (err) {
      console.error(err);
      if (showLoading) alert("Помилка побудови маршруту");
    } finally {
      if (showLoading) setTrafficLoading(false);
    }
  };

  const buildRoute = async (dest, label) => {
    const start = `${position[1]},${position[0]}`;
    const end   = `${dest[1]},${dest[0]}`;
    const params = { start, end, address: label, tok: token };
    routeParamsRef.current = params;
    await fetchRoutes(params, true);
  };

  const selectSuggestion = (item) => {
    const coords = [parseFloat(item.lat), parseFloat(item.lon)];
    const label  = (item.label || item.display_name.split(",").slice(0,2).join(",")).trim();
    setDestination(coords);
    setQuery(label); setSuggestions([]); setActiveIndex(-1);
    setSearchHistory(prev => {
      const next = [{ ...item, label }, ...prev.filter(h => h.place_id !== item.place_id)].slice(0,5);
      localStorage.setItem("search_history", JSON.stringify(next));
      return next;
```

```
    });
    buildRoute(coords, label);
  };

  const handleKeyDown = (e) => {
    if (!suggestions.length) return;
    if (e.key === "ArrowDown")  { e.preventDefault(); setActiveIndex(i => Math.min(i+1,
suggestions.length-1)); }
    if (e.key === "ArrowUp")    { e.preventDefault(); setActiveIndex(i => Math.max(i-1, 0)); }
    if (e.key === "Enter" && activeIndex >= 0) selectSuggestion(suggestions[activeIndex]);
    if (e.key === "Escape")     { setSuggestions([]); setActiveIndex(-1); }
  };

  if (!position) return (
    <div style={{ height:"100vh", display:"flex", alignItems:"center", justifyContent:"center",
background:"#0f172a", color:"white", fontFamily:"sans-serif", fontSize:18 }}>
      Отримуємо геолокацію...
    </div>
  );

  return (
    <>
      {/* ═══ АВТОРИЗАЦІЙНА МОДАЛКА ═══ */}
      {authModal && (
        <div style={{ position:"fixed", inset:0, zIndex:3000, background:"rgba(0,0,0,0.75)",
display:"flex", alignItems:"center", justifyContent:"center", padding:16 }}>
          <div style={{ background:"#1e293b", borderRadius:16, padding:28, width:"100%",
maxWidth:380, boxShadow:"0 20px 60px rgba(0,0,0,0.5)", fontFamily:"'Segoe UI',sans-serif" }}>
            <div style={{ fontSize:20, fontWeight:700, color:"white", marginBottom:4 }}>
              {authModal === "login" ? "Вхід" : "Реєстрація"}
            </div>
            <div style={{ fontSize:13, color:"#64748b", marginBottom:20 }}>
              {authModal === "login" ? "Увійдіть для збереження маршрутів" : "Створіть акаунт
для повного доступу"}
            </div>
            <form onSubmit={handleAuth}>
              {authModal === "register" && (
                <div style={{ marginBottom:12 }}>
                  <label style={{ fontSize:13, color:"#94a3b8", display:"block", marginBottom:6
}}>Ім'я</label>
                  <input placeholder="Ваше ім'я" value={authForm.name}
                    onChange={e => setAuthForm(f => ({ ...f, name: e.target.value }))}
                    style={{ width:"100%", padding:"12px 14px", borderRadius:10, border:"1.5px solid
#334155", background:"#0f172a", color:"white", fontSize:15, boxSizing:"border-box",
outline:"none" }} />
                </div>
              )}
              <div style={{ marginBottom:12 }}>
                <label style={{ fontSize:13, color:"#94a3b8", display:"block", marginBottom:6
}}>Email</label>
                <input type="email" placeholder="your@email.com" value={authForm.email}
                  onChange={e => setAuthForm(f => ({ ...f, email: e.target.value }))}
```

```
              style={{ width:"100%", padding:"12px 14px", borderRadius:10, border:"1.5px solid
#334155", background:"#0f172a", color:"white", fontSize:15, boxSizing:"border-box",
outline:"none" }} />
            </div>
            <div style={{ marginBottom:20 }}>
              <label style={{ fontSize:13, color:"#94a3b8", display:"block", marginBottom:6
}}>Пароль</label>
              <input type="password" placeholder="••••••••" value={authForm.password}
                onChange={e => setAuthForm(f => ({ ...f, password: e.target.value }))}
                style={{ width:"100%", padding:"12px 14px", borderRadius:10, border:"1.5px solid
#334155", background:"#0f172a", color:"white", fontSize:15, boxSizing:"border-box",
outline:"none" }} />
            </div>
            {authError && <div style={{ color:"#ef4444", fontSize:13, marginBottom:14
}}>{authError}</div>}
            <button type="submit" disabled={authLoading}
              style={{ width:"100%", padding:"14px", borderRadius:10, border:"none", background:
authLoading ? "#334155" : "#3b82f6", color:"white", fontSize:15, fontWeight:600,
cursor:"pointer", marginBottom:14 }}>
              {authLoading ? "..." : authModal === "login" ? "Увійти" : "Зареєструватись"}
            </button>
          </form>
          <div style={{ display:"flex", justifyContent:"space-between", alignItems:"center" }}>
            <span style={{ fontSize:13, color:"#64748b" }}>
              {authModal === "login" ? "Немає акаунту?" : "Вже є акаунт?"}
              <span onClick={() => { setAuthModal(authModal === "login" ? "register" : "login");
setAuthError(""); }}
                style={{ color:"#3b82f6", cursor:"pointer", marginLeft:6 }}>
                {authModal === "login" ? "Реєстрація" : "Увійти"}
              </span>
            </span>
            <span onClick={() => setAuthModal(null)} style={{ fontSize:13, color:"#475569",
cursor:"pointer" }}>Закрити</span>
          </div>
        </div>
      </div>
    )}

