Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти – магістр

за освітньо-професійною програмою

«Комп'ютерні науки»

зі спеціальності

122 – Комп'ютерні науки

тема роботи:

«*Розробка інформаційної системи контролю за переміщенням міського пасажирського транспорту*»

Виконав студент гр. КН-24м _____ Мізюкін П. В.

Керівник _____ Харламенко В. Ю.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2025

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Магістр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« *13* » *травня* 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентові групи КН-24м *Мізюкіну Павлу Володимировичу*

1. Тема кваліфікаційної роботи: *«Розробка інформаційної системи контролю за переміщенням міського пасажирського транспорту»*

затверджено наказом по університету № *257с* від *13.05.2025 р.*

2. Термін здачі кваліфікаційної роботи: *01.12.2025 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 102 с., додатки, презентація у Microsoft PowerPoint (16 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-3 ст. викл. Харламенко В. Ю.

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*10.07.25*
2	*Розділ 1*	*15.07.25*
3	*Розділ 2*	*15.08.25*
4	*Висновки*	*15.09.25*
5	*Оформлення кваліфікаційної роботи*	*20.11.25*
6	*Підготовка презентації та графічного матеріалу*	*28.11.25*
7	*Підготовка доповіді до захисту*	*01.12.25*

6. Дата видачі завдання: *13.05.2025р.*

Керівник _____ / Хармаленко В. Ю./

7. Запевнення: *Я, Мізюкін Павло Володимирович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студент _____ / Мізюкін П.В./

АНОТАЦІЯ

Мізюкін П.В. Розробка інформаційної системи контролю за переміщенням міського пасажирського транспорту.

Кваліфікаційна робота на здобуття ступеня вищої освіти – магістр, за спеціальністю 122 Комп'ютерні науки. Криворізький національний університет, Кривий Ріг, 2025.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та списку використаної літератури з 30 позицій. Загальний обсяг роботи становить 102 сторінки, з яких основний зміст викладено на 76 сторінках. Включено 7 таблиць та 28 рисунки.

У роботі розглянуто теоретичні та практичні аспекти розробки інформаційної системи контролю за переміщенням міського пасажирського транспорту. Проведено аналіз сучасних систем моніторингу транспорту та визначено їх основні технічні характеристики. Запропоновано архітектуру системи, яка забезпечує обробку даних від GPS-трекерів через API перевізника та їх подальше збереження у базі даних MySQL. Серверна частина реалізована на платформі Node.js, а клієнтська частина – у вигляді веб-інтерфейсу на основі JavaScript, React та картографічної бібліотеки Leaflet для відображення маршрутів руху транспорту.

Описано процеси проєктування бази даних, структуру серверної логіки, механізми отримання й оновлення даних, а також функціональність веб-інтерфейсу для візуалізації та аналітики. Проведено тестування системи, що підтвердило її працездатність і ефективність у забезпеченні моніторингу міських перевезень у режимі реального часу.

Ключові слова:

ІНФОРМАЦІЙНА СИСТЕМА, МІСЬКИЙ ТРАНСПОРТ, GPS-МОНІТОРИНГ, NODE.JS, MYSQL, REACT, JAVASCRIPT, LEAFLET, API.

ANNOTATION

Mizyukin P.V. Development of an Information System for Monitoring Urban Public Transport Movement.

Qualification thesis for obtaining the Master's degree in specialty 122 – Computer Science.

Kryvyi Rih National University, Kryvyi Rih, 2025.

The qualification thesis consists of an introduction, three chapters, conclusions, and a list of references comprising 30 sources. The total volume of the thesis is 80 pages, of which 102 contain the main content. The work includes 7 tables and 28 figures.

The thesis examines the theoretical and practical aspects of developing an information system for monitoring urban public transport movement. An analysis of existing transport monitoring systems was conducted, and their main technical characteristics were identified. The proposed system architecture ensures the processing of data from GPS trackers via the carrier's API and their subsequent storage in a MySQL database. The server side is implemented using the Node.js platform, while the client side is a web interface built with JavaScript, React, and the Leaflet mapping library for visualizing transport routes.

The thesis describes the processes of database design, the structure of server logic, data retrieval and update mechanisms, as well as the functionality of the web interface for visualization and analytics. System testing was conducted, confirming its operability and efficiency in providing real-time monitoring of urban transportation.

Keywords:

INFORMATION SYSTEM, URBAN TRANSPORT, GPS MONITORING, NODE.JS, MYSQL, REACT, JAVASCRIPT, LEAFLET, API.

ЗМІСТ

ВСТУП

Сучасний розвиток міської інфраструктури потребує впровадження цифрових технологій для підвищення ефективності управління транспортними процесами. Зростання кількості транспортних засобів, нерівномірність пасажиропотоків та необхідність оптимізації маршрутів зумовлюють актуальність створення інформаційних систем моніторингу та аналізу руху міського пасажирського транспорту.

Одним із важливих напрямів цифровізації міського середовища є впровадження систем контролю за переміщенням транспортних засобів, які дозволяють відстежувати їх розташування у реальному часі, аналізувати ефективність маршрутів, покращувати планування розкладів і підвищувати якість транспортного обслуговування населення. Такі системи забезпечують прозорість діяльності перевізників, зменшують кількість затримок, сприяють економії ресурсів і підвищують довіру пасажирів до громадського транспорту.

На сьогодні в Україні та світі існує низка комерційних і муніципальних систем моніторингу транспорту (Dozor, EasyWay, Eway тощо), однак більшість із них мають обмеження у масштабованості, відсутність гнучких аналітичних інструментів або недоступність відкритих даних для подальшої інтеграції з іншими сервісами. Це створює потребу у розробці власних інформаційних рішень, які б відповідали сучасним вимогам до продуктивності, зручності використання та адаптивності.

Метою даної роботи є розробка інформаційної системи контролю за переміщенням міського пасажирського транспорту, що забезпечує збір, обробку та візуалізацію даних про рух транспортних засобів у режимі реального часу.

Для досягнення поставленої мети у роботі вирішуються такі основні завдання:

1. Провести аналіз існуючих систем та технологій моніторингу транспорту.

2. Розробити архітектуру інформаційної системи контролю за

переміщенням транспорту.

3. Спроєктувати базу даних для збереження інформації про транспортні засоби, маршрути та координати їх переміщення.

4. Реалізувати серверну частину системи на платформі Node.js з використанням бази даних MySQL.

5. Створити клієнтську частину системи на основі JavaScript, React та Leaflet для відображення геолокаційних даних.

6. Провести тестування та оцінити ефективність роботи системи.

Об’єкт дослідження – процес контролю та аналізу переміщення міського пасажирського транспорту.

Предмет дослідження – методи, моделі та програмні засоби розробки інформаційної системи контролю руху транспорту.

Методи дослідження, використані у роботі: системний аналіз, об’єктно-орієнтоване моделювання, методи програмної інженерії, технології веброзробки, засоби візуалізації просторових даних.

Практичне значення результатів роботи полягає у створенні прототипу інформаційної системи, який може бути використаний міськими транспортними компаніями для підвищення ефективності перевезень та оптимізації маршрутів.

Отже, проведене дослідження та розробка інформаційної системи контролю за переміщенням міського пасажирського транспорту спрямовані на вирішення актуальної проблеми підвищення ефективності управління транспортними процесами в умовах розвитку інтелектуальних транспортних систем. Реалізація програмного комплексу дозволяє забезпечити автоматизований збір, обробку та відображення даних про рух транспортних засобів у режимі реального часу, що створює передумови для підвищення прозорості, оперативності та якості транспортного обслуговування населення.

Запропонована система має потенціал подальшого розвитку – зокрема, розширення функціональності за рахунок впровадження аналітичних модулів для прогнозування пасажиропотоку, інтеграції з мобільними додатками для пасажирів, а також використання технологій машинного навчання для

оптимізації маршрутів.

Отримані результати можуть бути застосовані у діяльності міських транспортних підприємств, органів місцевого самоврядування, а також компаній, що займаються GPS-моніторингом та аналітикою руху транспорту. Розроблений підхід демонструє практичну цінність використання сучасних вебтехнологій – Node.js, MySQL, React та Leaflet – для створення гнучких, масштабованих та зручних у використанні інформаційних систем.

Таким чином, виконана кваліфікаційна робота має як наукову, так і прикладну значущість, а розроблена інформаційна система може стати основою для подальших досліджень та розширення можливостей у сфері цифрового управління міським транспортом.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПІДХОДІВ ДО РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ КОНТРОЛЮ МІСЬКОГО ТРАНСПОРТУ

1.1 Актуальність предметної області

Інформаційна система контролю за переміщенням міського пасажирського транспорту – це комплексне програмне рішення, яке реалізується у вигляді веб-платформи або спеціалізованого програмного забезпечення. Її головною функцією є здійснення моніторингу, обліку та управління рухом транспортних засобів у межах міської транспортної мережі. Основна мета впровадження подібних систем полягає у підвищенні ефективності роботи громадського транспорту, зниженні кількості затримок і покращенні рівня інформування пасажирів щодо пересування маршрутів.

Робота інформаційної системи спрямована на відстеження поточного місцезнаходження транспорту, розрахунок прогнозованого часу прибуття на зупинки, контроль дотримання графіків руху, виявлення відхилень від маршруту та підтримку диспетчерського управління у випадках нестандартних або аварійних ситуацій. Крім того, подібні системи можуть інтегруватися з електронними інформаційними табло, мобільними застосунками для пасажирів та адміністративними веб-порталами для ведення звітності, статистичного аналізу та планування транспортних потоків.

Такі рішення активно впроваджуються в рамках концепції «розумного міста» (Smart City), оскільки сприяють раціональному використанню транспортної інфраструктури, підвищенню комфорту користувачів та зменшенню транспортних заторів. Завдяки ефективнішому плануванню маршрутів і контролю за їх виконанням знижується час очікування, покращується якість перевезень і зменшується негативний вплив транспорту на довкілля.

Інформаційні системи цього типу забезпечують оперативну підтримку

процесу прийняття управлінських рішень для міських служб і сприяють підвищенню ефективності функціонування транспортної мережі. Вони є невід'ємним елементом сучасних цифрових платформ управління міською інфраструктурою, що спрямовані на підвищення якості послуг для населення.

У сучасних умовах урбанізації та зростання інтенсивності транспортних потоків ефективна організація міських пасажирських перевезень набуває стратегічного значення. Збільшення кількості транспортних засобів, нерівномірність пасажиропотоків у різні години доби та постійна зміна транспортних маршрутів створюють значне навантаження на міську транспортну інфраструктуру. Це, у свою чергу, призводить до виникнення заторів, зниження пунктуальності руху, збільшення часу очікування пасажирів та нераціонального використання паливно-енергетичних ресурсів [6].

Управління транспортними потоками без використання сучасних інформаційних технологій є малоефективним, оскільки воно не забезпечує своєчасного отримання та аналізу даних про місцезнаходження транспортних засобів у реальному часі. Традиційні методи контролю, засновані на ручному обліку та звітності, не дозволяють оперативно реагувати на зміни у дорожній ситуації та приймати обґрунтовані управлінські рішення.

У зв'язку з цим виникає необхідність впровадження інформаційних систем моніторингу міського транспорту, які забезпечують збір, передачу, обробку та візуалізацію даних про переміщення транспортних засобів у режимі реального часу. Такі системи дозволяють не лише контролювати дотримання розкладу руху, але й аналізувати ефективність маршрутів, виявляти відхилення від графіків, формувати статистичні звіти та прогнозувати навантаження на окремі напрямки перевезень.

Використання GPS-трекерів і засобів автоматизованого збору даних забезпечує точність і повноту інформації про переміщення транспортних засобів. Передача цих даних через API перевізника до серверної частини системи створює технічну основу для побудови інтегрованих рішень на базі вебтехнологій. Застосування Node.js для серверної логіки, MySQL для

зберігання даних та React із бібліотекою Leaflet для створення клієнтського інтерфейсу дозволяє забезпечити масштабованість, швидкодію та зручність користування системою.

Впровадження подібних систем є ключовим етапом у розвитку концепції «розумного міста» (Smart City), де інформаційні технології використовуються для підвищення ефективності управління міськими ресурсами, покращення мобільності населення та зниження екологічного навантаження.

Таким чином, актуальність предметної області полягає у необхідності створення сучасних, надійних і доступних інформаційних систем контролю за переміщенням міського пасажирського транспорту, які забезпечують комплексне вирішення завдань моніторингу, аналізу та управління транспортними процесами.

Подальший розвиток транспортної інфраструктури міста неможливий без впровадження інтелектуальних транспортних систем (ITC), що поєднують апаратні засоби моніторингу з програмними компонентами аналізу даних. Інформаційні системи цього типу дозволяють автоматизувати процеси керування транспортом, підвищити оперативність реагування на зміни дорожньої ситуації та оптимізувати використання транспортних засобів.

Важливою перевагою впровадження таких систем є можливість накопичення історичних даних, які можна використовувати для аналітики та прогнозування. Аналіз часових рядів даних GPS дає змогу виявляти закономірності руху транспорту, оцінювати рівень дотримання розкладу, виявляти перевантажені маршрути та зони затримок. Це створює передумови для подальшого застосування методів машинного навчання у прогнозуванні транспортних потоків та підвищенні ефективності логістичних рішень [8].

З технічної точки зору, розробка інформаційної системи контролю за переміщенням транспорту потребує використання сучасних вебтехнологій і гнучкої архітектури. Застосування Node.js як середовища серверної логіки забезпечує високу швидкодію при обробці великої кількості одночасних запитів, що є критичним для систем реального часу. Використання MySQL як

основної системи керування базами даних гарантує структуроване зберігання даних про координати, маршрути та параметри руху. На клієнтському рівні React у поєднанні з Leaflet дозволяє реалізувати інтерактивний вебінтерфейс із динамічним оновленням геолокаційної інформації на карті.

Особливу актуальність така система набуває для міських перевізників і органів місцевого самоврядування, оскільки вона дає змогу не лише здійснювати моніторинг, а й отримувати аналітичну інформацію для прийняття управлінських рішень: визначення найефективніших маршрутів, планування графіків руху, контролю дотримання розкладу, а також своєчасного реагування на позаштатні ситуації. Крім того, інформаційні системи моніторингу транспорту є важливим елементом забезпечення прозорості транспортних послуг і підвищення рівня довіри з боку пасажирів. Можливість у реальному часі відстежувати розташування транспортного засобу зменшує час очікування на зупинках, підвищує комфортність користування громадським транспортом і сприяє зростанню лояльності пасажирів до міських перевізників. Таким чином, актуальність предметної області визначається потребою у створенні сучасних, відкритих і масштабованих інформаційних систем, які забезпечують надійний контроль за переміщенням міського пасажирського транспорту, сприяють розвитку концепції «розумного міста» та підвищують ефективність управління транспортною інфраструктурою.

На рис. 1.1 представлено узагальнену структурну схему, що відображає актуальність предметної області розробки інформаційної системи контролю за переміщенням міського пасажирського транспорту. Діаграма демонструє причинно-наслідкові зв’язки між зовнішніми факторами, проблемами міського транспортного середовища, технічними засобами реалізації системи та очікуваними перевагами її впровадження [4].

.

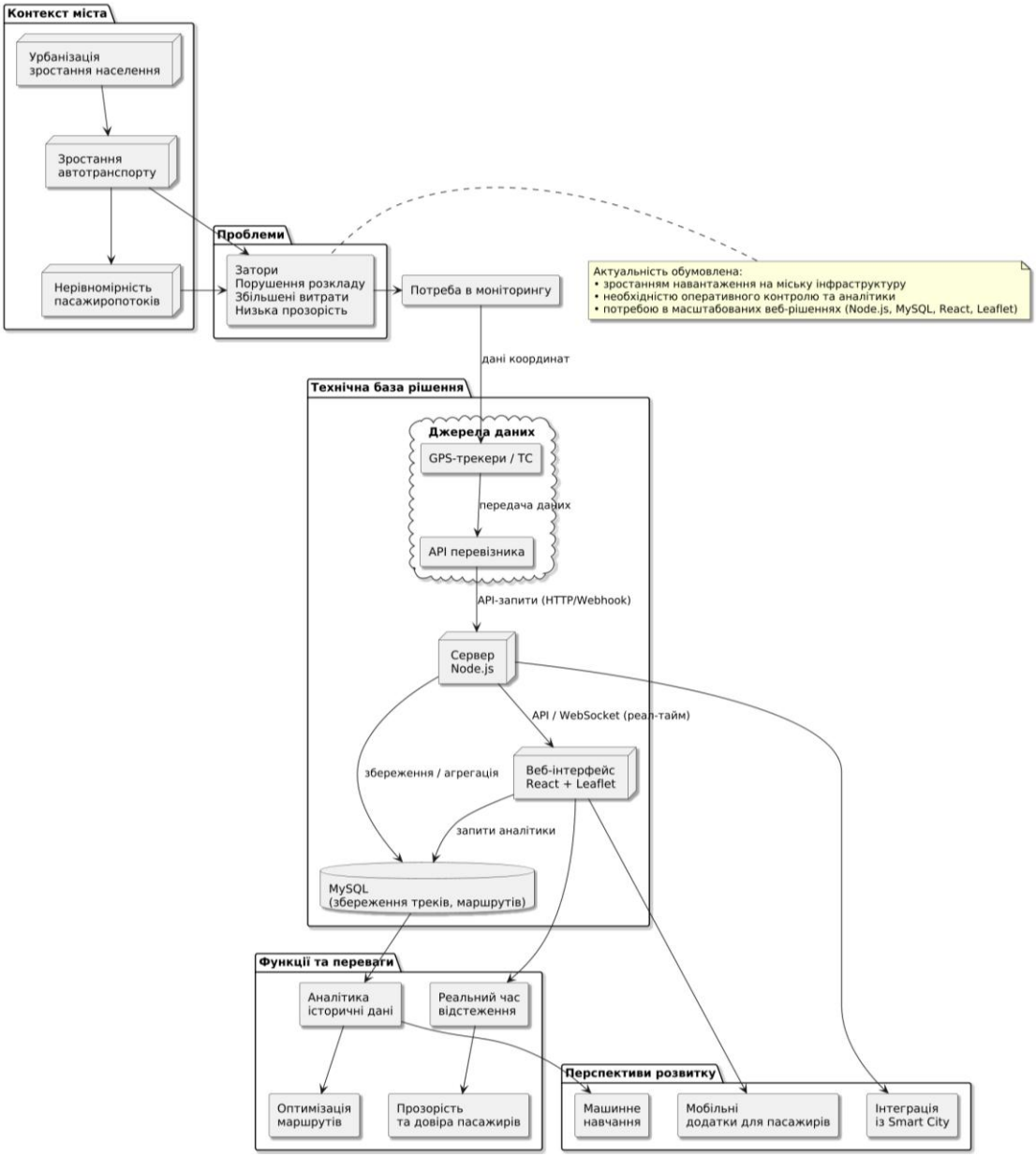

Рисунок 1.1 – Актуальність предметної області

У верхній частині схеми наведено контекст міського середовища, до якого належать такі чинники, як урбанізація та зростання населення, збільшення кількості автотранспорту та нерівномірність пасажиропотоків. Ці фактори створюють підґрунтя для перевантаження транспортної інфраструктури та зниження ефективності пасажирських перевезень.