    {/* ═══ OVERLAY ═══ */}
    <div className={`overlay${sidebarOpen ? " overlay--visible" : ""}`}
      onClick={() => setSidebarOpen(false)} />

    {/* ═══ КНОПКА МЕНЮ ═══ */}
    <button className="menu-btn" onClick={() => setSidebarOpen(o => !o)}>
      {sidebarOpen ? "✕" : "☰"}
    </button>

    {/* ═══ САЙДБАР ═══ */}
    <div className={`sidebar${sidebarOpen ? "" : " sidebar--closed"}`}>

      {/* Шапка */}
```

```
      <div style={{ background:"linear-gradient(135deg,#1e40af,#3b82f6)", padding:"16px 16px
12px", flexShrink:0 }}>
        <div style={{ display:"flex", alignItems:"center", justifyContent:"space-between" }}>
          <div style={{ display:"flex", alignItems:"center", gap:10 }}>
            <span style={{ fontSize:24 }}>🚦</span>
            <div>
              <div style={{ fontSize:17, fontWeight:700 }}>SmartTrafficAI</div>
              <div style={{ fontSize:10, opacity:0.75 }}>Оцінка та прогнозування трафіку</div>
            </div>
          </div>
          {user ? (
            <div style={{ textAlign:"right" }}>
              <div style={{ fontSize:13, fontWeight:600 }}>{user.name}</div>
              <span onClick={logout} style={{ fontSize:11, opacity:0.7, cursor:"pointer"
}}>Вийти</span>
            </div>
          ) : (
            <button onClick={() => setAuthModal("login")}
              style={{ padding:"7px 14px", borderRadius:8, border:"1.5px solid
rgba(255,255,255,0.4)", background:"transparent", color:"white", fontSize:13, cursor:"pointer" }}>
              Увійти
            </button>
          )}
        </div>
      </div>

      {/* Вкладки */}
      <div style={{ display:"flex", background:"#0a1628", flexShrink:0 }}>
        {TABS.map(({ key, label, icon }) => (
          <button key={key} onClick={() => setActiveTab(key)}
            style={{ flex:1, padding:"11px 4px", border:"none",
              background: activeTab === key ? "#1e293b" : "transparent",
              borderBottom: activeTab === key ? "2px solid #3b82f6" : "2px solid transparent",
              color: activeTab === key ? "#60a5fa" : "#475569", cursor:"pointer",
              fontSize:11, fontWeight: activeTab === key ? 700 : 400,
              display:"flex", flexDirection:"column", alignItems:"center", gap:3 }}>
            <span style={{ fontSize:15 }}>{icon}</span>{label}
          </button>
        ))}
      </div>

      {/* ——— МАРШРУТ ——— */}
      {activeTab === "route" && (
        <div style={{ flex:1, overflowY:"auto", display:"flex", flexDirection:"column" }}>

          {/* Пошук */}
          <div style={{ padding:"14px 14px 0" }}>
            <div style={{ display:"flex", alignItems:"center", gap:8, background:"#1e293b",
              border:`1.5px solid ${inputFocused ? "#3b82f6" : "#334155"}`,
              borderRadius:12, padding:"11px 14px", transition:"border-color 0.2s" }}>
              <span style={{ fontSize:15, opacity:0.5 }}>🔍</span>
              <input type="text" placeholder="Введіть адресу призначення..."
```

```
            value={query} onChange={e => setQuery(e.target.value)}
            onFocus={() => setInputFocused(true)}
            onBlur={() => setTimeout(() => setInputFocused(false), 150)}
            onKeyDown={handleKeyDown}
            style={{ flex:1, background:"transparent", border:"none", outline:"none", color:"white",
fontSize:14 }} />
          {query && (
            <span onClick={() => { setQuery(""); setSuggestions([]); }}
              style={{ cursor:"pointer", opacity:0.5, fontSize:16, padding:4 }}>✕</span>
          )}
        </div>