Як наслідок, формується комплекс проблем, серед яких виділено: утворення заторів, порушення графіків руху, зростання експлуатаційних витрат і низький рівень прозорості роботи перевізників. Усе це обумовлює потребу в моніторингу міського пасажирського транспорту в реальному часі.

Для вирішення зазначених проблем у діаграмі виділено блок «Технічна база рішення», який містить основні компоненти розроблюваної системи [9]:

– GPS-трекери та API перевізника – джерела первинних даних про координати транспортних засобів;

– Серверна частина на Node.js, яка приймає, обробляє та агрегує дані;

– База даних MySQL для збереження інформації про маршрути, треки та параметри руху;

– Веб-інтерфейс на React із використанням бібліотеки Leaflet, який забезпечує візуалізацію транспортних засобів на карті та доступ користувачів до аналітичної інформації.

Зазначені компоненти формують логічну архітектуру системи, у якій дані передаються за ланцюгом: GPS-трекери → API перевізника → сервер (Node.js) → база даних (MySQL) → веб-інтерфейс (React + Leaflet).

У нижній частині діаграми наведено блок «Функції та переваги», який відображає результати впровадження системи, а саме: можливість відстеження транспорту в реальному часі, накопичення історичних даних для аналітики, оптимізацію маршрутів руху та підвищення прозорості перевізників, що позитивно впливає на рівень довіри пасажирів.

Окремо виділено блок «Перспективи розвитку», що охоплює потенційні напрями розширення системи: інтеграцію технологій машинного навчання для прогнозування пасажиропотоків, розробку мобільних застосунків для пасажирів та інтеграцію з елементами концепції «Smart City».

Таким чином, представлена діаграма комплексно відображає взаємозв’язок між факторами, що зумовлюють потребу у створенні системи моніторингу, технічною архітектурою рішення та очікуваними результатами її впровадження. Це підтверджує актуальність предметної області, оскільки інформаційні системи такого типу є необхідним інструментом для підвищення ефективності управління міським пасажирським транспортом і розвитку цифрової інфраструктури міста.

1.2 Аналіз існуючих системи контролю міського транспорту

У сучасних умовах розвитку цифрових технологій системи контролю міського пасажирського транспорту є невід’ємною складовою ефективного управління міською інфраструктурою. Їх основна мета полягає у забезпеченні моніторингу транспортних засобів у реальному часі, збиранні статистичних даних, контролі дотримання розкладу та підвищенні якості перевезень для пасажирів.

Для аналізу предметної області доцільно розглянути існуючі рішення, що вже застосовуються в різних містах України та світу.

Одними з найпоширеніших типів таких систем є GPS/ГЛОНАСС-платформи моніторингу транспорту, які дозволяють відстежувати місцеположення транспортних засобів, швидкість руху, дотримання маршруту та стоянки. Серед відомих комерційних рішень можна виділити такі платформи, як Wialon, EasyTrack, АвтоГраф, SkyTrack, UkrTracking тощо. Ці системи надають базові інструменти контролю за транспортом через веб-інтерфейс або мобільний додаток, використовуючи супутникові технології навігації та передачу даних через GPRS або GSM-мережі [1].

Платформа Wialon є однією з найрозповсюдженіших у Європі та Україні. Вона підтримує підключення понад 300 типів GPS-трекерів, надає засоби для побудови звітів, аналітики, контролю за витратами пального та виявлення відхилень від маршрутів. Основний недолік цього рішення полягає у його закритості, високій вартості ліцензій та складності інтеграції з іншими інформаційними системами.

Система EasyTrack орієнтована на комерційні автопарки та дозволяє не лише спостерігати за транспортом, але й керувати логістикою перевезень. Однак, для задач міського пасажирського транспорту її функціональність є обмеженою, оскільки вона не враховує специфіку громадських маршрутів, розкладів та взаємодію з пасажирами.

Вітчизняні рішення, такі як UkrTracking або АвтоГраф, мають подібний

набір можливостей, проте часто не забезпечують достатньої масштабованості та стабільності при роботі з великим обсягом даних, характерним для систем громадського транспорту. Їх інтерфейси зазвичай мають обмежену гнучкість і не передбачають інтеграції з сучасними веб-технологіями, такими як React або Leaflet [2].

Окремо слід відзначити рішення, які впроваджуються муніципалітетами великих міст, зокрема «Kyiv Smart City», «Львівська система GPS-моніторингу» та подібні локальні проекти. Їх перевагою є відкритість для громадян – користувачі можуть у режимі реального часу переглядати місцеположення автобусів і тролейбусів через офіційні веб-сайти або мобільні додатки. Водночас такі рішення часто базуються на комерційних API, мають обмежену функціональність для внутрішнього використання перевізниками та не завжди забезпечують достатню точність геолокаційних даних.

Dozor City – це український вебсервіс та мобільний додаток, призначений для моніторингу руху міського громадського транспорту в режимі реального часу. Система надає користувачам можливість отримувати актуальну інформацію про місцеположення транспортних засобів, маршрути та час прибуття на зупинки, що сприяє підвищенню зручності користування громадським транспортом.

Основні цілі та функціональні можливості системи Dozor City [1]:

1. Забезпечення пасажирів інструментом для спостереження за рухом міського транспорту з точним відображенням його поточного місцеположення на карті;

2. Надання інформації про орієнтовний час прибуття транспортного засобу на вибрану зупинку, що дозволяє ефективніше планувати поїздки;

3. Можливість перегляду маршрутів і зупинок на інтерактивній карті з фільтрацією за видами транспорту (автобус, тролейбус, трамвай тощо);

4. Використання GPS-даних від обладнання, встановленого на транспортних засобах, для підвищення точності, оперативності та достовірності інформації.

На таблиці 1.1 представлено основні переваги та недоліки використання вебсервісу Dozor City як прикладу сучасної інформаційної системи моніторингу міського транспорту.

Таблиця 1.1 – Переваги та недоліки системи Dozor City

№	Критерій	Характеристика
1	Переваги	локалізація сервісу для українських міст; інтуїтивно зрозумілий інтерфейс та зручна навігація; доступ до реальних даних у режимі онлайн.
2	Недоліки	обмежена кількість населених пунктів, у яких реалізовано можливість відстеження транспорту; відсутність розширеного функціоналу для користувачів із роллю водіїв або представників транспортних компаній (зокрема, функцій управління маршрутами чи зворотного зв’язку).

Як видно з табл. 1.1, система Dozor City має низку переваг, що роблять її ефективним інструментом для пасажирів – зокрема простоту використання, локальну адаптацію та доступ до актуальних даних у режимі реального часу. Водночас певні обмеження функціональності, такі як відсутність інтегрованих модулів для перевізників та обмежене географічне покриття, знижують потенціал системи для комплексного управління транспортною мережею.

Завдяки переліченим характеристикам Dozor City можна охарактеризувати як зручний і практичний інструмент для пасажирів, що підвищує прозорість транспортних перевезень і сприяє цифровізації міської інфраструктури.

На рис. 1.2 представлена головна сторінка інтерфейсу інтерактивної карти вебсервісу Dozor City, яка відображає поточне положення транспортних засобів на маршрутах.

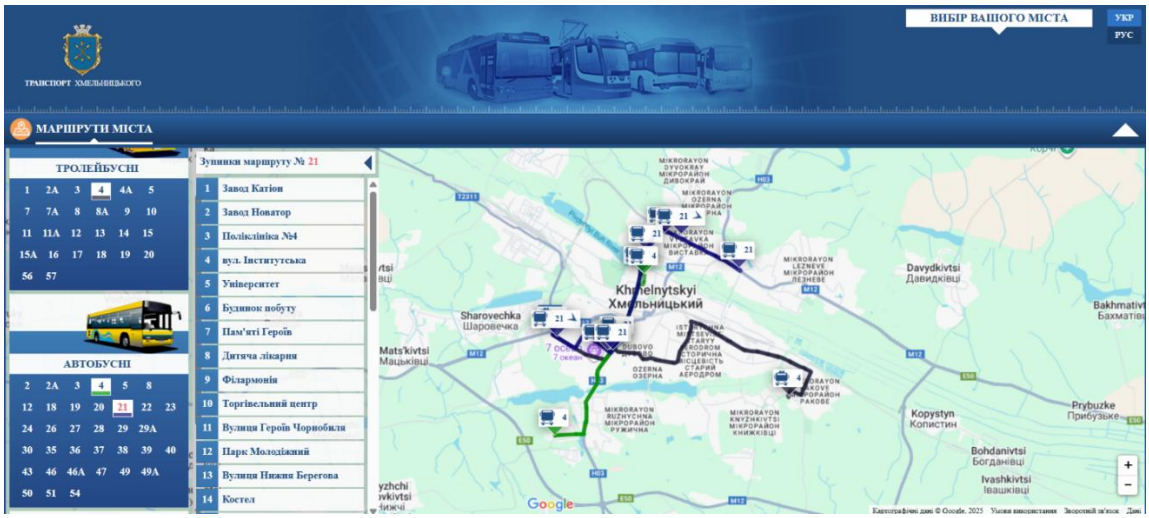

Рисунок 1.2 – Інтерфейс веб-сервісу Dozor City

Проведений аналіз дозволяє зробити висновок, що більшість існуючих систем моніторингу міського транспорту:

– орієнтовані переважно на відображення геолокації транспортних засобів без глибокої аналітики;

– мають закриту архітектуру, що ускладнює інтеграцію з іншими сервісами міської інфраструктури;

– використовують застарілі технологічні стеки або обмежені інтерфейси;

– не надають достатніх можливостей для масштабування та адаптації під потреби конкретного міста.

Таким чином, сервіс Dozor City забезпечує базові функції моніторингу міського пасажирського транспорту в режимі реального часу, що дозволяє користувачам відстежувати місцезнаходження транспортних засобів, переглядати маршрути та отримувати інформацію про орієнтовний час прибуття. Завдяки використанню GPS-технологій та інтерактивної карти система сприяє підвищенню прозорості роботи громадського транспорту та зручності для пасажирів. Водночас обмежене географічне покриття та відсутність розширених можливостей для перевізників і водіїв свідчать про потенціал подальшого розвитку та вдосконалення подібних рішень у напрямі створення повнофункціональних інформаційних систем контролю транспорту.

EasyWay – це інформаційний вебсервіс і мобільний додаток, призначений

для пошуку оптимальних маршрутів громадського транспорту та перегляду розкладу руху з можливістю відображення поточного місця транспорту в окремих містах України. Основна мета системи полягає у спрощенні процесу планування поїздок пасажирами та підвищенні ефективності використання міського транспорту.

Основні цілі та функціональні можливості системи EasyWay [1]:

– забезпечення користувачів зручним інструментом для відстеження руху громадського транспорту у містах України;

– відображення інформації про поточне місцеположення транспортних засобів, прогнозований час прибуття та маршрути в інтерактивному інтерфейсі;

– надання можливості швидкого пошуку потрібного маршруту чи зупинки з відображенням їх розташування на карті;

– оптимізація часу очікування пасажирів завдяки точному прогнозуванню прибуття транспорту, особливо у години пікового навантаження.

Переваги системи: простота використання та зрозумілий інтерфейс; інтеграція з картою міста для зручної навігації; можливість перегляду розкладу руху та поточного місцеположення транспортних засобів.

Недоліки системи: функція відстеження транспорту доступна не для всіх міст України; залежність точності даних від своєчасності оновлення інформації перевізниками.

Таким чином, сервіс EasyWay забезпечує базові функції моніторингу громадського транспорту та зручний пошук маршрутів, проте його можливості залишаються обмеженими у порівнянні з більш спеціалізованими рішеннями.

На рис. 1.3 представлено приклад інтерфейсу веб-сервісу EasyWay, який демонструє відображення маршрутів і поточного місця розташування транспортних засобів на карті.

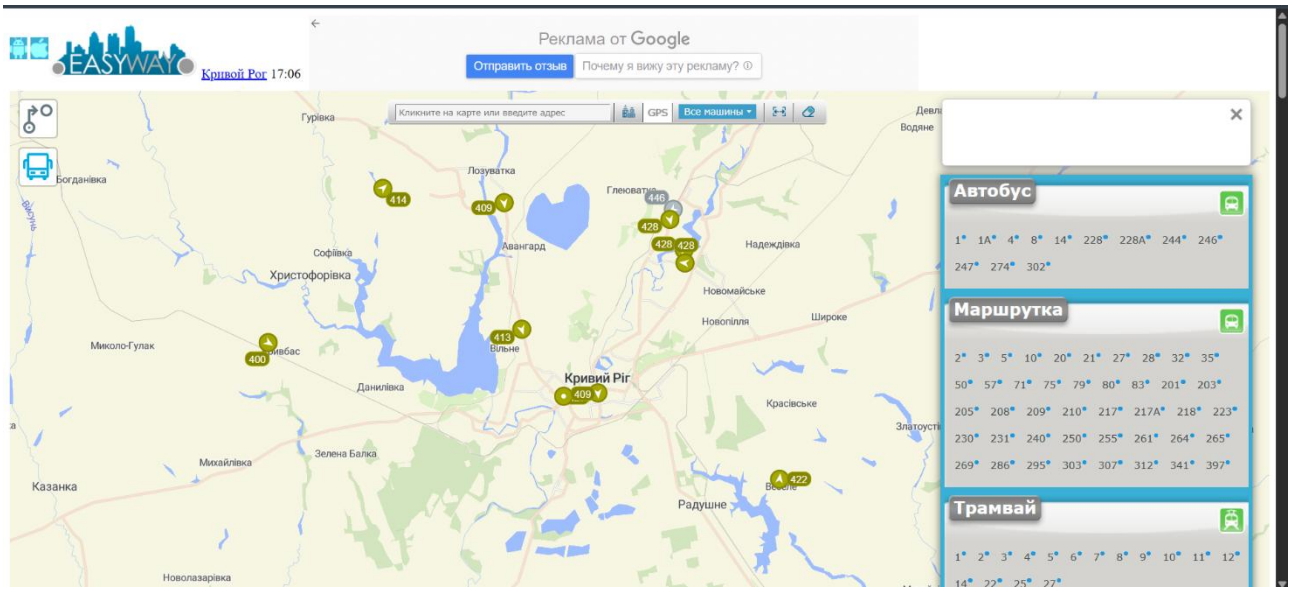

Рисунок 1.3 – Інтерфейс веб-сервісу EasyWay

City Transport – це український мобільний додаток, призначений для відображення даних GPS-трекінгу міського громадського транспорту в режимі реального часу. Система надає користувачам актуальну інформацію про маршрути, розклад руху та поточне місцеположення транспортних засобів, що сприяє підвищенню ефективності планування поїздок та зменшенню часу очікування. Основною метою розробки додатку є забезпечення пасажирів зручним мобільним інструментом для моніторингу руху транспорту у містах України. Додаток дозволяє переглядати прогнозований час прибуття, визначати оптимальні маршрути та відображати їх розташування на інтерактивній карті з високим рівнем точності.

Функціональні можливості системи спрямовані на підвищення комфорту користувачів за рахунок оперативного доступу до живих GPS-даних та інтуїтивно зрозумілого інтерфейсу. Завдяки локалізації для українських міст і простоті навігації застосунок має високий рівень доступності для широкого кола користувачів. До переваг City Transport належать адаптація до українських умов, відкритий і зручний інтерфейс, а також відображення даних у режимі реального часу. Разом із тим, недоліками системи є недостатня деталізація інформації для малих міст та залежність точності від регулярності оновлення даних, що надходять від транспортних операторів [2].

Таким чином, City Transport є сучасним прикладом мобільного рішення для моніторингу міського транспорту, що поєднує функціональність, простоту використання та високу інформативність.

На рис. 1.4 подано приклад інтерфейсу мобільного додатку City Transport, який демонструє відображення маршрутів та поточного положення транспортних засобів на інтерактивній карті.

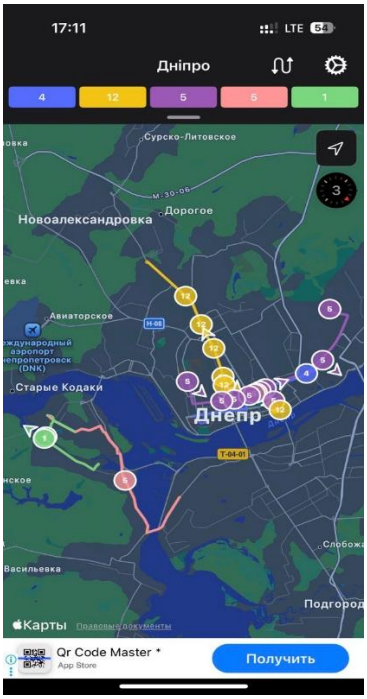

Рисунок 1.4 – Інтерфейс мапи відображення транспорту мобільного додатку City Transport

Moovit – це міжнародна навігаційна платформа громадського транспорту, яка поєднує інформацію про маршрути, розклади руху та дані з оновленнями в режимі реального часу. Система забезпечує користувачів можливістю планування поїздок у багатомодальному форматі, включаючи різні види транспорту – автобус, метро, трамвай, велосипед, спільні автомобілі та електросамокати. Для підвищення точності прогнозів використовуються GPS-дані транспортних засобів у поєднанні з офіційною інформацією перевізників та краудсорсинговими оновленнями від користувачів.

Платформа Moovit відзначається високим рівнем інтеграції та зручністю користування. Вона надає можливість відображення очікуваного часу прибуття

транспорту та його поточного місцеположення в реальному часі. Додатково користувачі отримують сповіщення про зміни у русі, затримки, аварії та інші позаштатні ситуації, що дозволяє оперативно реагувати на зміни транспортної ситуації. У деяких містах сервіс підтримує функцію мобільних платежів, яка забезпечує придбання та валідацію електронних квитків безпосередньо через додаток.

Особливістю Moovit є режим «живої» навігації, що реалізується через функцію Way Finder із використанням технологій доповненої реальності (AR). Такий підхід дозволяє користувачам орієнтуватися у просторі під час поїздок і знаходити необхідні зупинки або пересадки. Крім того, платформа активно залучає користувачів до участі в оновленні та уточненні даних – наприклад, через повідомлення про переповненість транспорту, наявність Wi-Fi, стан станцій тощо. Це сприяє підвищенню достовірності інформації та формуванню активної спільноти користувачів.

Серед основних переваг Moovit слід відзначити широке географічне покриття, постійне оновлення даних на основі GPS-моніторингу, а також доступність як у форматі мобільного додатку, так і у вебверсії. Водночас існують певні недоліки, зокрема обмеження у доступності функції реального трекінгу в окремих регіонах, а також наявність реклами чи платних розширених функцій [3].

Таким чином, Moovit є потужним інструментом для навігації громадським транспортом, який об’єднує точні навігаційні алгоритми, інтегровані сервіси оплати та елементи взаємодії користувачів, забезпечуючи високий рівень сервісу у сфері «розумної мобільності».

На рис. 1.5 наведено приклад інтерфейсу мобільного додатку Moovit, що демонструє інтерактивну карту з відображенням маршрутів і прогнозованого часу прибуття транспортних засобів.

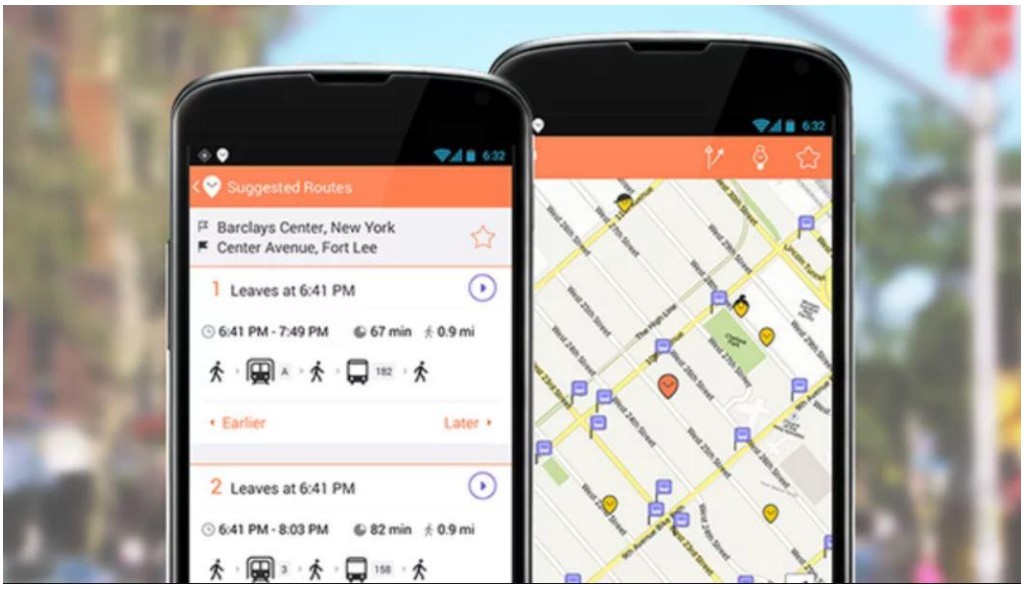

Рисунок 1.5 – Інтерфейс мобільного додатку Moovit

Transit – це сучасний мобільний додаток та веб-сервіс, призначений для надання користувачам актуальної інформації про переміщення міського громадського транспорту, можливі затримки, зміни маршрутів та альтернативні варіанти поїздок. Система спрямована на забезпечення зручності планування пересування в межах міста, пропонуючи точні та своєчасні дані про транспортні засоби у режимі реального часу [4].

Основним призначенням Transit є надання користувачам доступу до оперативних даних щодо руху транспорту – автобусів, трамваїв, тролейбусів та поїздів метро. Після відкриття додатку користувач одразу бачить найближчі маршрути й зупинки, що значно спрощує процес планування поїздки без необхідності попереднього введення запиту. Однією з особливостей системи є використання краудсорсингового підходу: користувачі можуть ділитися актуальною інформацією про місцезнаходження транспортних засобів, що підвищує точність та оперативність даних.

Додаток також підтримує функцію сповіщень про збої у русі, зміну маршрутів або затримки через push-нотифікації, особливо для ліній, позначених користувачем як улюблені. Важливою перевагою Transit є можливість роботи в офлайн-режимі: додаток зберігає розклади, мапи та інформацію про зупинки, що дозволяє користувачам орієнтуватися навіть за відсутності інтернет-

з’єднання.

Крім того, система має інтеграцію з альтернативними видами транспорту, такими як каршеринг, служби таксі (Uber, Lyft), міські велосипеди та електросамокати, що сприяє розвитку концепції багатомодальної мобільності. Завдяки цьому користувач може отримати цілісну картину транспортної мережі міста та обрати найзручніший спосіб пересування.

Серед ключових переваг Transit варто відзначити високу точність визначення місцеположення транспортних засобів за допомогою GPS, простоту інтерфейсу та швидкість пошуку необхідних маршрутів. Водночас певні обмеження зумовлені залежністю сервісу від співпраці з міськими адміністраціями, а також наявністю додаткових функцій, що доступні лише у платній версії додатку.

Отже, Transit є ефективним інструментом для оперативного моніторингу та планування поїздок міським транспортом, який поєднує сучасні технології навігації, краудсорсинг даних і підтримку офлайн-доступу, забезпечуючи високий рівень зручності для пасажирів.

На рис. 1.6 наведено інтерфейс мобільного додатку Transit, що демонструє інтерактивну карту з відображенням маршрутів, зупинок і прогнозованого часу прибуття транспорту.

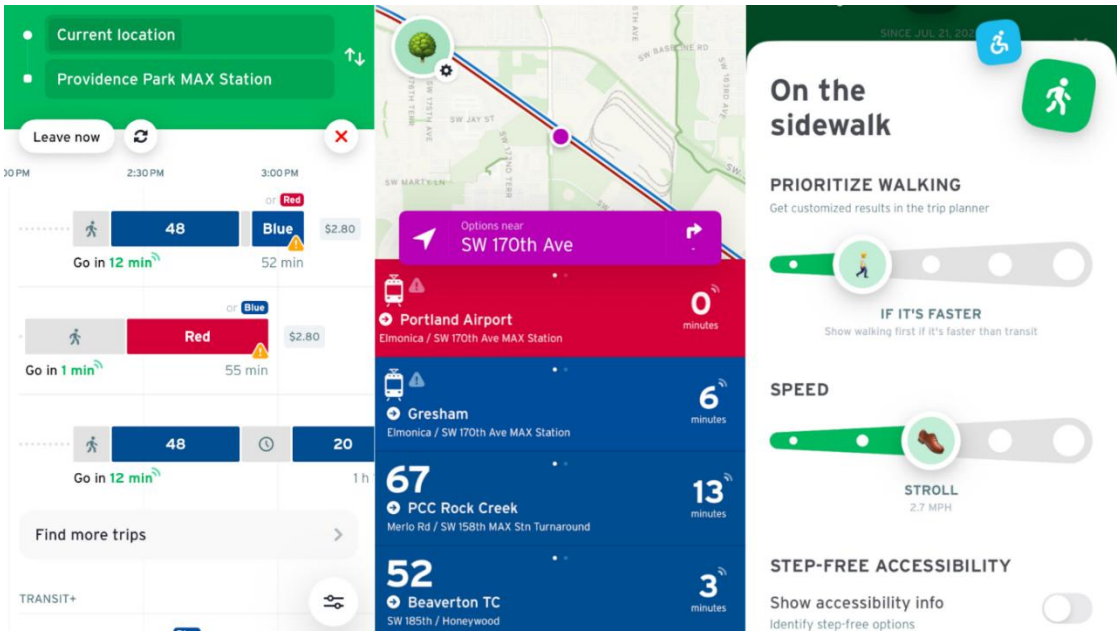

Рисунок 1.6 – Інтерфейс мобільного додатку Transit

У результаті проведеного аналізу наявних програмних рішень у сфері інформаційних систем управління та контролю за переміщенням міського пасажирського транспорту було сформовано уявлення про їхні основні функціональні можливості, архітектурні підходи та обмеження. Це дозволило виокремити ключові вимоги – як функціональні, так і нефункціональні – що стали основою для подальшого проєктування та розроблення запропонованої в межах магістерської роботи інформаційної системи [1].

1.3 Аналіз технологій для розробки сучасних web-платформ для міського транспорту

Розробка інформаційних систем моніторингу міського пасажирського транспорту потребує вибору оптимального технологічного стеку, який забезпечує стабільну роботу, масштабованість та високу швидкість оброблення даних у режимі реального часу. Основними критеріями при виборі технологій є продуктивність, зручність інтеграції з зовнішніми сервісами, безпека, а також сумісність між серверною та клієнтською частинами.

На серверному рівні доцільним є використання Node.js, який завдяки подієво-орієнтованій архітектурі та асинхронній моделі оброблення запитів забезпечує ефективну роботу з великою кількістю одночасних підключень. Це особливо важливо для систем, що приймають потік GPS-даних у реальному часі від численних транспортних засобів. Перевагою Node.js є також наявність широкої екосистеми бібліотек і фреймворків (зокрема, Express.js), які дозволяють швидко реалізувати REST API для обміну інформацією між клієнтським інтерфейсом та базою даних.

Для зберігання, оброблення та структурування інформації доцільно застосовувати MySQL – реляційну систему керування базами даних, що характеризується високою стабільністю, швидкістю виконання запитів і підтримкою транзакцій. У контексті системи контролю транспорту MySQL використовується для зберігання даних про транспортні засоби, маршрути,

координати GPS, час руху та статистичні показники. Перевагою MySQL є підтримка складних SQL-запитів і можливість оптимізації таблиць для роботи з великими обсягами даних [4].

Клієнтська частина системи реалізується з використанням бібліотеки React, яка дозволяє створювати динамічні та адаптивні інтерфейси користувача. React забезпечує високу продуктивність завдяки віртуальному DOM і компонентному підходу до побудови інтерфейсу. Це спрощує оновлення даних на сторінці без необхідності повного перезавантаження, що є критично важливим для систем із відображенням руху транспорту у реальному часі.

Для візуалізації географічних даних використовується бібліотека Leaflet, яка є відкритим JavaScript-інструментом для побудови інтерактивних карт. Вона забезпечує інтеграцію з різними картографічними сервісами (OpenStreetMap, Mapbox тощо) та дозволяє відображати маркери транспортних засобів, маршрути, зупинки і зони обслуговування в зручному графічному вигляді.

Передача даних між GPS-трекерами та сервером здійснюється через API перевізників, які формують потоки координат та додаткових параметрів (швидкість, напрямок, стан транспорту). Ці дані обробляються сервером у режимі реального часу та передаються на клієнтську частину через WebSocket-з’єднання або REST API [5].

З точки зору архітектури, обраний стек технологій (Node.js + MySQL + React + Leaflet) забезпечує гнучкість, масштабованість і кросплатформність розроблюваної системи. Крім того, він відповідає сучасним тенденціям створення web-рішень у сфері «розумного транспорту» (Smart Mobility), де важливу роль відіграють інтеграція з зовнішніми джерелами даних, швидка реакція на зміни транспортної ситуації та зручний інтерфейс користувача.

На рис. 1.7 представлено узагальнену схему, що демонструє, як різні технології взаємодіють між собою у процесі створення сучасної веб-платформи для моніторингу міського пасажирського транспорту. На нижньому рівні відбувається збір даних із GPS-трекерів, установлених на транспортних засобах, які через API перевізника передають координати та супровідну інформацію

(швидкість, напрямок руху, час оновлення). Отримані дані надходять на серверну частину, реалізовану на Node.js, де вони проходять оброблення, фільтрацію та запис у базу даних MySQL [7].

Далі оброблена інформація через REST API або WebSocket-з'єднання передається на клієнтську частину системи, створену з використанням React, яка забезпечує інтерактивний веб-інтерфейс для користувачів. Візуалізація географічних даних здійснюється за допомогою бібліотеки Leaflet, що дозволяє відображати маршрути, зупинки та поточні координати транспорту на інтерактивній карті (рис. 1.7).

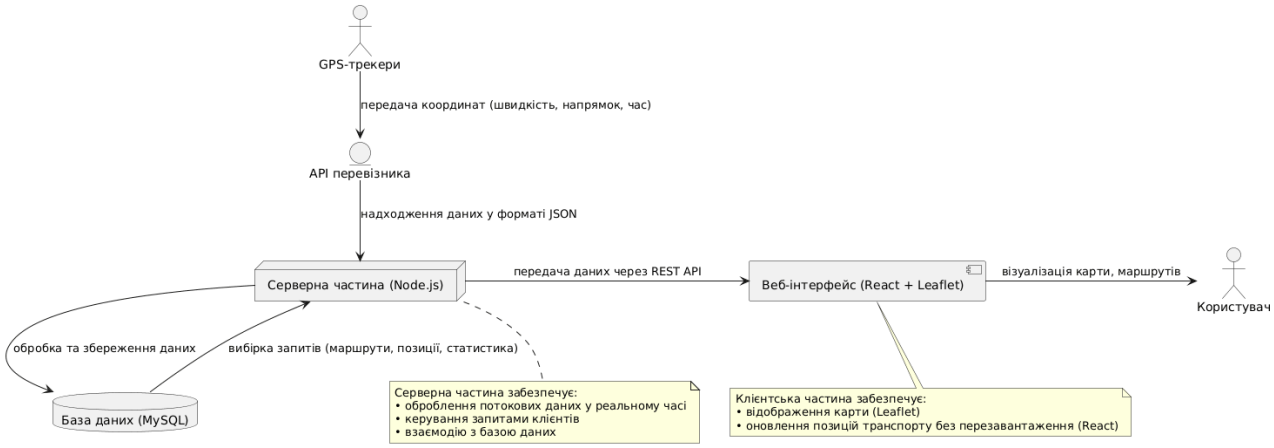

Рисунок 1.7 – Взаємодія технологій у веб-платформі моніторингу міського транспорту

Така архітектура забезпечує ефективну взаємодію між усіма компонентами системи – від отримання польових даних до їх графічного подання в режимі реального часу. Завдяки цьому веб-платформа здатна підтримувати безперервне оновлення інформації, високу швидкодію та масштабованість, що є ключовими вимогами до сучасних інформаційних систем міського транспорту.

На рис. 1.8 представлено узагальнену модель технологій, які використовуються для створення сучасної веб-платформи моніторингу міського пасажирського транспорту. Модель демонструє взаємозв'язок між рівнями системи – від збору даних GPS-трекерами до їх оброблення на сервері Node.js, зберігання у базі MySQL та подальшої візуалізації на клієнтській частині за допомогою React і Leaflet. Така багаторівнева архітектура забезпечує

високу масштабованість, продуктивність і зручність взаємодії користувача з системою.

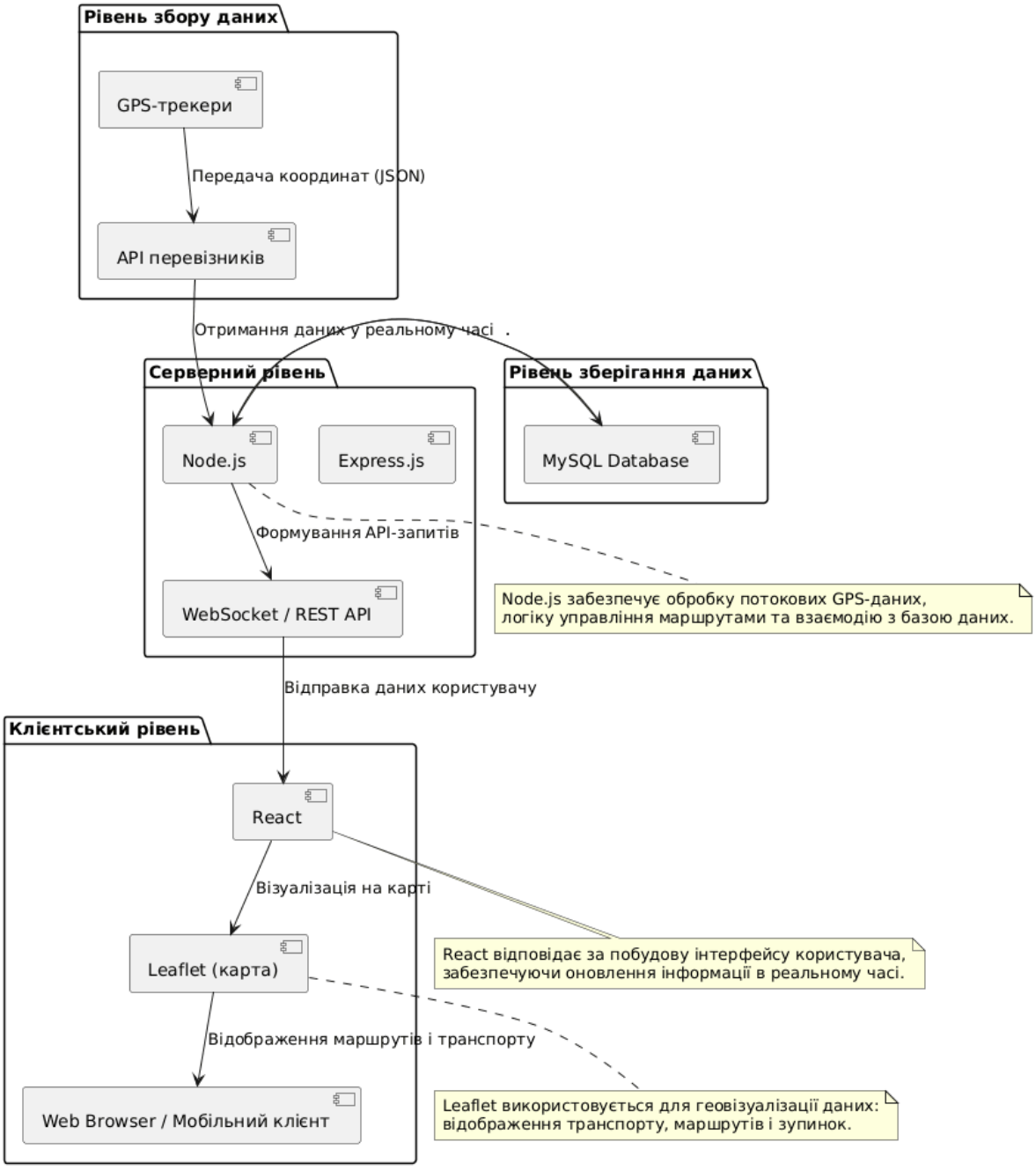

Рисунок 1.8 – Технології для створення сучасної веб-платформи для міського транспорту

1.4 Постановка задачі досліджень

Сучасні міста стикаються з численними викликами у сфері організації громадського транспорту, серед яких виділяються проблеми непередбачуваності прибуття транспортних засобів, відсутності оперативної

інформації про їх місцезнаходження та складності планування маршрутів пересування для пасажирів. Ці фактори призводять до зниження ефективності використання громадського транспорту, збільшення часу очікування на зупинках та загального погіршення якості транспортного обслуговування населення [9].

Метою даного дослідження є розробка комплексної інформаційної системи контролю за переміщенням міського пасажирського транспорту, яка забезпечить можливість моніторингу в режимі реального часу, планування оптимальних маршрутів пересування та надання актуальної інформації про рух транспортних засобів. Система має забезпечити інтеграцію даних про маршрути, зупинки, транспортні засоби та їх поточне місцезнаходження в єдиному інформаційному просторі з можливістю доступу через веб-інтерфейс.