        {inputFocused && query.length < 2 && searchHistory.length > 0 && (
          <div style={{ background:"#1e293b", borderRadius:12, marginTop:8, border:"1px solid
#334155", overflow:"hidden" }}>
            <div style={{ padding:"8px 14px 4px", fontSize:10, color:"#475569",
textTransform:"uppercase", letterSpacing:0.8 }}>Нещодавні</div>
            {searchHistory.map((item, i) => (
              <div key={i} onClick={() => selectSuggestion(item)}
                style={{ padding:"12px 14px", borderTop:"1px solid #1e3a5f", cursor:"pointer",
fontSize:13, display:"flex", alignItems:"center", gap:10 }}>
                <span>🕐</span>
                <span style={{ color:"#cbd5e1", overflow:"hidden", textOverflow:"ellipsis",
whiteSpace:"nowrap" }}>
                  {item.label || item.display_name.split(",").slice(0,2).join(",")}
                </span>
              </div>
            ))}
          </div>
        )}

        {suggestions.length > 0 && (
          <div style={{ background:"#1e293b", borderRadius:12, marginTop:8, border:"1px solid
#334155", overflow:"hidden" }}>
            {suggestions.map((item, i) => (
              <div key={i} onClick={() => selectSuggestion(item)}
                style={{ padding:"12px 14px",
                  borderBottom: i < suggestions.length-1 ? "1px solid #1e3a5f" : "none",
                  cursor:"pointer", fontSize:13, display:"flex", alignItems:"flex-start", gap:10,
                  background: i === activeIndex ? "#1e3a5f" : "transparent" }}>
                <span style={{ marginTop:1, flexShrink:0 }}>{getCategoryIcon(item)}</span>
                <div style={{ minWidth:0 }}>
                  <div style={{ color:"#e2e8f0", overflow:"hidden", textOverflow:"ellipsis",
whiteSpace:"nowrap" }}>
                    {item.display_name.split(",").slice(0,2).join(",").trim()}
                  </div>
                  <div style={{ color:"#64748b", fontSize:11, marginTop:2 }}>
                    {item.display_name.split(",").slice(2,4).join(",").trim()}
                  </div>
                  <div style={{ color:"#3b82f6", fontSize:11, marginTop:2
}}>{formatDist(item.dist)} від вас</div>
                </div>
```

```
            </div>
          ))}
        </div>
      )}
    </div>

    {/* Маршрути */}
    {routes.length > 0 && (
      <div style={{ padding:"14px 14px 0" }}>
        <div style={{ fontSize:10, color:"#475569", marginBottom:8,
textTransform:"uppercase", letterSpacing:0.8 }}>
          {routes.length > 1 ? `${routes.length} маршрути` : "Маршрут"}
          {lastUpdated && <span style={{ float:"right", color:"#334155" }}>оновлено
{lastUpdated}</span>}
        </div>

        {routes.map((r, i) => {
          const isSelected = i === selectedIdx;
          const dotColor   = i === 0 ? "#3b82f6" : ALT_COLORS[i-1] || "#64748b";
          return (
            <div key={i} onClick={() => { setSelectedIdx(i); if (window.innerWidth < 768)
setSidebarOpen(false); }}
              style={{ background: isSelected ? "#1e3a5f" : "#1e293b",
                border:`1.5px solid ${isSelected ? "#3b82f6" : "#334155"}`,
                borderRadius:12, padding:"12px 14px", marginBottom:8, cursor:"pointer",
transition:"all 0.2s" }}>
              <div style={{ display:"flex", alignItems:"center", justifyContent:"space-between",
marginBottom:8 }}>
                <div style={{ display:"flex", alignItems:"center", gap:8 }}>
                  <div style={{ width:12, height:12, borderRadius:"50%", background:dotColor }}
/>
                  <span style={{ fontSize:13, fontWeight:600, color: isSelected ? "#60a5fa" :
"#94a3b8" }}>
                    {i === 0 ? "Рекомендований" : `Альтернатива ${i}`}
                  </span>
                </div>
                {isSelected && trafficLoading && <span style={{ fontSize:12, color:"#64748b"
}}>⏳</span>}
              </div>
              <div style={{ display:"flex", gap:10, marginBottom:8 }}>
                {[
                  { val:(r.distance/1000).toFixed(1), label:"км" },
                  { val:Math.round(r.duration/60),    label:"хв" },
                  { val:arrivalTime(r.duration),      label:"прибуття", color:"#22c55e" },
                ].map(({ val, label, color }) => (
                  <div key={label} style={{ textAlign:"center", flex:1 }}>
                    <div style={{ fontSize:19, fontWeight:700, color: color || "#e2e8f0"
}}>{val}</div>
                    <div style={{ fontSize:10, color:"#64748b" }}>{label}</div>
                  </div>
                ))}
              </div>
```

```
                <TrafficBar g={r.pct_green} y={r.pct_yellow} r={r.pct_red} height={7} />
              </div>
            );
          })}

          {!user && (
            <div style={{ fontSize:12, color:"#475569", textAlign:"center", marginTop:4 }}>
              <span onClick={() => setAuthModal("login")} style={{ color:"#3b82f6",
cursor:"pointer" }}>Увійдіть</span>, щоб зберегти маршрут
            </div>
          )}
        </div>
      )}