Для досягнення поставленої мети необхідно вирішити наступний комплекс взаємопов'язаних задач. По-перше, необхідно створити архітектуру системи, яка забезпечить ефективну взаємодію між клієнтською та серверною частинами, базою даних та зовнішніми сервісами. По-друге, потрібно розробити механізм збору, обробки та зберігання геопросторових даних про переміщення транспортних засобів з урахуванням необхідності забезпечення високої частоти оновлення інформації. По-третє, система повинна надавати користувачам зручний інтерфейс для візуалізації даних на інтерактивній карті з можливістю вибору маршрутів, фільтрації транспортних засобів за типами та планування пересувань між довільними точками міста [10].

Особлива увага приділяється забезпеченню масштабованості системи, що дозволить обробляти великі обсяги даних від численних транспортних засобів одночасно. Система має підтримувати одночасну роботу значної кількості користувачів без втрати продуктивності та забезпечувати оновлення даних про місцезнаходження транспорту з інтервалом, достатнім для прийняття оперативних рішень пасажирами.

Важливим аспектом дослідження є вибір технологічного стеку, який забезпечить ефективну реалізацію всіх компонентів системи. Для клієнтської

частини доцільним є використання сучасних JavaScript фреймворків, які надають можливості створення динамічних односторінкових додатків з високою інтерактивністю. Серверна частина має забезпечувати ефективну обробку HTTP запитів та підтримку протоколів двостороннього обміну даними для реалізації функціоналу оновлення в режимі реального часу.

Результатом дослідження має стати функціонуюча інформаційна система, яка демонструє можливість практичної реалізації концепції моніторингу міського транспорту з використанням сучасних веб-технологій та геоінформаційних систем. Система повинна підтвердити ефективність обраних архітектурних рішень та технологічних підходів у контексті задач управління громадським транспортом.

Розроблена інформаційна система базується на трирівневій клієнт-серверній архітектурі, яка забезпечує чітке розділення відповідальності між компонентами та полегшує подальший розвиток і масштабування системи. Архітектура включає рівень представлення даних (клієнтська частина), рівень бізнес-логіки (серверна частина) та рівень зберігання даних (система управління базами даних) [7].

Для формального опису архітектури системи моніторингу транспорту було розроблено діаграму компонентів у нотації UML (Unified Modeling Language), яка відображає структурну організацію програмного забезпечення та взаємозв'язки між його складовими елементами. Діаграма компонентів є одним з типів структурних діаграм мови UML і призначена для моделювання фізичної архітектури системи з акцентом на компоненти програмного забезпечення та їх залежності.

Запропонована діаграма організована у вигляді трьох основних пакетів, що відповідають рівням системної архітектури. Перший пакет під назвою «Клієнтська частина» об'єднує компоненти, які виконуються в браузері користувача та забезпечують інтерфейс взаємодії з системою. До складу даного пакету входять компонент веб-додатку, реалізований з використанням бібліотеки React.js, компонент інтерактивної карти на базі Leaflet, модуль

планувальника маршрутів з інтеграцією Google Maps API та WebSocket клієнт для забезпечення двостороннього обміну даними.

Другий пакет «Серверна частина» містить компоненти серверного програмного забезпечення, які виконуються на стороні сервера та реалізують бізнес-логіку системи. Центральним елементом цього рівня є компонент API сервера, побудований на фреймворку Express.js, який обробляє вхідні HTTP запити та забезпечує доступ до даних через RESTful інтерфейс. Поряд з ним функціонує WebSocket сервер на базі бібліотеки Socket.IO, що відповідає за підтримку постійних з'єднань для передачі даних в режимі реального часу. Обробник GPS даних здійснює прийом координат від транспортних засобів, їх валідацію та підготовку до збереження. Менеджер маршрутів забезпечує управління інформацією про транспортні маршрути та зупинки.

Третій рівень архітектури представлено базою даних MySQL, яка забезпечує персистентне зберігання інформації. На діаграмі база даних зображена у вигляді стереотипу database та включає дві логічні групи таблиць. Перша група об'єднує таблиці routes, stops та route_stops, які зберігають інформацію про маршрути, зупинки та їх взаємозв'язки. Друга група включає таблиці vehicles та gps_data для зберігання даних про транспортні засоби та історії їх переміщень [11].

Окремим елементом діаграми є пакет «Зовнішні сервіси», який представляє компоненти, що знаходяться поза межами розробленої системи, але інтегровані з нею для розширення функціональності. До складу цього пакету входять Google Maps API, який надає картографічні дані та сервіси геокодування, а також GPS трекери транспортних засобів, які є джерелом первинних даних про координати.

Взаємодія між компонентами відображена на діаграмі за допомогою різних типів з'єднань. Суцільні лінії з стрілками позначають прямі залежності та виклики між компонентами в межах одного рівня архітектури. Так, компонент веб-додатку безпосередньо використовує компоненти інтерактивної карти та планувальника для відображення інформації користувачу. Пунктирні лінії з

стрілками вказують на комунікацію між різними рівнями архітектури через мережеві протоколи. Зокрема, веб-додаток взаємодіє з API сервером через HTTP протокол для отримання статичних даних, а WebSocket клієнт підтримує постійне з'єднання з WebSocket сервером для отримання оновлень в режимі реального часу [12].

Особливу увагу на діаграмі приділено відображенню інформаційних потоків. API сервер здійснює запити до менеджера маршрутів для отримання необхідних даних, який у свою чергу виконує SQL запити до бази даних. WebSocket сервер отримує оновлення від обробника GPS даних, який безперервно приймає координати від GPS трекерів транспортних засобів та зберігає їх в базі даних. Планувальник маршрутів на клієнтській стороні здійснює API запити до зовнішнього сервісу Google Maps для побудови оптимальних шляхів пересування між точками на карті.

Додаткові примітки на діаграмі надають контекстну інформацію щодо призначення окремих компонентів та особливостей їх функціонування. Зокрема, зазначено, що WebSocket сервер здійснює передачу оновлень з інтервалом п'ять секунд, що забезпечує достатню актуальність інформації без створення надмірного навантаження на мережеву інфраструктуру. Примітка до компонента веб-додатку деталізує його основні функції, включаючи вибір маршрутів, фільтрацію транспортних засобів за типами та відображення статистичної інформації. Опис бази даних уточнює структуру збережених даних, включаючи маршрути та зупинки, інформацію про транспортні засоби, історію їх переміщень та поточні GPS координати [16].

Представлена діаграма компонентів демонструє високий рівень модульності архітектури, що забезпечує можливість незалежної розробки та тестування окремих компонентів системи. Чітке розділення на рівні дозволяє локалізувати зміни в одному з них без необхідності модифікації інших частин системи. Використання стандартизованих протоколів взаємодії HTTP та WebSocket гарантує сумісність з широким спектром клієнтських пристроїв та спрощує можливе розширення системи додатковими клієнтськими додатками,

включаючи мобільні застосунки.

Архітектурне рішення щодо використання окремого обробника GPS даних забезпечує можливість масштабування системи шляхом розгортання множини екземплярів цього компонента для паралельної обробки даних від великої кількості транспортних засобів. Застосування WebSocket технології для передачі оновлень дозволяє значно знизити затримки в доставці інформації порівняно з традиційними підходами на базі періодичних HTTP запитів, що критично важливо для забезпечення актуальності даних в системі моніторингу реального часу [13].

Клієнтська частина системи базується на компонентному підході, притаманному бібліотеці React.js, що забезпечує високу ступінь повторного використання коду та спрощує розробку складного користувацького інтерфейсу. Інтеграція з бібліотекою Leaflet для відображення карт забезпечує ефективну візуалізацію великої кількості об'єктів на карті без втрати продуктивності, що досягається завдяки використанню векторної графіки та оптимізованих алгоритмів рендерингу.

Серверна частина побудована на платформі Node.js, яка забезпечує високу продуктивність при обробці великої кількості одночасних з'єднань завдяки асинхронній неблокуючій моделі вводу-виводу. Використання фреймворку Express.js дозволяє швидко розробляти RESTful API з підтримкою middleware для реалізації крос-функціональних вимог, таких як аутентифікація, логування та обробка помилок.

Вибір реляційної системи управління базами даних MySQL обґрунтовується необхідністю забезпечення цілісності даних та підтримки складних запитів з об'єднаннями множини таблиць. Структура бази даних спроектована з дотриманням принципів нормалізації, що виключає дублювання інформації та забезпечує ефективне використання дискового простору. Використання індексів на полях, що часто використовуються в умовах запитів, значно підвищує швидкість виконання операцій читання даних.

Таким чином, представлена на UML діаграмі архітектура інформаційної

системи моніторингу транспорту відображає сучасні підходи до проектування розподілених веб-систем з урахуванням вимог масштабованості, продуктивності та надійності. Використання стандартизованої нотації UML забезпечує однозначність трактування архітектурних рішень різними учасниками процесу розробки та полегшує документування системи для подальшого супроводу та розвитку.

Клієнтська частина системи реалізована у вигляді веб-додатку на базі бібліотеки React.js, що забезпечує створення динамічного користувацького інтерфейсу з високою інтерактивністю. Основними компонентами клієнтської частини є модуль інтерактивної карти на базі бібліотеки Leaflet, який відповідає за візуалізацію маршрутів та транспортних засобів, планувальник маршрутів з інтеграцією Google Maps API для побудови оптимальних шляхів пересування, та WebSocket клієнт для забезпечення отримання оновлень в режимі реального часу [15].

Серверна частина побудована на платформі Node.js з використанням фреймворку Express.js, що забезпечує ефективну обробку HTTP запитів та побудову RESTful API. До складу серверної частини входять API сервер, який обробляє запити від клієнтів та забезпечує доступ до даних, WebSocket сервер на базі бібліотеки Socket.IO для підтримки двостороннього обміну даними в реальному часі, обробник GPS даних, який здійснює прийом та обробку координат від транспортних засобів, та менеджер маршрутів, що забезпечує управління інформацією про маршрути та зупинки.

Рівень зберігання даних реалізовано на базі реляційної системи управління базами даних MySQL, яка забезпечує надійне зберігання структурованої інформації та ефективне виконання складних запитів. База даних включає таблиці для зберігання інформації про маршрути, зупинки та їх взаємозв'язки, дані про транспортні засоби та їх прив'язку до маршрутів, а також історичні записи GPS координат з часовими мітками для аналізу переміщень.

На діаграмі, зображеній на рис. 1.9, представлено архітектуру інформаційної системи контролю за переміщенням міського

пасажирського транспорту.

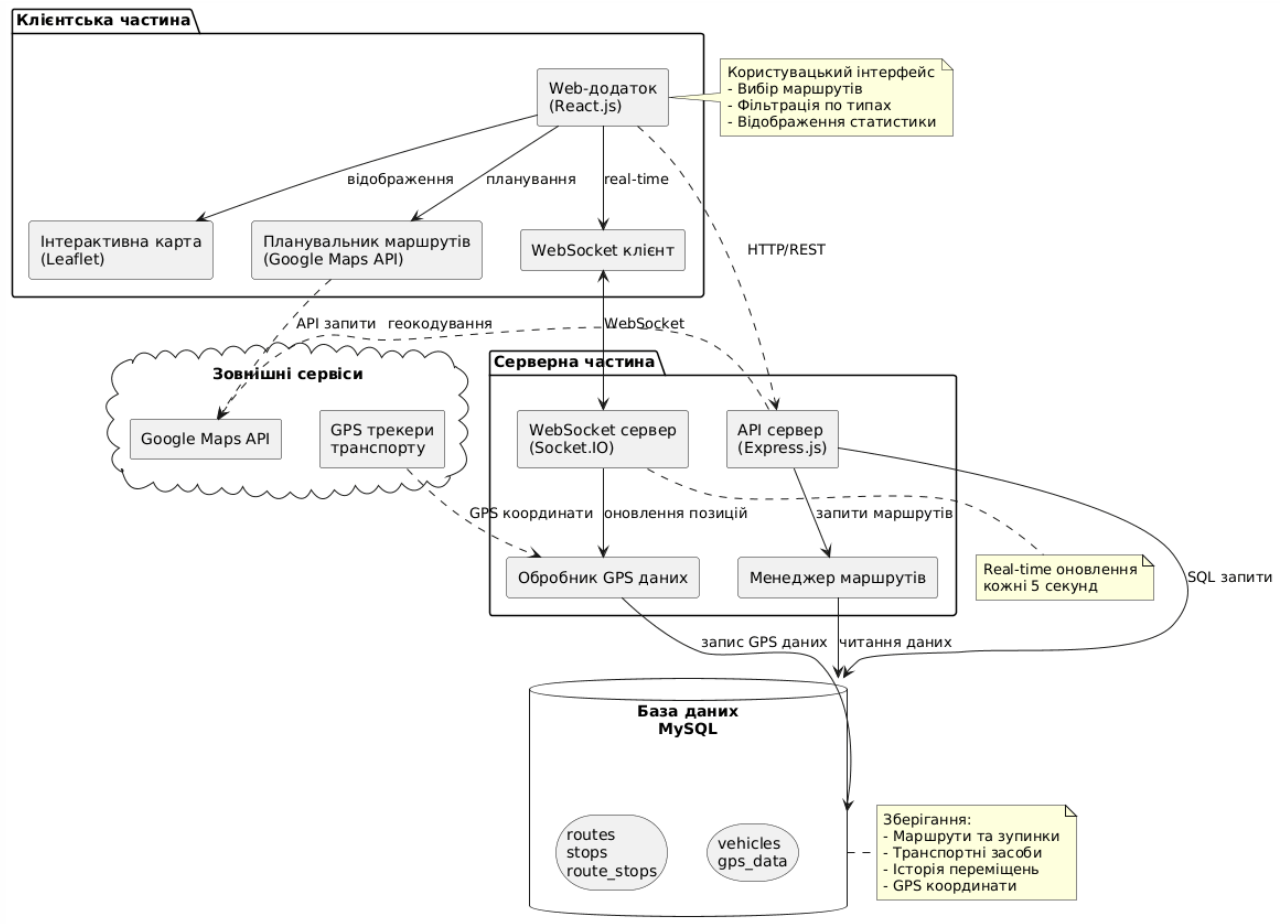

Рисунок 1.9 – Діаграма інформаційної системи контролю за переміщенням міського пасажирського транспорту

Взаємодія між компонентами системи організована наступним чином. Клієнтський додаток здійснює HTTP запити до API сервера для отримання статичних даних про маршрути, зупинки та транспортні засоби. Для забезпечення актуальності інформації про поточне місцезнаходження транспорту використовується WebSocket з'єднання між клієнтом та сервером, яке дозволяє серверу ініціювати передачу оновлених даних без явного запиту від клієнта. GPS трекери, встановлені на транспортних засобах, періодично передають координати до обробника GPS даних на сервері, який здійснює їх валідацію, обробку та збереження в базі даних.

Інтеграція з зовнішніми сервісами здійснюється через Google Maps API, який використовується для геокодування адрес, побудови маршрутів

пересування та надання картографічної інформації. Використання зовнішнього сервісу дозволяє зосередитися на специфічних задачах моніторингу транспорту без необхідності самостійної розробки повноцінної геоінформаційної системи.

Обрана архітектура забезпечує модульність системи, що дозволяє незалежно розвивати окремі компоненти та полегшує тестування. Використання стандартизованих протоколів взаємодії HTTP та WebSocket забезпечує сумісність з широким спектром клієнтських пристроїв та можливість створення мобільних додатків на базі існуючого API. Застосування реляційної бази даних гарантує цілісність інформації та підтримку транзакційності при виконанні операцій модифікації даних [18].

Висновки до розділу:

У першому розділі виконано аналіз існуючих інформаційних систем контролю міського транспорту та програмних рішень, що забезпечують моніторинг руху громадського транспорту у реальному часі. Зокрема, було досліджено можливості, переваги й недоліки таких популярних сервісів, як Dozor City, EasyWay, City Transport, Moovit та Transit. Ці системи надають користувачам доступ до актуальної інформації про маршрути, зупинки, розклади руху транспорту, а також дозволяють відстежувати його місцезнаходження на карті у реальному часі. Їх використання підвищує зручність пересування містом, зменшує час очікування на зупинках і сприяє підвищенню ефективності функціонування транспортної інфраструктури.

Разом із тим, аналіз показав наявність певних недоліків і обмежень зазначених систем. Зокрема, вони характеризуються залежністю від стабільного інтернет-з’єднання, неточністю у відображенні GPS-даних через затримку передачі, відсутністю персоналізованого доступу для перевізників та адміністраторів, а також закритістю програмних інтерфейсів (API), що ускладнює інтеграцію з іншими сервісами.

На основі отриманих результатів запропоновано архітектуру системи, яка забезпечує обробку даних від GPS-трекерів через API перевізника та їх

подальше збереження у базі даних MySQL. Серверна частина системи реалізована на платформі Node.js, що забезпечує асинхронну обробку запитів і швидку взаємодію з базою даних. Клієнтська частина розроблена у вигляді веб-інтерфейсу на основі JavaScript, React та картографічної бібліотеки Leaflet, яка використовується для візуалізації маршрутів руху транспорту та поточного розташування транспортних засобів.

Розроблений підхід демонструє практичну цінність використання сучасних вебтехнологій – Node.js, MySQL, React і Leaflet – для створення гнучких, масштабованих і зручних у використанні інформаційних систем. Запропонована архітектура забезпечує стабільну роботу системи у реальному часі, підтримує інтеграцію з різними джерелами даних і дає змогу ефективно обробляти запити користувачів.

Крім того, було сформульовано вимоги до функціоналу системи, які візуалізовано за допомогою діаграми варіантів використання, побудованої із застосуванням синтаксису PlantUML. Застосування діаграми варіантів використання дало змогу чітко визначити основні сценарії взаємодії користувачів із системою та наочно представити структуру її функціональності, що є основою для подальшого ефективного проєктування та розроблення інформаційної системи контролю міського транспорту.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ КОНТРОЛЮ ЗА ПЕРЕМІЩЕННЯМ МІСЬКОГО ПАСАЖИРСЬКОГО ТРАНСПОРТУ

2.1 Вибір та обґрунтування технологій реалізації web-платформи для міського транспорту

Вибір технологічного стеку для реалізації інформаційної системи моніторингу міського транспорту є критично важливим етапом проєктування, оскільки обрані технології визначають не лише можливості системи на етапі впровадження, а й перспективи її подальшого розвитку, масштабування та інтеграції з іншими інформаційними системами. При формуванні технологічної основи проєкту було проведено комплексний аналіз сучасних технологій веб-розробки, геоінформаційних систем та засобів забезпечення комунікації в режимі реального часу з урахуванням специфічних вимог предметної області.

Для реалізації клієнтської частини системи було обрано бібліотеку React.js версії 18, яка на сьогоднішній день є одним з найпопулярніших інструментів розробки сучасних веб-додатків. Вибір React.js обґрунтовується низкою переваг, які критично важливі для системи моніторингу транспорту. По-перше, React використовує віртуальний DOM (Document Object Model), що забезпечує високу продуктивність при оновленні інтерфейсу користувача навіть при частих змінах даних, які є характерними для систем реального часу. Механізм узгодження (reconciliation) дозволяє мінімізувати кількість операцій з реальним DOM, що критично важливо при відображенні динамічної інформації про переміщення транспортних засобів [21].

Компонентна архітектура React забезпечує високий рівень модульності та повторного використання коду, що значно спрощує розробку та підтримку складного користувацького інтерфейсу. Кожен функціональний елемент системи може бути реалізований у вигляді незалежного компонента з власним станом та логікою, що полегшує тестування окремих частин додатку та дозволяє розподілити роботу між різними розробниками. Декларативний підхід до опису

інтерфейсу робить код більш читабельним та зменшує ймовірність виникнення помилок порівняно з імперативними підходами до маніпуляції DOM.

Екосистема React включає велику кількість готових бібліотек та інструментів, які значно прискорюють процес розробки. Зокрема, для управління складним станом додатку можуть використовуватися бібліотеки Redux або MobX, а для маршрутизації між різними розділами додатку застосовується React Router. Наявність офіційних інструментів розробки React Developer Tools полегшує процес налагодження та оптимізації продуктивності додатку [22].

Для відображення інтерактивних карт було обрано бібліотеку Leaflet, яка є провідним відкритим рішенням для створення картографічних веб-додатків. Leaflet характеризується малим розміром бібліотеки (приблизно 42 кілобайти в стисненому вигляді), високою продуктивністю навіть при відображенні великої кількості маркерів на карті та підтримкою широкого спектру мобільних та десктопних браузерів. На відміну від комерційних альтернатив, таких як Google Maps JavaScript API, Leaflet надає повний контроль над відображенням карти та не накладає обмежень на кількість запитів або використання в некомерційних проєктах.

Архітектура Leaflet побудована на принципі розширюваності через систему плагінів, що дозволяє додавати необхідну функціональність без зміни базового коду бібліотеки. Для інтеграції Leaflet з React використовується бібліотека React-Leaflet, яка надає набір React компонентів, що інкапсулюють функціональність Leaflet та забезпечують природну інтеграцію з React екосистемою. Це дозволяє керувати картою декларативно, використовуючи JSX синтаксис, що характерний для React додатків.

Для здійснення HTTP запитів до серверного API було обрано бібліотеку Axios, яка надає зручний інтерфейс для роботи з асинхронними запитами та автоматичну трансформацію JSON даних. Axios підтримує перехоплювачі запитів та відповідей (interceptors), що дозволяє централізовано обробляти автентифікацію, логування та обробку помилок. На відміну від нативного Fetch

API, Axios забезпечує кращу сумісність зі старішими версіями браузерів та має більш зручний API для скасування запитів та налаштування таймаутів.

Для забезпечення комунікації в режимі реального часу на клієнтській стороні використовується бібліотека Socket.IO Client, яка автоматично обирає найбільш оптимальний транспортний протокол з доступних (WebSocket, HTTP long-polling, Server-Sent Events) залежно від можливостей браузера та мережевого оточення. Це забезпечує надійну доставку оновлень про позиції транспортних засобів навіть в умовах нестабільного з'єднання або при роботі через проксі-сервери, які можуть блокувати WebSocket з'єднання.

Інтеграція з Google Maps API здійснюється для забезпечення функціональності планування маршрутів між довільними точками міста. Google Maps надає потужні сервіси геокодування для перетворення текстових адрес в географічні координати, побудови оптимальних маршрутів з урахуванням поточної дорожньої ситуації та розрахунку часу в дорозі. Використання Places API дозволяє реалізувати автодоповнення адрес при введенні користувачем, що значно покращує зручність використання системи [23].

2.1.1 Технології серверної частини системи

Серверна частина системи реалізована на платформі Node.js, яка являє собою серверне середовище виконання JavaScript коду, побудоване на базі рушія V8 від компанії Google. Вибір Node.js обґрунтовується його архітектурою, яка ідеально підходить для додатків, що обробляють велику кількість одночасних з'єднань з відносно невеликим обчислювальним навантаженням на кожне з'єднання, що є типовим сценарієм для систем моніторингу реального часу.

Однопоточна асинхронна модель вводу-виводу Node.js базується на циклі подій (event loop), який забезпечує неблокуючу обробку операцій вводу-виводу. Коли виконується операція, яка потребує очікування, така як читання з бази даних або мережевий запит, Node.js не блокує виконання інших операцій, а реєструє callback функцію, яка буде викликана по завершенню операції. Це

дозволяє одному процесу Node.js ефективно обслуговувати тисячі одночасних з'єднань без необхідності створення окремого потоку для кожного з'єднання, як це робиться в традиційних багатопоточних серверних платформах.

Використання JavaScript як на клієнтській, так і на серверній стороні забезпечує уніфікацію технологічного стеку, що дозволяє розробникам працювати з усіма рівнями додатку, використовуючи єдину мову програмування. Це спрощує обмін кодом між клієнтом та сервером, зокрема для валідації даних та обробки бізнес-логіки, а також знижує поріг входу для нових розробників у проєкт [22].

Для побудови RESTful API було обрано фреймворк Express.js, який є де-факто стандартом для розробки веб-додатків на Node.js. Express надає мінімалістичний, але потужний набір функцій для обробки HTTP запитів, маршрутизації, роботи з middleware та рендерингу відповідей. Архітектура middleware в Express дозволяє організувати обробку запитів у вигляді ланцюжка функцій, кожна з яких виконує певну задачу, таку як логування, автентифікація, парсинг тіла запиту або обробка помилок.

Мінімалістичний підхід Express означає, що фреймворк не нав'язує певну структуру проєкту або архітектурні рішення, надаючи розробникам свободу у виборі найбільш підходящих патернів проєктування. Водночас, велика екосистема middleware модулів дозволяє легко інтегрувати необхідну функціональність, таку як обробка CORS запитів, стиснення відповідей, обмеження частоти запитів та захист від типових веб-атак.

Для реалізації двостороннього обміну даними в режимі реального часу використовується бібліотека Socket.IO, яка надає високорівневу абстракцію над протоколом WebSocket з автоматичним fallback до альтернативних транспортних механізмів при недоступності WebSocket. Socket.IO автоматично керує reconnection при втраті з'єднання, підтримує передачу бінарних даних та надає механізм rooms для організації груп з'єднань, що дозволяє ефективно розсилати оновлення тільки зацікавленим клієнтам.

Концепція rooms в Socket.IO дозволяє організувати підписку клієнтів на

оновлення конкретних маршрутів, що значно знижує навантаження на мережу та процесор клієнтського пристрою, оскільки клієнт отримує тільки релевантні для нього дані. Коли користувач обирає певний маршрут для перегляду, клієнт приєднується до відповідної room, після чого сервер розсилає оновлення про позиції транспортних засобів на цьому маршруті тільки учасникам даної room.

Для взаємодії з базою даних MySQL використовується бібліотека mysql2, яка є сучасною реалізацією MySQL клієнта для Node.js з підтримкою prepared statements та асинхронних операцій через Promise API. На відміну від оригінальної бібліотеки mysql, mysql2 надає значно кращу продуктивність завдяки використанню нативних Promise та підтримки stream обробки великих результатів запитів. Connection pooling в mysql2 дозволяє ефективно управляти множиною з'єднань з базою даних, повторно використовуючи існуючі з'єднання замість створення нових для кожного запиту [21].

2.1.2 Система управління базами даних

Для зберігання структурованих даних системи було обрано реляційну систему управління базами даних MySQL версії 8.0, яка є однією з найпопулярніших та перевірених часом СУБД з відкритим вихідним кодом. Вибір реляційної моделі даних обґрунтовується характером інформації, що зберігається в системі, яка має чітку структуру та складні взаємозв'язки між сутностями, такими як маршрути, зупинки, транспортні засоби та GPS координати.

MySQL забезпечує повну підтримку ACID властивостей (Atomicity, Consistency, Isolation, Durability) транзакцій, що гарантує цілісність даних навіть при одночасному виконанні множини операцій модифікації. Механізм транзакцій дозволяє групувати логічно пов'язані операції та відкочувати їх у випадку виникнення помилок, що критично важливо для забезпечення коректності даних в системі [19].

Підтримка складних SQL запитів з множинними JOIN операціями дозволяє ефективно витягувати агреговану інформацію з декількох пов'язаних таблиць в

одному запиті, що знижує навантаження на мережу та прискорює обробку даних. Можливість створення індексів на полях, які часто використовуються в умовах WHERE та JOIN, значно підвищує швидкість виконання запитів читання даних. MySQL підтримує різні типи індексів, включаючи B-tree індекси для звичайних полів та просторові індекси для геоданих.

MySQL 8.0 включає підтримку JSON типу даних, що дозволяє зберігати неструктуровані або частково структуровані дані в реляційній базі, зберігаючи при цьому можливість індексації та ефективного запиту окремих полів JSON документів. Це може бути корисним для зберігання додаткових метаданих про транспортні засоби або маршрути, структура яких може змінюватися з часом.

Механізм реплікації в MySQL дозволяє створювати копії бази даних на декількох серверах, що забезпечує відмовостійкість системи та можливість розподілу навантаження читання між репліками. Master-slave реплікація підтримує асинхронну або напівсинхронну передачу змін, що дозволяє балансувати між продуктивністю та гарантією збереження даних [18].

Підтримка географічних типів даних та просторових функцій в MySQL дозволяє ефективно виконувати геопросторові запити, такі як пошук всіх зупинок в радіусі певної відстані від точки або визначення чи знаходиться координата всередині певної області. Це може використовуватися для оптимізації запитів при відображенні зупинок на карті або при пошуку найближчої зупинки до користувача.

2.1.3 Додаткові технології та інструменти

Для управління змінами в схемі бази даних використовуються інструменти міграцій, які дозволяють версіонувати структуру бази даних подібно до того, як системи контролю версій керують змінами в коді. Це забезпечує можливість відслідковувати історію змін схеми, відкочувати невдалі зміни та автоматизувати процес розгортання оновлень структури бази даних на різних оточеннях.

Для забезпечення безпеки з'єднань між клієнтом та сервером

використовується протокол HTTPS, який шифрує весь трафік за допомогою TLS сертифікатів. Це запобігає перехопленню даних при передачі через незахищені мережі та підтверджує автентичність сервера для клієнта. Для отримання безкоштовних TLS сертифікатів може використовуватися сервіс Let's Encrypt з автоматичним продовженням терміну дії.

Для обмеження частоти запитів від одного клієнта та захисту від DDoS атак використовується middleware для rate limiting, який відстежує кількість запитів з певної IP адреси в одиницю часу та блокує запити при перевищенні встановленого ліміту. Це запобігає перевантаженню сервера навмисними або ненавмисними зловживаннями API [17].

Логування подій системи реалізується за допомогою бібліотек, таких як Winston або Bunyan, які надають структуроване логування з різними рівнями деталізації (error, warn, info, debug) та можливістю направлення логів у різні цілі (файли, консоль, віддалені сервіси збору логів). Структуровані логи дозволяють ефективно аналізувати поведінку системи та швидко діагностувати проблеми.

Для моніторингу продуктивності додатку можуть використовуватися APM (Application Performance Monitoring) рішення, які збирають метрики про час виконання запитів, використання пам'яті, навантаження на процесор та інші показники. Це дозволяє виявляти вузькі місця в продуктивності та оптимізувати критичні частини системи.

2.1.4 Обґрунтування технологічних рішень

Обраний технологічний стек базується на сучасних, перевірених у продакшн оточеннях технологіях з активною підтримкою спільноти та регулярними оновленнями безпеки. Використання технологій з відкритим вихідним кодом знижує вартість володіння системою та забезпечує незалежність від конкретних постачальників програмного забезпечення.

Уніфікація мови програмування (JavaScript) на клієнтській та серверній стороні знижує складність розробки та підтримки системи, дозволяє використовувати спільні бібліотеки та утиліти, а також полегшує перехід

розробників між різними частинами проєкту. Асинхронна природа Node.js ідеально підходить для додатків реального часу, де необхідно підтримувати велику кількість одночасних з'єднань з мінімальними затримками [20].

Компонентна архітектура React та модульна структура Express забезпечують високий рівень повторного використання коду та спрощують тестування окремих частин системи. Можливість розробки та тестування компонентів ізольовано прискорює процес розробки та знижує ймовірність внесення регресій при додаванні нової функціональності.

Використання Leaflet для відображення карт забезпечує повний контроль над візуалізацією та не накладає обмежень на використання, характерних для комерційних картографічних сервісів. Водночас, інтеграція з Google Maps API для планування маршрутів дозволяє використовувати переваги потужних алгоритмів побудови оптимальних шляхів, які враховують поточну дорожню ситуацію.

Реляційна база даних MySQL забезпечує надійність та цілісність даних завдяки підтримці ACID властивостей транзакцій, а також дозволяє ефективно виконувати складні запити з об'єднаннями множини таблиць. Зріла екосистема інструментів для резервного копіювання, моніторингу та оптимізації продуктивності MySQL знижує ризики при експлуатації системи.

Таким чином, обраний технологічний стек забезпечує оптимальний баланс між продуктивністю, масштабованістю, зручністю розробки та вартістю володіння, що робить його придатним для реалізації інформаційної системи моніторингу міського транспорту з перспективою подальшого розвитку та масштабування.

2.1.5 Візуальне представлення технологічного стеку

Для систематизації та наочного представлення обраних технологій було розроблено діаграму компонентів у нотації UML, яка відображає багаторівневу архітектуру технологічного стеку системи моніторингу міського транспорту. Діаграма структурована у вигляді чотирьох основних рівнів, що відповідають

логічному розподілу технологій за їх функціональним призначенням та місцем в архітектурі системи [21].

Верхній рівень діаграми представлено пакетом «Клієнтський рівень», який об'єднує всі технології, що виконуються в браузері користувача. Цей рівень організовано у п'ять функціональних підпакетів відповідно до специфічних задач, які вирішують відповідні технології. Підпакет «UI Framework» містить компонент React.js версії 18, який є базовою бібліотекою для побудови користувацького інтерфейсу. До компонента додано анотацію, що деталізує ключові особливості React, зокрема механізм віртуального DOM, компонентну архітектуру, Hooks API для управління станом компонентів та JSX синтаксис для декларативного опису інтерфейсу.

Підпакет «Картографія» включає два взаємопов'язані компоненти. Основний компонент Leaflet версії 1.9 відповідає за низькорівневу роботу з картами, використовуючи векторну графіку для ефективного відображення географічних об'єктів. Анотація до цього компонента підкреслює малий розмір бібліотеки у стисненому вигляді, що становить лише сорок два кілобайти, та наявність системи плагінів для розширення функціональності. Компонент React-Leaflet виступає як обгортка над Leaflet, яка забезпечує інтеграцію картографічної функціональності з React екосистемою, дозволяючи керувати картою декларативно через React компоненти.

Підпакет «HTTP Клієнт» представлено компонентом Axios, який забезпечує здійснення асинхронних HTTP запитів до серверного API. Супровідна анотація вказує на підтримку Promise API для зручної роботи з асинхронними операціями, наявність механізму interceptors для централізованої обробки запитів та відповідей, а також автоматичну трансформацію JSON даних, що спрощує роботу з серверним API.

Підпакет «Real-time» містить компонент Socket.IO Client, який відповідає за підтримку з'єднання з сервером в режимі реального часу. Анотація деталізує використання протоколу WebSocket для двостороннього обміну даними, механізм автоматичного повторного з'єднання при втраті зв'язку та можливість

автоматичного вибору альтернативного транспортного протоколу, якщо WebSocket недоступний в мережевому оточенні.

Підпакет «Зовнішні API» включає компонент Google Maps API, який надає доступ до картографічних сервісів компанії Google. Анотація перераховує три основні API, які використовуються в системі: Directions API для побудови маршрутів між точками, Places API для автодоповнення адрес та пошуку місць, та Geocoding API для перетворення текстових адрес в географічні координати та навпаки.

Другий рівень діаграми представлено пакетом «Мережевий рівень», який містить компоненти протоколів та механізмів, що забезпечують комунікацію між клієнтською та серверною частинами. Компонент HTTPS/TLS відповідає за шифрування HTTP трафіку з використанням Transport Layer Security протоколу, що гарантує конфіденційність та цілісність переданих даних. Додаткова примітка вказує на використання безкоштовних сертифікатів від сервісу Let's Encrypt для забезпечення HTTPS з'єднань. Компонент WebSocket Protocol представляє окремий протокол для підтримки постійного двостороннього з'єднання між клієнтом та сервером. Компонент CORS (Cross-Origin Resource Sharing) відповідає за управління політиками безпеки, які регулюють можливість виконання запитів до API з різних доменів [19].

Третій рівень діаграми представлено пакетом «Серверний рівень», який є найбільш складним за структурою та включає множину підпакетів та компонентів. Підпакет «Runtime» містить компонент Node.js версії 18 LTS (Long Term Support), який забезпечує середовище виконання JavaScript коду на сервері. Детальна анотація описує використання рушія V8 від компанії Google для компіляції та виконання JavaScript, механізм циклу подій (Event Loop) для неблокуючої обробки операцій, асинхронну модель вводу-виводу та однопоточну архітектуру з можливістю обробки множини одночасних операцій.

Підпакет «Web Framework» включає основний компонент Express.js версії 4 та три middleware компоненти. Express.js анотовано як фреймворк, що

забезпечує middleware pipeline для послідовної обробки запитів, систему маршрутизації для направлення запитів до відповідних обробників та побудову RESTful API. Компонент CORS Middleware реалізує серверну частину політики CORS, визначаючи які домени мають доступ до API. Компонент Rate Limiter відповідає за обмеження частоти запитів від одного клієнта для захисту від зловживань. Компонент Body Parser здійснює парсинг тіла HTTP запитів різних форматів, перетворюючи їх у JavaScript об'єкти.

Підпакет «Real-time Server» містить компонент Socket.IO Server, який є серверною частиною системи real-time комунікації. Анотація підкреслює підтримку концепції rooms для групування з'єднань, механізму broadcasting для розсилки повідомлень множині клієнтів одночасно та namespaces для логічного розділення різних каналів комунікації.

Підпакет «Database Client» представлено компонентом mysql2, який забезпечує взаємодію серверного додатку з базою даних MySQL. Анотація вказує на використання connection pool для ефективного управління множиною з'єднань з базою даних, Promise API для зручної роботи з асинхронними запитами та prepared statements для захисту від SQL ін'єкцій та підвищення продуктивності повторюваних запитів.