      {/* Легенда */}
      <div style={{ padding:"14px 14px 0" }}>
        <div style={{ background:"#1e293b", borderRadius:12, border:"1px solid #334155",
padding:"12px 14px" }}>
          {[
            { color:"#22c55e", label:"Вільний рух" },
            { color:"#eab308", label:"Помірна завантаженість" },
            { color:"#ef4444", label:"Затор" },
          ].map(({ color, label }) => (
            <div key={color} style={{ display:"flex", alignItems:"center", gap:10, marginBottom:6
}}>
              <div style={{ width:28, height:6, borderRadius:3, background:color, flexShrink:0 }} />
              <span style={{ fontSize:13, color:"#cbd5e1" }}>{label}</span>
            </div>
          ))}
        </div>
      </div>

      <div style={{ padding:"10px 16px 16px", fontSize:10, color:"#334155", textAlign:"center"
}}>
        Дані трафіку: Mapbox Traffic API
      </div>
    </div>
  )}

  {/* ——— ІСТОРІЯ ——— */}
  {activeTab === "history" && (
    <div style={{ flex:1, overflowY:"auto", padding:14 }}>
      {!user ? (
        <div style={{ textAlign:"center", padding:"40px 0" }}>
          <div style={{ fontSize:40, marginBottom:12 }}>🔒</div>
          <div style={{ color:"#94a3b8", fontSize:14, marginBottom:20 }}>Увійдіть, щоб
бачити<br/>історію маршрутів</div>
          <button onClick={() => setAuthModal("login")}
            style={{ padding:"12px 28px", borderRadius:10, border:"none", background:"#3b82f6",
color:"white", fontSize:14, cursor:"pointer", fontWeight:600 }}>
            Увійти
          </button>
```

```
            </div>
          ) : routeHistory.length === 0 ? (
            <div style={{ textAlign:"center", padding:"40px 0", color:"#475569", fontSize:14 }}>
              <div style={{ fontSize:40, marginBottom:12 }}>🗺️</div>
              Ще немає збережених маршрутів.<br/>Побудуйте перший!
            </div>
          ) : (
            <>
              <div style={{ fontSize:10, color:"#475569", marginBottom:12,
textTransform:"uppercase", letterSpacing:0.8 }}>
                Останні {routeHistory.length} маршрутів
              </div>
              {routeHistory.map(r => (
                <div key={r.id} style={{ background:"#1e293b", borderRadius:12, border:"1px solid
#334155", padding:14, marginBottom:10 }}>
                  <div style={{ fontSize:13, color:"#e2e8f0", marginBottom:6, overflow:"hidden",
textOverflow:"ellipsis", whiteSpace:"nowrap" }}>📍 {r.end_address}</div>
                  <div style={{ display:"flex", gap:12, fontSize:12, color:"#64748b", marginBottom:10
}}>
                    <span>🛣️ {r.distance_km} км</span>
                    <span>⏱️ {r.duration_min} хв</span>
                    <span style={{ marginLeft:"auto" }}>{formatTime(r.created_at)}</span>
                  </div>
                  <TrafficBar g={r.pct_green} y={r.pct_yellow} r={r.pct_red} height={7} />
                </div>
              ))}
            </>
          )}
        </div>
      )}

      {/* ——— ПРОГНОЗ ——— */}
      {activeTab === "forecast" && (
        <div style={{ flex:1, overflowY:"auto", padding:14 }}>
          <div style={{ fontSize:10, color:"#475569", marginBottom:4, textTransform:"uppercase",
letterSpacing:0.8 }}>Прогноз завантаженості по годинах</div>
          <div style={{ fontSize:11, color:"#334155", marginBottom:14 }}>На основі накопичених
даних усіх користувачів</div>

          {forecastData.length === 0 ? (
            <div style={{ textAlign:"center", padding:"30px 0", color:"#475569", fontSize:14 }}>
              <div style={{ fontSize:40, marginBottom:12 }}>📊</div>
              Дані накопичуються.<br/>Будуйте маршрути – і тут з'явиться прогноз.
            </div>
          ) : (
            <>
              {(() => {
                const now = new Date().getHours();
                const cur = forecastData.find(d => d.hour_of_day === now);
                if (!cur) return null;
                const status = cur.avg_red > 30 ? "🔴 Очікуються затори" : cur.avg_yellow > 30 ?
"🟡 Помірний рух" : "🟢 Вільно";
```