Підпакет «Utilities» об'єднує допоміжні компоненти загального призначення. Компонент Winston Logger забезпечує структуроване логування подій системи з підтримкою різних рівнів деталізації та можливістю направлення логів у різні цілі. Компонент dotenv відповідає за завантаження конфігураційних параметрів з файлів оточення, що дозволяє налаштовувати поведінку додатку без зміни коду.

Четвертий рівень діаграми представлено пакетом «Рівень даних», який містить компоненти системи управління базою даних MySQL версії 8.0. Основна база даних MySQL включає чотири внутрішні компоненти, що представляють різні аспекти її функціональності. Компонент InnoDB Engine є механізмом зберігання даних, що забезпечує підтримку ACID властивостей транзакцій, блокування на рівні рядків для підвищення паралелізму та

підтримку зовнішніх ключів для забезпечення цілісності даних. Анотація детально описує ці критично важливі характеристики.

Компонент Spatial Index представляє спеціалізований тип індексу для ефективного виконання геопросторових запитів, таких як пошук об'єктів у певній області або визначення відстані між точками. Компонент B-Tree Index є класичним типом індексу, що використовується для прискорення пошуку за звичайними полями таблиць. Компонент JSON Support відображає підтримку MySQL 8.0 нативного JSON типу даних з можливістю індексації та запиту окремих полів JSON документів [18].

Окремий елемент storage «Таблиці» містить перелік основних таблиць системи: routes для маршрутів, stops для зупинок, route_stops для зв'язку між маршрутами та зупинками, vehicles для транспортних засобів та gps_data для історії GPS координат. Цей елемент візуально відображає фізичне зберігання структурованих даних у базі.

Взаємозв'язки між компонентами на діаграмі представлено різними типами ліній залежно від характеру зв'язку. Суцільні лінії зі стрілками вказують на прямі залежності та використання одного компонента іншим в межах одного рівня. Наприклад, React використовує React-Leaflet, який у свою чергу є обгорткою над Leaflet. Express використовує множину middleware компонентів, організованих у ланцюжок обробки запитів.

Пунктирні лінії зі стрілками позначають мережеву комунікацію між різними рівнями архітектури. Axios з клієнтського рівня здійснює запити через протокол HTTPS до Express на серверному рівні. Socket.IO Client підтримує з'єднання через протокол WebSocket з Socket.IO Server. Google Maps API також комунікує через HTTPS для отримання картографічних даних та виконання геокодування.

Особливу увагу на діаграмі приділено відображенню middleware pipeline в Express. Компоненти CORS Middleware, Rate Limiter та Body Parser послідовно обробляють кожен вхідний запит перед тим, як він досягне бізнес-логіки додатку. Ця послідовна обробка забезпечує виконання крос-функціональних

вимог, таких як безпека, обмеження доступу та парсинг даних.

Зв'язок між серверним рівнем та рівнем даних реалізовано через компонент mysql2, який використовує нативний протокол MySQL для виконання SQL запитів. Цей компонент абстрагує деталі протоколу комунікації з базою даних, надаючи зручний JavaScript API для виконання запитів та обробки результатів.

Діаграма також демонструє інтеграцію між Express та Socket.IO Server, які хоча і є окремими компонентами, але тісно співпрацюють для забезпечення як традиційного HTTP API, так і real-time комунікації через WebSocket. Обидва компоненти мають доступ до бази даних через mysql2 для читання та запису даних.

Використання анотацій для кожного ключового компонента забезпечує діаграму додатковим контекстом, що допомагає зрозуміти не лише структурну організацію технологій, а й ключові характеристики кожної з них. Ці анотації виділяють найважливіші особливості технологій, які обґрунтовують їх вибір для реалізації системи [16].

Колірне кодування пакетів на діаграмі відповідає різним рівням архітектури: блакитний колір для клієнтського рівня підкреслює його орієнтацію на взаємодію з користувачем, помаранчевий для мережевого рівня символізує транзитний характер цього рівня, зелений для серверного рівня асоціюється з бізнес-логікою та обробкою даних, а рожевий для рівня даних вказує на критично важливу роль збереження інформації.

Представлена діаграма технологічного стеку демонструє чітку стратифікацію архітектури системи на логічні рівні з мінімізацією зв'язків між рівнями через добре визначені інтерфейси. Така організація забезпечує можливість незалежної еволюції технологій на різних рівнях без необхідності перепроєктування всієї системи. Наприклад, заміна Leaflet на альтернативну картографічну бібліотеку не вплине на серверний рівень, оскільки взаємодія здійснюється виключно через стандартизований REST API.

На рис. 2.1, представлено діаграму, яка візуалізує модульність серверного рівня, де кожен компонент має чітко визначену відповідальність та може бути

замінений або оновлений незалежно від інших компонентів за умови збереження інтерфейсів взаємодії. Middleware архітектура Express дозволяє легко додавати нові функціональні можливості шляхом впровадження додаткових middleware компонентів у ланцюжок обробки запитів.

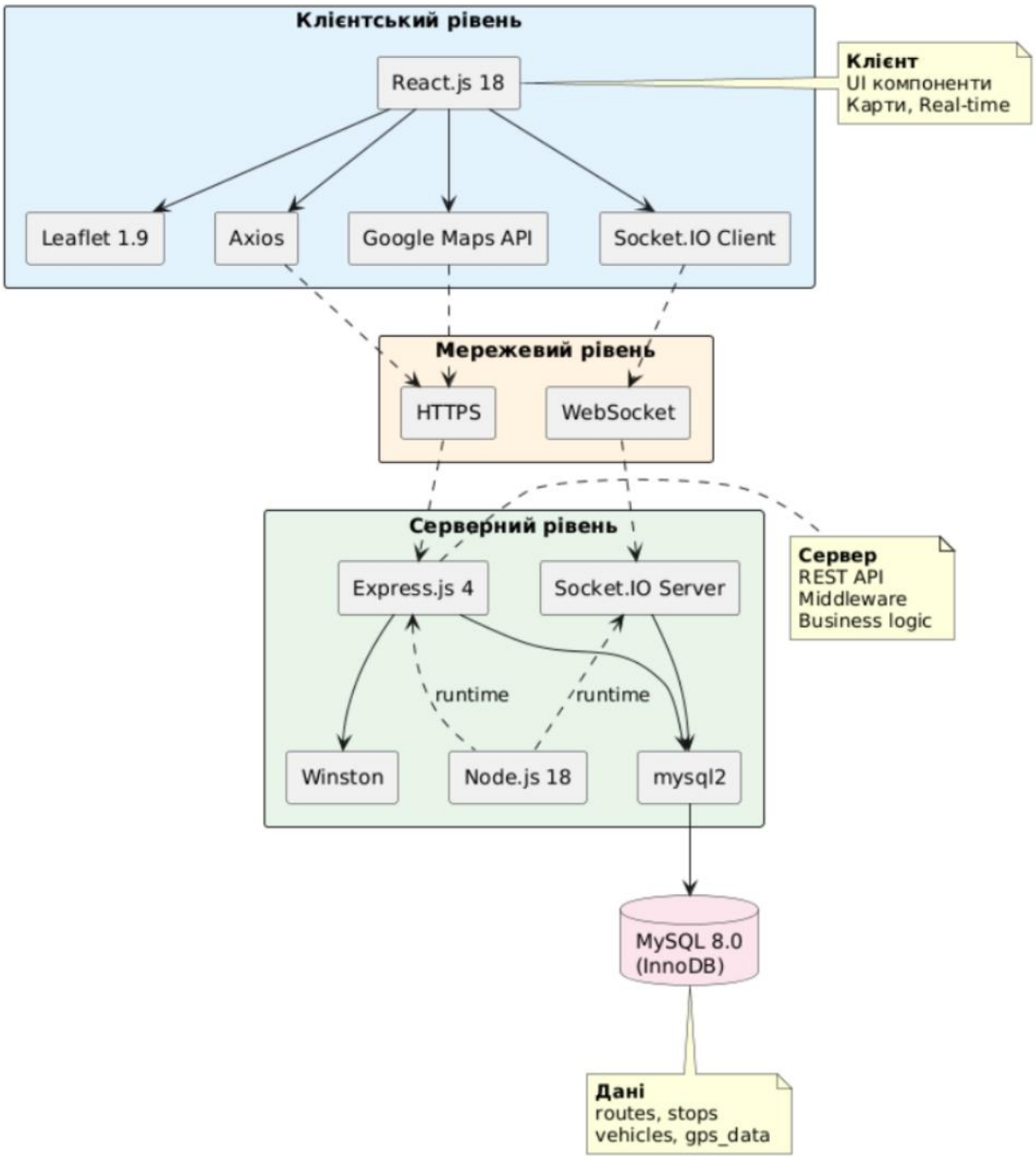

Рисунок 2.1 – Діаграма технологічного стеку

Використання стандартизованих протоколів комунікації, таких як HTTP, HTTPS та WebSocket, забезпечує сумісність системи з широким спектром клієнтських пристроїв та платформ, що особливо важливо для веб-додатків, які

повинні працювати на різних типах пристроїв з різними операційними системами та браузерами. Діаграма чітко показує точки інтеграції з зовнішніми сервісами, такими як Google Maps API, що полегшує розуміння залежностей системи від третіх сторін [19].

Таким чином, розроблена діаграма компонентів технологічного стеку в нотації UML надає комплексне візуальне представлення архітектурних рішень щодо вибору технологій, демонструє їх взаємозв'язки та ілюструє багаторівневу організацію системи від клієнтського інтерфейсу до фізичного зберігання даних. Діаграма слугує важливим документаційним артефактом, який полегшує розуміння технічної архітектури системи як для розробників, так і для інших учасників проєкту.

2.2 Проєктування структури бази даних інформаційної системи та моделі даних

Проєктування структури бази даних є критично важливим етапом розробки інформаційної системи моніторингу міського транспорту, оскільки від якості організації даних залежить продуктивність системи, цілісність інформації та можливості подальшого масштабування. Структура бази даних повинна забезпечувати ефективне зберігання великих обсягів геопросторових даних, підтримку складних зв'язків між сутностями предметної області та швидкий доступ до актуальної інформації про стан транспортної системи.

Предметна область системи моніторингу міського транспорту характеризується наявністю декількох ключових сутностей та складних взаємозв'язків між ними. Основною сутністю є транспортний маршрут, який представляє собою встановлений шлях руху певного виду громадського транспорту між початковою та кінцевою зупинками. Кожен маршрут має унікальний номер, назву, тип транспорту та додаткові атрибути, такі як колір для візуалізації на карті та статус активності.

Зупинки громадського транспорту є другою ключовою сутністю системи, яка представляє фізичні місця, де пасажири можуть сісти або вийти з

транспортного засобу. Кожна зупинка характеризується географічними координатами, унікальною назвою та може обслуговуватися множиною різних маршрутів. Важливою особливістю є те, що зупинка може зустрічатися на одному маршруті двічі у випадку кільцевих маршрутів або маршрутів з однаковими зупинками у прямому та зворотному напрямках.

Зв'язок між маршрутами та зупинками має складний характер, оскільки один маршрут проходить через множину зупинок у певній послідовності, а одна зупинка може належати множині різних маршрутів. Така взаємодія класифікується як зв'язок типу багато-до-багатьох, що потребує створення окремої асоціативної сутності для зберігання інформації про конкретні випадки проходження маршруту через зупинку. Ця асоціативна сутність зберігає додаткові атрибути, такі як порядковий номер зупинки на маршруті та напрямок руху, що дозволяє коректно відображати послідовність зупинок для прямого та зворотного напрямків руху.

Транспортні засоби представляють собою фізичні одиниці рухомого складу, такі як автобуси, тролейбуси або трамваї, які виконують перевезення пасажирів на конкретних маршрутах. Кожен транспортний засіб має унікальний бортовий номер, тип транспорту та прив'язку до певного маршруту, на якому він здійснює рейси. Важливим атрибутом є статус активності транспортного засобу, який визначає чи знаходиться він наразі на лінії чи у депо.

Геопросторові дані про переміщення транспортних засобів зберігаються у вигляді послідовності GPS координат з часовими мітками. Кожен запис містить широту та довготу положення транспортного засобу у конкретний момент часу, а також додаткові параметри, такі як швидкість руху та азимут напрямку. Обсяг таких даних може бути значним, оскільки система збирає інформацію від множини транспортних засобів з високою частотою, що накладає особливі вимоги до організації їх зберігання та індексації [15,16].

2.2.1 Логічна модель даних

На основі концептуального аналізу предметної області було розроблено логічну модель даних, яка використовує реляційну парадигму для представлення сутностей та їх взаємозв'язків. Центральною таблицею моделі є таблиця маршрутів routes, яка зберігає базову інформацію про кожен транспортний маршрут. Первинним ключем таблиці виступає автоінкрементний ідентифікатор id, який забезпечує унікальність кожного запису. Атрибут route_number зберігає номер маршруту у текстовому форматі, оскільки номери маршрутів можуть включати літери та інші символи, наприклад «10А» або «240Е». Поле route_name містить повну назву маршруту, яка зазвичай включає назви початкової та кінцевої зупинок.

Тип транспорту зберігається в полі transport_type з використанням перелічуваного типу даних ENUM, що обмежує можливі значення заздалегідь визначеним набором: bus для автобусів, trolleybus для тролейбусів, tram для звичайних трамваїв та speed_tram для швидкісних трамваїв. Використання перелічуваного типу забезпечує цілісність даних на рівні бази даних та економить місце для зберігання порівняно з текстовими полями. Атрибут color зберігає колір маршруту у шістнадцятковому форматі для використання при візуалізації на карті, що дозволяє користувачам легко розрізняти різні маршрути. Логічне поле is_active визначає чи є маршрут діючим на даний момент, що дозволяє зберігати історичну інформацію про маршрути, які більше не функціонують, без їх видалення з бази даних [14].

Таблиця зупинок stops містить інформацію про всі зупинки громадського транспорту в місті. Кожна зупинка ідентифікується унікальним ідентифікатором id та має назву stop_name, яка зберігається у форматі Unicode для коректної підтримки назв різними мовами. Географічне положення зупинки визначається парою координат latitude та longitude, які зберігаються як числа з плаваючою комою подвійної точності DOUBLE для забезпечення необхідної точності позиціонування. Точність до шести знаків після коми забезпечує точність визначення положення до одного метра, що є достатнім для потреб

системи моніторингу транспорту.

Асоціативна таблиця route_stops реалізує зв'язок багато-до-багатьох між маршрутами та зупинками. Кожен запис у цій таблиці представляє конкретний випадок проходження маршруту через зупинку та містить зовнішні ключі route_id та stop_id, які посилаються на відповідні записи в таблицях routes та stops. Атрибут stop_sequence зберігає порядковий номер зупинки на маршруті, що дозволяє визначити правильну послідовність зупинок при побудові шляху маршруту на карті. Поле direction використовується для розрізнення зупинок прямого та зворотного напрямків руху, що особливо важливо для маршрутів, де зупинки різняться залежно від напрямку або розташовані на різних сторонах вулиці.

Таблиця транспортних засобів vehicles зберігає інформацію про фізичні одиниці рухомого складу. Первинний ключ id однозначно ідентифікує кожен транспортний засіб у системі. Поле vehicle_number містить бортовий номер, який зазвичай відображається на корпусі транспортного засобу та використовується для його ідентифікації водіями та диспетчерами. Тип транспортного засобу зберігається в полі transport_type з тим самим набором можливих значень, що й у таблиці маршрутів, що забезпечує узгодженість даних. Зовнішній ключ route_id встановлює зв'язок транспортного засобу з маршрутом, на якому він працює, причому це поле може приймати значення NULL для транспортних засобів, які тимчасово не закріплені за жодним маршрутом. Логічне поле is_active визначає чи знаходиться транспортний засіб на лінії у даний момент часу [20].

Таблиця GPS даних gps_data є найбільш динамічною частиною бази даних, оскільки постійно поповнюється новими записами про переміщення транспортних засобів. Кожен запис ідентифікується унікальним ідентифікатором id та обов'язково містить зовнішній ключ vehicle_id, який посилається на транспортний засіб, від якого отримано дані. Географічні координати latitude та longitude зберігаються з тим самим рівнем точності, що й координати зупинок. Часова мітка timestamp типу DATETIME фіксує точний

момент реєстрації координат, що критично важливо для аналізу траєкторій руху та визначення поточного положення транспортного засобу.

Додаткові атрибути speed та heading зберігають інформацію про швидкість руху транспортного засобу у кілометрах на годину та азимут напрямку руху у градусах відповідно. Ці дані можуть бути корисними для аналітики та прогнозування часу прибуття транспорту на зупинки. Поле speed може приймати значення NULL у випадках, коли транспортний засіб стоїть на місці або дані про швидкість недоступні. Аналогічно, поле heading може бути NULL для нерухомого транспорту.

2.2.2 Забезпечення цілісності даних

Цілісність даних у розробленій моделі забезпечується на декількох рівнях через використання механізмів, які надає система управління базами даних MySQL. На рівні структури таблиць використовуються обмеження первинних ключів PRIMARY KEY, які гарантують унікальність кожного запису в таблиці та автоматично створюють кластерний індекс для ефективного доступу до даних. Автоінкрементні ідентифікатори забезпечують автоматичне генерування унікальних значень первинних ключів при вставці нових записів без необхідності ручного управління ідентифікаторами.

Зовнішні ключі FOREIGN KEY встановлюють референційну цілісність між пов'язаними таблицями, забезпечуючи що значення в полях, які посилаються на інші таблиці, завжди відповідають існуючим записам у цих таблицях. Наприклад, зовнішній ключ на полі route_id у таблиці route_stops гарантує, що не може бути створено зв'язок з неіснуючим маршрутом. При спробі видалення запису, на який посилаються інші таблиці, база даних або відхилить операцію, або виконає каскадне видалення залежних записів, залежно від налаштованої стратегії ON DELETE [18].

Обмеження NOT NULL на критичних полях забезпечує що важлива інформація завжди присутня у записах. Наприклад, запис про маршрут обов'язково повинен мати номер та тип транспорту, а запис GPS координат

повинен містити дані про широту, довготу та часову мітку. Це запобігає появі неповних записів, які могли б призвести до некоректної роботи системи.

Використання перелічуваних типів ENUM для полів з обмеженим набором можливих значень забезпечує контроль коректності даних на рівні бази даних. Спроба вставити значення, яке не входить до визначеного набору, буде відхилена системою управління базою даних, що запобігає появі некоректних даних через помилки в програмному коді або ручному введенні даних.

Унікальні індекси UNIQUE на комбінаціях полів забезпечують запобігання дублювання логічно унікальних комбінацій даних. Наприклад, комбінація route_id, stop_id, direction та stop_sequence в таблиці route_stops повинна бути унікальною, оскільки зупинка не може двічі зустрічатися на маршруті в одному напрямку з однаковим порядковим номером.

2.2.3 Оптимізація продуктивності через індексацію

Продуктивність роботи з базою даних значною мірою залежить від правильно організованої системи індексів, яка дозволяє швидко знаходити необхідні дані без повного сканування таблиць. В розробленій моделі використовується декілька типів індексів відповідно до характеру запитів, які виконуються системою.

На всіх полях зовнішніх ключів створюються некластерні B-tree індекси, які значно прискорюють операції об'єднання JOIN таблиць. Наприклад, індекс на полі route_id у таблиці route_stops дозволяє швидко знайти всі зупинки конкретного маршруту без необхідності сканування всієї таблиці. Аналогічно, індекс на полі vehicle_id у таблиці gps_data забезпечує ефективний пошук всіх GPS координат конкретного транспортного засобу [21].

Для таблиці stops створюється просторовий індекс SPATIAL INDEX на комбінацію полів latitude та longitude, що дозволяє ефективно виконувати геопросторові запити, такі як пошук всіх зупинок у певному радіусі від точки або визначення найближчої зупинки до користувача. Просторовий індекс використовує R-tree структуру даних, яка оптимізована для роботи з

багатовимірними геометричними об'єктами.

Таблиця gps_data потребує особливої уваги до індексації через великий обсяг даних та специфічні патерни доступу. Окрім індексу на vehicle_id, створюється композитний індекс на комбінацію vehicle_id та timestamp, що дозволяє ефективно виконувати запити на отримання останніх координат транспортного засобу або історії його переміщень за певний період часу. Порядок полів у композитному індексі обирається таким чином, щоб максимально відповідати найбільш частим запитам системи.

Для підтримки високої продуктивності при зростанні обсягу історичних GPS даних може застосовуватися стратегія партиціонування таблиці gps_data за часовим критерієм. Це дозволяє розділити велику таблицю на менші фізичні сегменти за місяцями або роками, що прискорює виконання запитів, які стосуються лише актуальних даних, та спрощує процес архівування старих записів.

2.2.4 Нормалізація структури бази даних

Розроблена структура бази даних відповідає третій нормальній формі реляційних баз даних, що забезпечує мінімізацію надмірності даних та запобігає аномаліям при виконанні операцій вставки, оновлення та видалення. Відповідність першій нормальній формі забезпечується тим, що всі атрибути таблиць є атомарними та не містять повторюваних груп. Кожне поле містить одне значення певного типу, а множинні значення, такі як список зупинок маршруту, виносяться в окрему асоціативну таблицю [22].

Друга нормальна форма вимагає відсутності часткової залежності неключових атрибутів від складеного ключа. В розробленій моделі асоціативна таблиця route_stops має складений первинний ключ з полів route_id, stop_id та direction, при цьому атрибут stop_sequence функціонально залежить від всього складеного ключа цілком, а не від його частини. Інформація про саму зупинку, така як її назва та координати, зберігається в окремій таблиці stops та не дублюється в асоціативній таблиці.

Третя нормальна форма вимагає відсутності транзитивних залежностей неключових атрибутів. В моделі всі неключові атрибути безпосередньо залежать від первинного ключа своєї таблиці та не залежать від інших неключових атрибутів. Наприклад, колір маршруту зберігається в таблиці routes та однозначно визначається ідентифікатором маршруту, а не виводиться з якихось інших атрибутів.

Нормалізована структура забезпечує відсутність дублювання інформації про зупинки при використанні їх на множині маршрутів. Кожна зупинка описується один раз в таблиці stops, після чого на неї посилаються записи в асоціативній таблиці. Це означає, що при зміні координат зупинки через уточнення її положення необхідно оновити лише один запис, і ця зміна автоматично відобразиться на всіх маршрутах, які використовують цю зупинку.

2.2.5 Стратегія управління геопросторовими даними

Геопросторові дані становлять значну частину інформації, що зберігається в системі, та потребують особливого підходу до організації зберігання та обробки. MySQL 8.0 надає підтримку геопросторових типів даних відповідно до стандарту OpenGIS, що дозволяє зберігати геометричні об'єкти, такі як точки, лінії та полігони, у спеціалізованих полях типу GEOMETRY.

В розробленій моделі координати зупинок та позиції транспортних засобів зберігаються як окремі поля latitude та longitude типу DOUBLE, що забезпечує сумісність з широким спектром бібліотек та інструментів обробки геоданих. Такий підхід дозволяє легко експортувати координати у різних форматах та використовувати їх безпосередньо в клієнтських додатках без необхідності перетворення складних геометричних типів [16].

Для виконання геопросторових запитів MySQL надає набір спеціалізованих функцій, таких як ST_Distance для обчислення відстані між двома точками, ST_Within для визначення чи знаходиться точка всередині певної області та ST_Buffer для створення буферної зони навколо об'єкта. Ці функції можуть використовуватися в SQL запитах для реалізації

функціональності пошуку найближчих зупинок, визначення чи рухається транспортний засіб у межах свого маршруту та інших геопросторових операцій.

Зберігання великих обсягів історичних GPS треків потребує врахування обмежень на розмір бази даних та швидкість доступу до даних. Для оптимізації може застосовуватися стратегія періодичного агрегування детальних GPS точок у узагальнені треки з меншою частотою дискретизації для давніших даних. Це дозволяє зберігати детальну інформацію про нещодавні переміщення для оперативного аналізу, при цьому економлячи місце на диску для історичних даних, які в основному використовуються для статистичного аналізу.

Проектування бази даних здійснювалося з урахуванням перспективи зростання обсягів даних та навантаження на систему. Структура таблиць організована таким чином, щоб мінімізувати необхідність структурних змін при розширенні функціональності системи. Використання ідентифікаторів типу BIGINT замість INT забезпечує достатній діапазон значень навіть при значному зростанні кількості записів [15].

Модель даних підтримує горизонтальне масштабування через можливість розподілу даних між множиною серверів баз даних. Таблиця gps_data може бути розподілена за критерієм vehicle_id або timestamp, що дозволяє розподілити навантаження на запис даних між декількома серверами. Таблиці довідників, такі як routes та stops, які змінюються рідко, можуть реплікуватися на всі сервери для забезпечення швидкого доступу до них без необхідності звернення до центрального сервера.

Використання кешування на рівні додатку дозволяє знизити навантаження на базу даних для часто запитуваних даних, таких як список маршрутів та їх зупинок, які змінюються нечасто. Стратегія інвалідації кешу при зміні даних забезпечує актуальність інформації, яка надається користувачам, при цьому мінімізуючи кількість звернень до бази даних.

У таблиці 2.1 представлено структуру таблиці routes (Маршрути), яка використовується для збереження інформації про побудовані користувачами маршрути в інформаційній системі. Таблиця містить ключові поля, необхідні

для ідентифікації маршруту, його опису та подальшої роботи з ним у системі.

Таблиця 2.1 – Структура таблиці routes (Маршрути)

Назва поля	Тип даних	Розмір	Обмеження	Опис
id	BIGINT	8 байт	PRIMARY KEY, AUTO_INCREMENT, NOT NULL	Унікальний ідентифікатор маршруту
route_number	VARCHAR	10 символів	NOT NULL, INDEX	Номер маршруту (може містити літери, наприклад «10А», «240»)
route_name	VARCHAR	255 символів	NOT NULL	Повна назва маршруту (початкова - кінцева зупинка)
transport_type	ENUM	1 байт	NOT NULL, VALUES('bus', 'trolleybus', 'tram', 'speed_tram')	Тип транспортного засобу
color	VARCHAR	7 символів	DEFAULT '#667eea'	Колір маршруту для візуалізації (HEX формат, наприклад "#FF5733")
is_active	BOOLEAN	1 байт	NOT NULL, DEFAULT TRUE	Статус активності маршруту (TRUE - діючий, FALSE - недіючий)

У таблиці 2.2 представлено структуру таблиці stops (Зупинки), яка використовується для збереження інформації про зупинки громадського транспорту, що використані під час побудови маршрутів. Таблиця містить основні атрибути, необхідні для відображення зупинок на мапі та їх подальшої інтеграції з даними транспорту.

Таблиця 2.2 – Структура таблиці stops (Зупинки)

Назва поля	Тип даних	Розмір	Обмеження	Опис
id	BIGINT	8 байт	PRIMARY KEY, AUTO_INCREMENT, NOT NULL	Унікальний ідентифікатор зупинки
stop_name	VARCHAR	255 символів	NOT NULL, INDEX	Назва зупинки (підтримка UTF-8 для українських назв)
latitude	DOUBLE	8 байт	NOT NULL	Географічна широта у десяткових градусах (діапазон: -90 до +90)
longitude	DOUBLE	8 байт	NOT NULL	Географічна довгота у десяткових градусах (діапазон: -180 до +180)

У таблиці 2.3 наведено структуру таблиці route_stops (Зв'язок маршрут–зупинка), яка використовується для реалізації зв’язку між таблицями routes та stops. Ця таблиця є проміжною (junction table) та дозволяє встановити відповідність між певним маршрутом і зупинками, через які він проходить.

Таблиця 2.3 – Структура таблиці route_stops (Зв'язок маршрут-зупинка)

Назва поля	Тип даних	Розмір	Обмеження	Опис
id	BIGINT	8 байт	PRIMARY KEY, AUTO_INCREMENT, NOT NULL	Унікальний ідентифікатор запису
route_id	BIGINT	8 байт	FOREIGN KEY (routes.id), NOT NULL, INDEX	Ідентифікатор маршруту
stop_id	BIGINT	8 байт	FOREIGN KEY (stops.id), NOT NULL, INDEX	Ідентифікатор зупинки
stop_sequence	INT	4 байт	NOT NULL	Порядковий номер зупинки на маршруті (1, 2, 3...)
direction	ENUM	1 байт	NOT NULL, VALUES('forward', 'backward')	Напрямок руху (прямий/зворотний)

У таблиці 2.4 представлено структуру таблиці vehicles (Транспортні засоби), яка використовується для збереження інформації про транспортні одиниці, що беруть участь у перевезеннях та прив’язані до маршрутів у системі.

Таблиця 2.4 – Структура таблиці vehicles (Транспортні засоби)

Назва поля	Тип даних	Розмір	Обмеження	Опис
id	BIGINT	8 байт	PRIMARY KEY, AUTO_INCREMENT, NOT NULL	Унікальний ідентифікатор транспортного засобу
vehicle_number	VARCHAR	20 символів	NOT NULL, UNIQUE	Бортовий номер транспортного засобу
transport_type	ENUM	1 байт	NOT NULL, VALUES('bus', 'trolleybus', 'tram', 'speed_tram')	Тип транспортного засобу

Продовження таблиці 2.4

Назва поля	Тип даних	Розмір	Обмеження	Опис
route_id	BIGINT	8 байт	FOREIGN KEY (routes.id), NULL, INDEX	Ідентифікатор маршруту (NULL якщо ТЗ не на лінії)
is_active	BOOLEAN	1 байт	NOT NULL, DEFAULT FALSE	Статус активності (TRUE - на лінії, FALSE - у депо)

Таблиця 2.5 – Структура таблиці gps_data (GPS координати)

Назва поля	Тип даних	Розмір	Обмеження	Опис
id	BIGINT	8 байт	PRIMARY KEY, AUTO_INCREMENT, NOT NULL	Унікальний ідентифікатор запису GPS даних
vehicle_id	BIGINT	8 байт	FOREIGN KEY (vehicles.id), NOT NULL, INDEX	Ідентифікатор транспортного засобу
latitude	DOUBLE	8 байт	NOT NULL	Географічна широта позиції ТЗ (діапазон: -90 до +90)
longitude	DOUBLE	8 байт	NOT NULL	Географічна довгота позиції ТЗ (діапазон: -180 до +180)
timestamp	DATETIME	8 байт	NOT NULL, INDEX	Часова мітка фіксації координат (формат: YYYY-MM-DD HH:MM:SS)
speed	DECIMAL	5 байт	NULL, PRECISION(5,2)	Швидкість руху в км/год (діапазон: 0.00 - 999.99, NULL якщо невідома)

Продовження таблиці 2.5

Назва поля	Тип даних	Розмір	Обмеження	Опис
heading	DECIMAL	5 байт	NULL, PRECISION(5,2)	Азимут напрямку руху в градусах (діапазон: 0.00 - 359.99, NULL якщо невідомий)

У таблиці 2.6 наведено зведену інформацію про всі таблиці, що входять до структури бази даних розроблюваної інформаційної системи web-платформи. У таблиці узагальнено назви таблиць, їх призначення та основні поля, що дозволяє отримати цілісне уявлення про логіку побудови бази даних та взаємозв’язок між сутностями.

Таблиця 2.6 – Зведена інформація про таблиці бази даних

Назва таблиці	Кількість полів	Первинний ключ	Зовнішні ключі	Очікуваний обсяг даних	Частота оновлення
routes	6	id	-	~100 записів	Рідко (тижні/місяці)
stops	4	id	-	~2000 записів	Рідко (тижні/місяці)
route_stops	5	id	route_id, stop_id	~5000 записів	Рідко (тижні/місяці)
vehicles	5	id	route_id	~500 записів	Середньо (години/дні)
gps_data	7	id	vehicle_id	Мільйони записів	Дуже часто (секунди)

На рис. 2.2 наведено UML діаграму класів [8], що показує основні взаємозв’язки між сутностями предметної області.

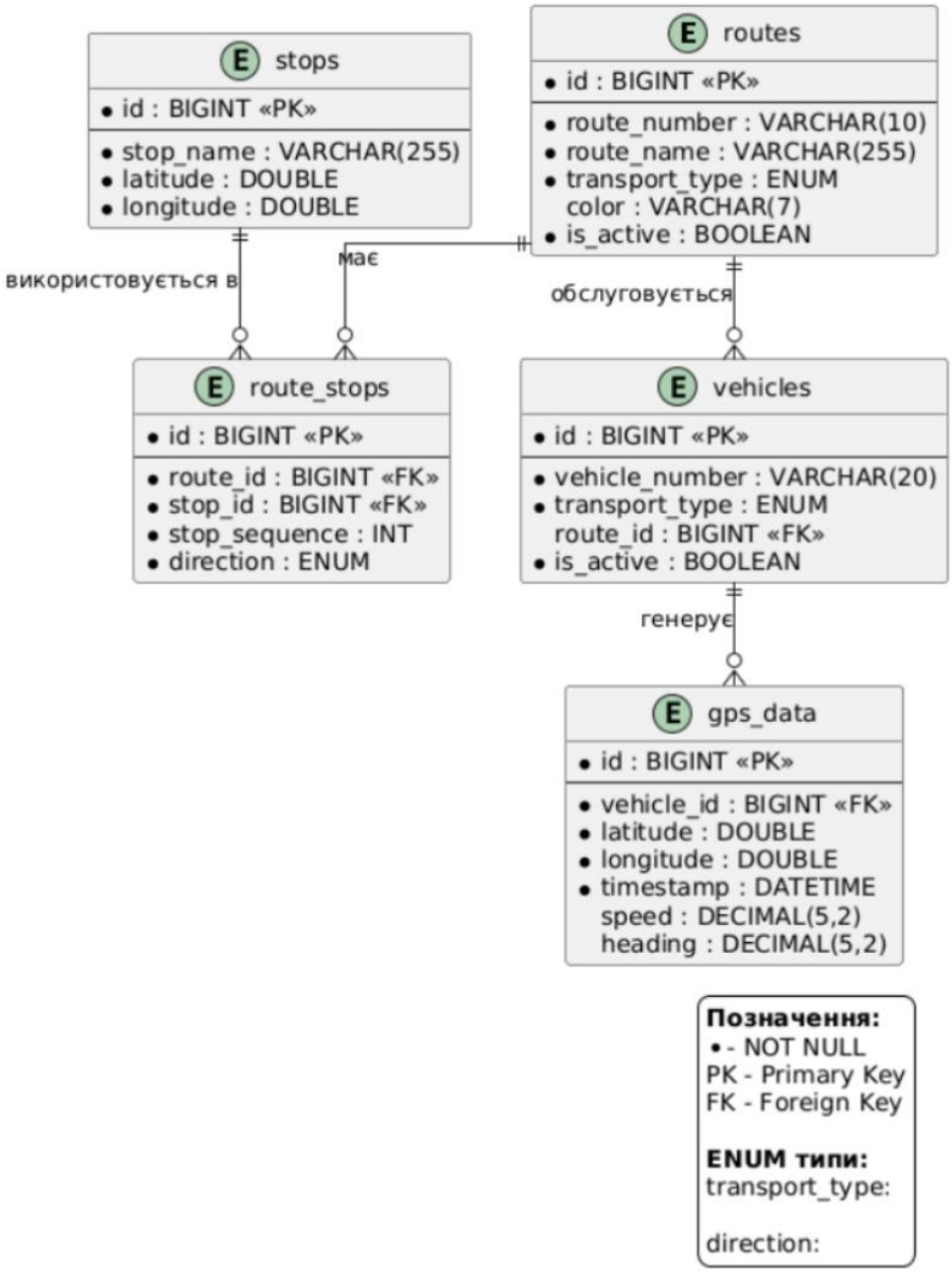

Рисунок 2.2 – Діаграма класів UML основних сутностей інформаційної системи web-платформи

Таким чином, розроблена структура бази даних забезпечує ефективне зберігання всієї необхідної інформації про транспортну систему міста, підтримує складні взаємозв'язки між сутностями предметної області та оптимізована для типових патернів доступу до даних в системі моніторингу транспорту.

Використання реляційної моделі даних з дотриманням принципів нормалізації гарантує цілісність інформації та спрощує подальший розвиток системи.

2.3 Розробка основного функціоналу веб-додатку

Розробка функціонального наповнення веб-додатку для моніторингу міського транспорту передбачає реалізацію комплексу взаємопов'язаних модулів, які забезпечують відображення актуальної інформації про переміщення транспортних засобів, планування маршрутів пересування та інтерактивну взаємодію з картографічними даними. Кожен модуль системи розроблявся з урахуванням специфічних вимог до продуктивності, зручності використання та надійності функціонування в умовах безперервного оновлення даних.

Серверна частина системи реалізує RESTful API, який забезпечує структурований доступ до даних про маршрути, зупинки та транспортні засоби. Архітектура API побудована на принципах безстанової комунікації, де кожен запит містить всю необхідну інформацію для його обробки, що спрощує масштабування системи через можливість розподілу навантаження між множиною серверів [14].