```
            return (
              <div style={{ background:"#1e3a5f", borderRadius:12, border:"1px solid #3b82f6",
padding:"14px 16px", marginBottom:14 }}>
                <div style={{ fontSize:11, color:"#60a5fa", marginBottom:4 }}>Зараз
{now}:00</div>
                <div style={{ fontSize:15, fontWeight:600, color:"white" }}>{status}</div>
                <div style={{ fontSize:12, color:"#64748b", marginTop:4 }}>{cur.samples}
спостережень</div>
              </div>
            );
          })()}
          <div style={{ background:"#1e293b", borderRadius:12, border:"1px solid #334155",
padding:"14px 10px" }}>
            <div style={{ display:"flex", alignItems:"flex-end", gap:2, height:100, marginBottom:6
}}>
              {Array.from({ length:24 }, (_, h) => {
                const d = forecastData.find(f => f.hour_of_day === h);
                const isNow = h === new Date().getHours();
                if (!d) return <div key={h} style={{ flex:1 }} />;
                return (
                  <div key={h} title={`${h}:00`}
                    style={{ flex:1, display:"flex", flexDirection:"column", justifyContent:"flex-end",
outline: isNow ? "1.5px solid #3b82f6" : "none", borderRadius:2 }}>
                    {d.avg_red    > 0 && <div style={{ height:`${d.avg_red}%`,
background:"#ef4444", borderRadius:"2px 2px 0 0" }} />}
                    {d.avg_yellow > 0 && <div style={{ height:`${d.avg_yellow}%`,
background:"#eab308" }} />}
                    {d.avg_green  > 0 && <div style={{ height:`${d.avg_green}%`,
background:"#22c55e" }} />}
                  </div>
                );
              })}
            </div>
            <div style={{ display:"flex", gap:2 }}>
              {Array.from({ length:24 }, (_, h) => (
                <div key={h} style={{ flex:1, textAlign:"center", fontSize:8,
                  color: h === new Date().getHours() ? "#60a5fa" : "#334155",
                  fontWeight: h === new Date().getHours() ? 700 : 400 }}>
                  {h % 3 === 0 ? h : ""}
                </div>
              ))}
            </div>
          </div>
          <div style={{ marginTop:14 }}>
            <div style={{ fontSize:10, color:"#475569", marginBottom:10,
textTransform:"uppercase", letterSpacing:0.8 }}>Найзавантаженіші години</div>
            {[...forecastData].sort((a,b) => (b.avg_red+b.avg_yellow)-
(a.avg_red+a.avg_yellow)).slice(0,3).map(d => (
              <div key={d.hour_of_day} style={{ display:"flex", alignItems:"center", gap:10,
padding:"10px 0", borderBottom:"1px solid #1e293b" }}>
                <div style={{ fontSize:13, color:"#60a5fa", fontWeight:700, width:40
}}>{String(d.hour_of_day).padStart(2,"0")}:00</div>
```

```
                <div style={{ flex:1 }}><TrafficBar g={d.avg_green} y={d.avg_yellow}
r={d.avg_red} height={8} /></div>
                <div style={{ fontSize:12, color:"#ef4444", width:36, textAlign:"right"
}}>{d.avg_red}% 🔴</div>
              </div>
            ))}
          </div>
        </>
      )}
    </div>
  )}
</div>

{/* ═══ НИЖНЯ ПАНЕЛЬ ═══ */}
<div className={`bottom-bar${selected && !sidebarOpen ? " bottom-bar--visible" : ""}`}
 onClick={() => setSidebarOpen(true)}>
 <div style={{ display:"flex", gap:24 }}>
  <div style={{ textAlign:"center" }}>
   <div style={{ fontSize:20, fontWeight:700, color:"#e2e8f0" }}>{selected ?
(selected.distance/1000).toFixed(1) : ""}</div>
   <div style={{ fontSize:11, color:"#64748b" }}>км</div>
  </div>
  <div style={{ textAlign:"center" }}>
   <div style={{ fontSize:20, fontWeight:700, color:"#e2e8f0" }}>{selected ?
Math.round(selected.duration/60) : ""}</div>
   <div style={{ fontSize:11, color:"#64748b" }}>хв</div>
  </div>
  <div style={{ textAlign:"center" }}>
   <div style={{ fontSize:20, fontWeight:700, color:"#22c55e" }}>{selected ?
arrivalTime(selected.duration) : ""}</div>
   <div style={{ fontSize:11, color:"#64748b" }}>прибуття</div>
  </div>
 </div>
 {selected && (
  <div style={{ display:"flex", flexDirection:"column", alignItems:"flex-end", gap:6,
minWidth:80 }}>
   <TrafficBar g={selected.pct_green} y={selected.pct_yellow} r={selected.pct_red}
height={8} />
   <div style={{ fontSize:12, color:"#60a5fa" }}>Деталі ↑</div>
  </div>
 )}
</div>

{/* ═══ КАРТА ═══ */}
<div className="map-wrap">
 <MapContainer center={position} zoom={16} style={{ height:"100%", width:"100%" }}
zoomControl={false}>
  <MapUpdater position={position} hasRoute={routes.length > 0} />
  <ZoomControl position="bottomright" />
  <TileLayer url="https://{s}.basemaps.cartocdn.com/light_all/{z}/{x}/{y}{r}.png"
   attribution='&copy; <a href="https://carto.com/">CARTO</a>' />
  <Marker position={position} icon={userIcon} />
```

```
        {destination && <Marker position={destination} icon={destIcon} />}
        {routes.map((r, i) => i === selectedIdx ? null : (
          <Polyline key={`alt-${i}`} positions={r.geometry} color={ALT_COLORS[i-1] ||
"#64748b"} weight={4} opacity={0.5} />
        ))}
        {selected && selected.colors.length > 0 && selected.geometry.map((_, i) => {
          if (i === selected.geometry.length - 1) return null;
          return (
            <Polyline key={`seg-${i}`}
              positions={[selected.geometry[i], selected.geometry[i+1]]}
              color={COLOR_MAP[selected.colors[i]] || "#22c55e"} weight={6} opacity={0.9} />
          );
        })}
        {trafficLoading && routes.length === 0 && destination && (
          <Polyline positions={[position, destination]} color="#3b82f6" weight={4} opacity={0.4}
dashArray="8" />
        )}
      </MapContainer>
    </div>
  </>
 );
}
```

Лістинг index,js

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import "./index.css";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<App />);
```

Лістинг index.css

```
* {
  box-sizing: border-box;
}

html, body, #root {
  margin: 0;
  padding: 0;
  height: 100%;
  overflow: hidden;
}

/* ——— Сайдбар ——— */
.sidebar {
  position: fixed;
  z-index: 1000;
  top: 0; left: 0; bottom: 0;
  width: 320px;
  background: #0f172a;
  display: flex;
  flex-direction: column;
```

```
  box-shadow: 4px 0 20px rgba(0,0,0,0.4);
  font-family: 'Segoe UI', sans-serif;
  color: white;
  transition: transform 0.3s ease;
  will-change: transform;
  overflow: hidden;
}

.sidebar.sidebar--closed {
  transform: translateX(-100%);
}

/* ——— Карта ——— */
.map-wrap {
  margin-left: 320px;
  height: 100vh;
}

/* ——— Кнопка меню ——— */
.menu-btn {
  display: none;
  position: fixed;
  z-index: 1100;
  top: 16px;
  left: 16px;
  width: 50px;
  height: 50px;
  background: #1e40af;
  border: none;
  border-radius: 12px;
  color: white;
  font-size: 24px;
  cursor: pointer;
  box-shadow: 0 4px 16px rgba(0,0,0,0.5);
  align-items: center;
  justify-content: center;
  touch-action: manipulation;
}

/* ——— Overlay ——— */
.overlay {
  display: none;
  position: fixed;
  top: 0; left: 0; right: 0; bottom: 0;
  z-index: 999;
  background: rgba(0,0,0,0.55);
}

/* ——— Нижня панель (мобільна) ——— */
.bottom-bar {
  display: none;
  position: fixed;
```

```
  bottom: 0; left: 0; right: 0;
  z-index: 998;
  background: #1e293b;
  border-top: 2px solid #3b82f6;
  padding: 12px 20px;
  cursor: pointer;
  align-items: center;
  justify-content: space-between;
  font-family: 'Segoe UI', sans-serif;
}

/* ——— МОБІЛЬНІ ——— */
@media (max-width: 767px) {
  .sidebar {
    width: 88%;
    max-width: 360px;
  }

  .map-wrap {
    margin-left: 0;
  }

  .menu-btn {
    display: flex;
  }

  .overlay.overlay--visible {
    display: block;
  }

  .bottom-bar.bottom-bar--visible {
    display: flex;
  }
}

/* ——— Маркер геолокації ——— */
.user-marker {
  position: relative;
  width: 20px;
  height: 20px;
}

.user-marker-dot {
  position: absolute;
  top: 50%; left: 50%;
  transform: translate(-50%, -50%);
  width: 16px; height: 16px;
  background: #2563eb;
  border: 3px solid white;
  border-radius: 50%;
  box-shadow: 0 2px 6px rgba(0,0,0,0.4);
  z-index: 2;
```

```
}

.user-marker-pulse {
  position: absolute;
  top: 50%; left: 50%;
  transform: translate(-50%, -50%);
  width: 16px; height: 16px;
  background: rgba(37, 99, 235, 0.35);
  border-radius: 50%;
  animation: pulse 2s ease-out infinite;
  z-index: 1;
}

@keyframes pulse {
  0%   { transform: translate(-50%, -50%) scale(1);   opacity: 1; }
  100% { transform: translate(-50%, -50%) scale(3.5); opacity: 0; }
}
```

Лістинг app.js

```
require("dotenv").config();
const express = require("express");
const cors    = require("cors");
const axios   = require("axios");
const { Pool } = require("pg");
const bcrypt  = require("bcryptjs");
const jwt     = require("jsonwebtoken");

const app = express();
app.use(cors({
  origin: (origin, cb) => {
    if (!origin) return cb(null, true);
    if (origin === (process.env.CLIENT_URL || "http://localhost:3000")) return cb(null, true);
    if (/\.vercel\.app$/.test(origin)) return cb(null, true);
    if (origin === "http://localhost:3000") return cb(null, true);
    cb(new Error("CORS: not allowed"));
  },
  allowedHeaders: ["Content-Type", "Authorization"]
}));
app.use(express.json());

const MAPBOX_TOKEN = process.env.MAPBOX_TOKEN;
const JWT_SECRET   = process.env.JWT_SECRET || "smarttraffic_diploma_2025";

const pool = process.env.DATABASE_URL
  ? new Pool({ connectionString: process.env.DATABASE_URL, ssl: { rejectUnauthorized: false }
})
  : new Pool({
      user:     "postgres",
      host:     "localhost",
      database: "smarttraffic",
      password: process.env.DB_PASSWORD || "Maaxxim65_",
      port:     5432,
```

```
  });

pool.query(`
  CREATE TABLE IF NOT EXISTS users (
    id           SERIAL PRIMARY KEY,
    name         VARCHAR(100)  NOT NULL,
    email        VARCHAR(150)  UNIQUE NOT NULL,
    password_hash VARCHAR(255)  NOT NULL,
    created_at    TIMESTAMP     DEFAULT NOW()
  );
  CREATE TABLE IF NOT EXISTS routes (
    id          SERIAL PRIMARY KEY,
    user_id     INTEGER REFERENCES users(id) ON DELETE CASCADE,
    end_address  TEXT,
    start_lat    FLOAT  NOT NULL,
    start_lon    FLOAT  NOT NULL,
    end_lat      FLOAT  NOT NULL,
    end_lon      FLOAT  NOT NULL,
    distance_km  FLOAT,
    duration_min INTEGER,
    pct_green    INTEGER DEFAULT 0,
    pct_yellow   INTEGER DEFAULT 0,
    pct_red      INTEGER DEFAULT 0,
    created_at   TIMESTAMP DEFAULT NOW()
  );
  CREATE TABLE IF NOT EXISTS traffic_logs (
    id          SERIAL PRIMARY KEY,
    route_id     INTEGER REFERENCES routes(id) ON DELETE CASCADE,
    hour_of_day  SMALLINT NOT NULL,
    day_of_week  SMALLINT NOT NULL,
    pct_green    INTEGER DEFAULT 0,
    pct_yellow   INTEGER DEFAULT 0,
    pct_red      INTEGER DEFAULT 0,
    recorded_at  TIMESTAMP DEFAULT NOW()
  );
`).then(() => console.log("БД ініціалізована")).catch(console.error);

// ——— Утиліти
const congestionToColor = (c) => {
  if (c === "severe" || c === "heavy") return "red";
  if (c === "moderate") return "yellow";
  return "green";
};

const authMiddleware = (req, res, next) => {
  const token = req.headers.authorization?.split(" ")[1];
  if (!token) return res.status(401).json({ error: "Не авторизовано" });
  try { req.user = jwt.verify(token, JWT_SECRET); next(); }
  catch { res.status(401).json({ error: "Невалідний токен" }); }
};

const optionalAuth = (req, res, next) => {
```

```
  const token = req.headers.authorization?.split(" ")[1];
  if (token) { try { req.user = jwt.verify(token, JWT_SECRET); } catch {} }
  next();
};

// Обробка одного маршруту з Mapbox у внутрішній формат
const processRoute = (route) => {
  const leg          = route.legs[0].annotation;
  const latlngs      = route.geometry.coordinates.map(c => [c[1], c[0]]);
  const rawCongestion = leg.congestion         || [];
  const rawNumeric    = leg.congestion_numeric || [];
  const rawSpeed      = leg.speed              || [];
  const rawMaxspeed   = leg.maxspeed           || [];

  const colors = rawCongestion.map((c, i) => {
    const speed    = rawSpeed[i];
    const maxspeed = rawMaxspeed[i]?.speed;

    if (speed != null && maxspeed) {
      const ratio = (speed * 3.6) / maxspeed;
      if (ratio >= 0.75) return "green";
      if (ratio >= 0.40) return "yellow";
      return "red";
    }
    const num = rawNumeric[i];
    if (num != null) {
      if (num >= 60) return "red";
      if (num >= 25) return "yellow";
      return "green";
    }
    return congestionToColor(c);
  });

  const total      = colors.length || 1;
  const pct_green  = Math.round(colors.filter(c => c === "green").length  / total * 100);
  const pct_yellow = Math.round(colors.filter(c => c === "yellow").length / total * 100);
  const pct_red    = Math.round(colors.filter(c => c === "red").length    / total * 100);

  return { geometry: latlngs, colors, distance: route.distance, duration: route.duration, pct_green,
pct_yellow, pct_red };
};

// ——— Авторизація
app.post("/api/auth/register", async (req, res) => {
  const { name, email, password } = req.body;
  if (!name || !email || !password)
    return res.status(400).json({ error: "Заповніть всі поля" });
  try {
    const hash   = await bcrypt.hash(password, 10);
    const result = await pool.query(
      "INSERT INTO users (name, email, password_hash) VALUES ($1,$2,$3) RETURNING id,
name, email",
```

```
      [name, email, hash]
    );
    const user  = result.rows[0];
    const token = jwt.sign({ id: user.id, name: user.name, email: user.email }, JWT_SECRET, {
expiresIn: "30d" });
    res.json({ user, token });
  } catch (err) {
    if (err.code === "23505") return res.status(400).json({ error: "Email вже зареєстровано" });
    res.status(500).json({ error: "Помилка сервера" });
  }
});

app.post("/api/auth/login", async (req, res) => {
  const { email, password } = req.body;
  try {
    const result = await pool.query("SELECT * FROM users WHERE email = $1", [email]);
    const user   = result.rows[0];
    if (!user) return res.status(400).json({ error: "Невірний email або пароль" });
    const valid  = await bcrypt.compare(password, user.password_hash);
    if (!valid)  return res.status(400).json({ error: "Невірний email або пароль" });
    const token  = jwt.sign({ id: user.id, name: user.name, email: user.email }, JWT_SECRET, {
expiresIn: "30d" });
    res.json({ user: { id: user.id, name: user.name, email: user.email }, token });
  } catch { res.status(500).json({ error: "Помилка сервера" }); }
});

// ─── Маршрут з альтернативами
app.get("/api/route", optionalAuth, async (req, res) => {
  const { start, end, address } = req.query;
  try {
    const response = await axios.get(
      `https://api.mapbox.com/directions/v5/mapbox/driving-traffic/${start};${end}`,
      {
        params: {
          geometries: "geojson",
          overview: "full",
          annotations: "congestion,congestion_numeric,speed,maxspeed",
          alternatives: true,
          access_token: MAPBOX_TOKEN
        }
      }
    );

    const processed = response.data.routes.map(processRoute);
    const main      = processed[0];

    console.log(`[Mapbox] ${processed.length} маршрут(и) | головний:
${(main.distance/1000).toFixed(1)} км | зел=${main.pct_green}% жов=${main.pct_yellow}%
чер=${main.pct_red}%`);

    if (req.user) {
      const [sLon, sLat] = start.split(",").map(Number);
```

```
      const [eLon, eLat] = end.split(",").map(Number);
      const saved = await pool.query(
        `INSERT INTO routes
(user_id,end_address,start_lat,start_lon,end_lat,end_lon,distance_km,duration_min,pct_green,pct_y
ellow,pct_red)
         VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11) RETURNING id`,
        [req.user.id, address || "Невідома адреса", sLat, sLon, eLat, eLon,
         parseFloat((main.distance/1000).toFixed(2)), Math.round(main.duration/60),
         main.pct_green, main.pct_yellow, main.pct_red]
      );
      const now = new Date();
      await pool.query(
        `INSERT INTO traffic_logs
(route_id,hour_of_day,day_of_week,pct_green,pct_yellow,pct_red) VALUES ($1,$2,$3,$4,$5,$6)`,
        [saved.rows[0].id, now.getHours(), now.getDay(), main.pct_green, main.pct_yellow,
main.pct_red]
      );
    }

    res.json({ routes: processed });
  } catch (err) {
    console.error("Помилка маршруту:", err.response?.data || err.message);
    res.status(500).json({ error: "Routing error", details: err.response?.data || err.message });
  }
});

// ─── Історія маршрутів
app.get("/api/routes/history", authMiddleware, async (req, res) => {
  try {
    const result = await pool.query(
      `SELECT id, end_address, distance_km, duration_min, pct_green, pct_yellow, pct_red,
created_at
       FROM routes WHERE user_id = $1 ORDER BY created_at DESC LIMIT 20`,
      [req.user.id]
    );
    res.json(result.rows);
  } catch { res.status(500).json({ error: "Помилка сервера" }); }
});

// ─── Прогноз
app.get("/api/forecast", async (req, res) => {
  try {
    const result = await pool.query(`
      SELECT
        hour_of_day,
        ROUND(AVG(pct_green))::int  AS avg_green,
        ROUND(AVG(pct_yellow))::int AS avg_yellow,
        ROUND(AVG(pct_red))::int    AS avg_red,
        COUNT(*)::int               AS samples
      FROM traffic_logs
      GROUP BY hour_of_day
      ORDER BY hour_of_day
```

```
    `);
    res.json(result.rows);
  } catch { res.status(500).json({ error: "Помилка сервера" }); }
});

// ——— Демо-кольори
app.post("/api/traffic-colors", (req, res) => {
  const { route, mode } = req.body;
  if (!route || route.length < 2) return res.json({ colors: [] });
  const blockSize = Math.max(1, Math.floor(route.length / 8));
  const patterns = {
    demo_moderate: ["green","yellow","yellow","green","yellow","green","yellow","green"],
    demo_heavy:    ["yellow","red","red","yellow","red","red","yellow","red"]
  };
  const pattern = patterns[mode];
  if (!pattern) return res.json({ colors: [] });
  res.json({ colors: Array.from({ length: route.length - 1 }, (_, i) => pattern[Math.floor(i/blockSize)
% pattern.length]) });
});

app.listen(5000, () => console.log("Сервер запущено на http://localhost:5000"));
```

ДОДАТОК Б

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Гладкий Максим Єгорович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення оцінки та прогнозування рівня міського транспортного трафіку» 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. _____ **Максим ГЛАДКИЙ**