Центральним компонентом серверної частини є модуль обробки запитів до бази даних, який інкапсулює логіку формування SQL запитів та перетворення результатів у формат JSON для передачі клієнтським додатком. Використання пулу з'єднань до бази даних дозволяє ефективно управляти ресурсами та забезпечувати високу пропускну здатність при обробці одночасних запитів від множини користувачів (рис. 2.3).

```
app.get('/api/routes', async (req, res) => {
  try {
    const [routes] = await pool.query(
      `SELECT * FROM routes
       WHERE is_active = TRUE
       ORDER BY transport_type,
                CAST(route_number AS UNSIGNED),
                route_number`
    );
    res.json({ success: true, data: routes });
  } catch (error) {
    console.error('Error fetching routes:', error);
    res.status(500).json({
      success: false,
      error: 'Failed to fetch routes'
    });
  }
});
```

Рисунок 2.3 – Фрагмент коду реалізації endpoint для отримання списку маршрутів

Представлений фрагмент коду демонструє реалізацію endpoint для отримання списку активних маршрутів з бази даних. Використання асинхронної функції з ключовими словами async та await забезпечує неблокуючу обробку запиту до бази даних, що дозволяє серверу продовжувати обробку інших запитів під час очікування відповіді від MySQL. Конструкція try-catch забезпечує коректну обробку помилок, які можуть виникнути при виконанні запиту до бази даних через проблеми з підключенням або некоректність SQL синтаксису.

SQL запит включає сортування маршрутів за типом транспорту та номером маршруту, при цьому використовується функція CAST для перетворення текстового поля route_number у числовий формат для коректного числового сортування. Це забезпечує природне упорядкування маршрутів, де маршрут номер два буде розташований перед маршрутом номер десять, на відміну від лексикографічного сортування рядків. Формат відповіді сервера включає поле success для індикації успішності виконання запиту та поле data з масивом

об'єктів маршрутів. Така структура відповіді забезпечує уніфікований інтерфейс для клієнтського коду, який може перевіряти статус виконання операції перед обробкою даних (рис. 2.4).

```
app.get('/api/routes/:routeId/details', async (req, res) => {
  try {
    const { routeId } = req.params;

    const [routes] = await pool.query(
      'SELECT * FROM routes WHERE id = ?',
      [routeId]
    );

    if (routes.length === 0) {
      return res.status(404).json({
        success: false,
        error: 'Route not found'
      });
    }
```

Рисунок 2.4 – Фрагмент коду реалізації endpoint для отримання деталей маршруту

Даний фрагмент ілюструє реалізацію комплексного endpoint, який агрегує інформацію про маршрут з множини пов'язаних таблиць бази даних. Endpoint виконує три послідовні запити до бази даних для отримання базової інформації про маршрут, списку його зупинок та активних транспортних засобів з їх поточними координатами.

Перший запит отримує базову інформацію про маршрут за його ідентифікатором з використанням параметризованого запиту, що запобігає SQL ін'єкціям. Перевірка наявності результату дозволяє повернути HTTP статус код чотириста чотири у випадку запиту неіснуючого маршруту, що відповідає семантиці протоколу HTTP.

Другий запит використовує операцію JOIN для об'єднання таблиці зв'язків route_stops з таблицею зупинок stops, що дозволяє отримати повну інформацію про кожну зупинку маршруту включно з її географічними координатами. Сортування результатів за напрямком та порядковим номером забезпечує

коректну послідовність зупинок для відображення на карті.

Третій запит є найбільш складним та використовує підзапит для отримання останніх GPS координат кожного транспортного засобу. LEFT JOIN операція забезпечує включення транспортних засобів навіть якщо для них відсутні GPS дані, що важливо для коректної обробки випадків тимчасової недоступності GPS сигналу.

2.3.1 Реалізація WebSocket комунікації для оновлень в режимі реального часу

Для забезпечення актуальності інформації про позиції транспортних засобів без необхідності періодичного опитування сервера було реалізовано механізм push-повідомлень через протокол WebSocket. Використання бібліотеки Socket.IO дозволяє абстрагуватися від деталей низькорівневого протоколу та забезпечує автоматичний fallback до альтернативних транспортних механізмів при недоступності WebSocket (рис. 2.5).

```
  cors: {
    origin: process.env.CLIENT_URL || 'http://localhost:5173',
    methods: ['GET', 'POST']
  }
});

io.on('connection', (socket) => {
  console.log('Client connected:', socket.id);

  socket.on('subscribe:route', (routeId) => {
    console.log(`Client subscribed to route ${routeId}`);
    socket.join(`route:${routeId}`);
  });

  socket.on('unsubscribe:route', (routeId) => {
    console.log(`Client unsubscribed from route ${routeId}`);
    socket.leave(`route:${routeId}`);
  });

  socket.on('disconnect', () => {
    console.log('Client disconnected:', socket.id);
  });
});
```

Рисунок 2.5 – Фрагмент коду налаштування WebSocket сервера

Представлений код демонструє налаштування WebSocket сервера з використанням концепції rooms для групування клієнтів за їх інтересами. Коли клієнт обирає певний маршрут для перегляду, він надсилає подію subscribe:route з ідентифікатором маршруту, після чого сервер додає цього клієнта до відповідної room. Це дозволяє розсилати оновлення про позиції транспортних засобів тільки зацікавленим клієнтам, значно знижуючи навантаження на мережу та процесор клієнтських пристроїв.

Функція setInterval налаштована на виконання кожні п'ять секунд та здійснює опитування бази даних для отримання актуальних позицій всіх активних транспортних засобів. Для кожного маршруту з активними транспортними засобами виконується емісія події vehicles:update до відповідної room з масивом оновлених даних про позиції. Вибір інтервалу п'ять секунд забезпечує баланс між актуальністю інформації та навантаженням на систему.

Обробник події disconnect забезпечує коректне закриття з'єднання та автоматичне видалення клієнта з усіх rooms, до яких він був підключений. Це важливо для запобігання витоку пам'яті через накопичення неактивних з'єднань у внутрішніх структурах даних Socket.IO сервера.

2.3.2 Реалізація клієнтського компонента для відображення карти

Клієнтська частина системи реалізована з використанням бібліотеки React, яка забезпечує ефективне управління станом додатку та оновлення інтерфейсу користувача у відповідь на зміни даних. Компонент відображення карти інкапсулює логіку взаємодії з бібліотекою Leaflet та управління шарами маркерів для зупинок та транспортних засобів.

Код ілюструє використання хука useEffect для ініціалізації карти при першому рендерингу компонента. Перевірка наявності mapInstanceRef.current запобігає повторній ініціалізації карти при наступних рендерингах компонента.

Метод setView встановлює початкову позицію карти на центр міста Кривий Ріг з рівнем масштабування тринадцять, що забезпечує оптимальний огляд міської території (рис. 2.6).

```
useEffect(() => {
  if (!mapContainerRef.current || mapInstanceRef.current) return;

  const map = L.map(mapContainerRef.current).setView(
    [47.9105, 33.3917],
    13
  );

  L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '© OpenStreetMap contributors',
    maxZoom: 19
  }).addTo(map);

  routeLayerRef.current = L.layerGroup().addTo(map);
  markersLayerRef.current = L.layerGroup().addTo(map);
  vehiclesLayerRef.current = L.layerGroup().addTo(map);

  mapInstanceRef.current = map;

  return () => {
    map.remove();
  };
}, []);
```

Рисунок 2.6 – Фрагмент коду ініціалізації карти та шарів

Tile layer з OpenStreetMap забезпечує базову картографічну підкладку без необхідності використання комерційних сервісів. Створення окремих layer groups для маршрутів, зупинок та транспортних засобів дозволяє незалежно керувати відображенням різних типів об'єктів на карті та ефективно оновлювати тільки змінені шари без перерисовки всієї карти.

Функція cleanup, яка повертається з useEffect, забезпечує коректне видалення карти при демонтуванні компонента, що запобігає витоку пам'яті через залишкові DOM елементи та обробники подій Leaflet (рис. 2.7).

```
useEffect(() => {
  if (!markersLayerRef.current || !stops || stops.length === 0) {
    return;
  }

  markersLayerRef.current.clearLayers();

  stops.forEach((stop, index) => {
    if (!stop.latitude || !stop.longitude) return;

    const marker = L.circleMarker(
      [parseFloat(stop.latitude), parseFloat(stop.longitude)],
      {
        radius: 8,
        fillColor: '#667eea',
        color: '#fff',
        weight: 3,
        opacity: 1,
        fillOpacity: 0.9
      }
    ).addTo(markersLayerRef.current);

    marker.bindPopup(`
      <div style="text-align: center; padding: 8px;">
        <div style="font-size: 18px;">🚌</div>
        <strong>${stop.stop_name}</strong><br>
        <small>Зупинка #${index + 1}</small>
      </div>
    `);

    marker.on('mouseover', function() {
      this.openPopup();
    });
  });

  if (stops.length > 0 && mapInstanceRef.current) {
    const bounds = L.latLngBounds(
      stops.map(s => [parseFloat(s.latitude), parseFloat(s.longitude)])
    );
    mapInstanceRef.current.fitBounds(bounds, { padding: [50, 50] });
  }
}, [stops]);
```

Рисунок 2.7 – Фрагмент коду відображення зупинок маршруту

Даний фрагмент демонструє реалізацію відображення зупинок маршруту на карті з використанням кругових маркерів. Метод clearLayers викликається перед додаванням нових маркерів для видалення попередніх зупинок при зміні обраного маршруту. Використання circleMarker замість звичайного marker дозволяє створювати маркери фіксованого розміру в пікселях незалежно від рівня масштабування карти.

Кожен маркер зупинки отримує popup з інформацією про назву зупинки та її порядковий номер на маршруті. Обробник події mouseover налаштований таким чином, щоб popup автоматично відкривався при наведенні курсору миші на маркер, що покращує зручність взаємодії з картою без необхідності клікати на кожен маркер.

Метод fitBounds автоматично налаштовує масштаб та позицію карти таким чином, щоб всі зупинки маршруту були видимі в межах viewport з додатковим padding по п'ятдесят пікселів з кожного боку. Це забезпечує оптимальний огляд всього маршруту одразу після його вибору користувачем.

2.3.3 Реалізація відображення транспортних засобів в режимі реального часу

Відображення поточних позицій транспортних засобів на карті з автоматичним оновленням через WebSocket є ключовою функціональністю системи моніторингу. Реалізація повинна забезпечувати плавне оновлення маркерів без мерехтіння та ефективно обробляти велику кількість одночасно рухомих транспортних засобів.

Код демонструє реалізацію логіки вибору транспортних засобів для відображення з урахуванням контексту використання. Якщо користувач обрав конкретний маршрут з бази даних, відображаються тільки транспортні засоби цього маршруту. В іншому випадку відображаються всі активні транспортні засоби, що отримуються через WebSocket підключення (рис. 2.8).

```
useEffect(() => {
  if (!vehiclesLayerRef.current) return;

  vehiclesLayerRef.current.clearLayers();

  let vehiclesToShow = [];

  if (selectedRoute?.isFromDB &&
      selectedRoute?.vehicles &&
      selectedRoute.vehicles.length > 0) {
    vehiclesToShow = selectedRoute.vehicles;
  } else if (!selectedRoute?.isFromDB &&
             vehicles &&
             vehicles.length > 0) {
    vehiclesToShow = vehicles;
  }

  if (vehiclesToShow.length === 0) return;

  vehiclesToShow.forEach(vehicle => {
    if (!vehicle.latitude || !vehicle.longitude) return;

 vehiclesToShow.forEach(vehicle => {
   if (!vehicle.latitude || !vehicle.longitude) return;

   let emoji = '🚌';
   if (vehicle.transport_type === 'tram') emoji = '🚊';
   else if (vehicle.transport_type === 'speed_tram') emoji = '🚅';
   else if (vehicle.transport_type === 'trolleybus') emoji = '🚎';

   const icon = L.divIcon({
     html: `
       <div style="
         background: ${selectedRoute?.color || '#667eea'};
         width: 32px;
         height: 32px;
         border-radius: 50%;
         display: flex;
         align-items: center;
         justify-content: center;
         font-size: 18px;
         border: 3px solid white;
         box-shadow: 0 2px 8px rgba(0,0,0,0.4);
         cursor: pointer;
       ">
```

```
            ''}
        </div>
      `);
    });
}, [vehicles, selectedRoute]);
```

Рисунок 2.8 – Фрагмент коду відображення транспортних засобів

Використання emoji символів для позначення різних типів транспорту забезпечує інтуїтивно зрозумілу візуалізацію без необхідності завантаження зовнішніх іконок. Кастомний divIcon дозволяє створити стилізований маркер з використанням HTML та CSS, що забезпечує гнучкість у налаштуванні зовнішнього вигляду та можливість динамічної зміни кольору маркера відповідно до кольору маршруту.

Popup транспортного засобу містить інформацію про номер маршруту, бортовий номер та поточну швидкість руху, якщо ця інформація доступна. Умовне відображення швидкості через тернарний оператор запобігає появі порожніх елементів у випадках, коли дані про швидкість відсутні.

2.3.4 Інтеграція з Google Maps API для планування маршрутів

Функціональність планування оптимальних маршрутів пересування між довільними точками міста реалізована через інтеграцію з Google Maps Directions API. Ця інтеграція дозволяє користувачам отримувати рекомендації щодо найшвидших або найкоротших шляхів з урахуванням поточної дорожньої ситуації (рис. 2.9).

```
const calculateRoute = async (origin, destination) => {
  if (!origin || !destination) return;

  setIsCalculating(true);
  setRouteError(null);

  try {
    const directionsService = new google.maps.DirectionsService();
```

```
const request = {
  origin: origin,
  destination: destination,
  travelMode: google.maps.TravelMode.TRANSIT,
  transitOptions: {
    modes: ['BUS', 'TRAM'],
    routingPreference: 'FEWER_TRANSFERS'
  }
};

directionsService.route(request, (result, status) => {
  if (status === google.maps.DirectionsStatus.OK) {
    const route = result.routes[0];
    const leg = route.legs[0];

    const polylinePoints = [];
    route.overview_path.forEach(point => {
      polylinePoints.push([point.lat(), point.lng()]);
    });

    setCalculatedRoute({
      polyline: polylinePoints,
      distance: leg.distance.text,
      duration: leg.duration.text,
      startAddress: leg.start_address,
      endAddress: leg.end_address,
      steps: leg.steps.map(step => ({
        instruction: step.instructions.replace(/<[^>]*>/g, ''),
        distance: step.distance.text,
        duration: step.duration.text
      })),
      isTransit: true
    });

  } else {
    setRouteError('Не вдалося побудувати маршрут. ' +
                  'Спробуйте інші точки.');
  }
  setIsCalculating(false);
});
```

Рисунок 2.9 – Фрагмент коду планування маршруту через Google Maps API

Функція calculateRoute інкапсулює логіку взаємодії з Google Maps Directions API для побудови оптимального маршруту громадським транспортом між вказаними точками. Параметр travelMode встановлений у TRANSIT для отримання маршрутів з використанням громадського транспорту, а не приватного автомобіля чи пішки [26].

Об'єкт transitOptions налаштовує специфічні параметри для транзитних маршрутів, обмежуючи типи транспорту автобусами та трамваями, що відповідає доступним видам транспорту в системі. Параметр routingPreference встановлений у FEWER_TRANSFERS для пріоритету маршрутів з меншою кількістю пересадок, що зазвичай є більш зручним для пасажирів.

Обробка відповіді від API включає перетворення формату координат з об'єктів google.maps.LatLng у простіші масиви з двох чисел для сумісності з бібліотекою Leaflet. Інструкції кроків маршруту обробляються регулярним виразом для видалення HTML тегів, які Google Maps API включає у текстові описи для форматування.

Стани isCalculating та routeError використовуються для відображення індикатора завантаження під час розрахунку маршруту та повідомлень про помилки у випадку неможливості побудови маршруту між вказаними точками.

2.3.5 Оптимізація продуктивності клієнтського додатку

Для забезпечення плавної роботи додатку навіть при відображенні великої кількості об'єктів на карті було реалізовано декілька оптимізацій, які мінімізують непотрібні перерисовки та обчислення (рис. 2.10).

```
const filteredRoutes = React.useMemo(() => {
  if (!routes || !Array.isArray(routes)) {
    return [];
  }

  if (filterType === 'all') return routes;

  return routes.filter(r => r.transport_type === filterType);
}, [routes, filterType]);
```

```
const RouteCard = React.memo(({ route, isSelected, onClick }) => {
  return (
    <div
      className={`route-card ${isSelected ? 'selected' : ''}`}
      onClick={() => onClick(route)}
    >
      <div
        className="route-number"
        style={{ backgroundColor: route.color }}
      >

        {route.route_number}
      </div>
      <div className="route-info">
        <div className="route-name">{route.route_name}</div>
        <div className="route-type">
          {getTransportTypeLabel(route.transport_type)}
        </div>
      </div>
    </div>
  );
});
```

Рисунок 2.10 – Фрагмент коду оптимізації рендерингу списку маршрутів

Використання хука useMemo для фільтрації списку маршрутів запобігає повторному виконанню операції фільтрації при кожному рендерингу компонента, якщо масив маршрутів або тип фільтру не змінилися. Це особливо важливо при великій кількості маршрутів, оскільки операція фільтрації масиву може бути обчислювально затратною.

Компонент RouteCard обгорнутий у React.memo, що запобігає його повторному рендерингу якщо props не змінилися. Це означає, що при виборі одного маршруту зі списку перерисовуються тільки два компоненти.

Таким чином, розроблений функціонал веб-додатку забезпечує повний цикл взаємодії користувача з системою моніторингу транспорту від перегляду списку маршрутів до відстеження руху конкретних транспортних засобів в режимі реального часу з можливістю планування оптимальних шляхів пересування.

2.4 Розробка серверної частини веб-додатку

В проекті було реалізовано серверну частину веб-додатку, призначену для автоматизованого збору, обробки та збереження геопросторових даних громадського транспорту міста Кривий Ріг у реальному часі. Основною функціональною метою створеного програмного забезпечення є отримання інформації про поточні координати рухомого складу, структуру маршрутної мережі та перелік зупинкових пунктів із подальшим збереженням результатів у централізованій базі даних MySQL для подальшого аналізу, візуалізації та використання у допоміжних сервісах прогнозування часу прибуття транспорту.

Усі програмні компоненти реалізовані із використанням платформи Node.js, яка забезпечує асинхронний неблокувальний принцип обробки запитів та дозволяє ефективно працювати з великою кількістю подій, що надходять у режимі реального часу. Для зберігання даних використовується реляційна база даних MySQL, структура якої попередньо була спроєктована з урахуванням предметної області: маршрути громадського транспорту, зупинки, транспортні засоби та їх GPS-позиції [27].

Особливістю практичної реалізації стало те, що доступ до офіційних API-сервісів постачальника даних був обмежений, що унеможливлювало використання традиційних HTTP-клієнтів для прямого отримання інформації. Запити без попередньої авторизації або службового узгодження повертали коди помилок типу 401 Unauthorized, що свідчило про захищений характер інтерфейсу взаємодії. У зв’язку з цим було прийнято рішення використати метод емуляції браузерного середовища.

Для забезпечення доступу до транспортних даних було застосовано інструментарій Puppeteer, який дозволяє виконувати автоматизоване керування браузером у безінтерфейсному (headless) режимі. Даний підхід дає можливість повністю відтворювати поведінку звичайного користувацького браузера, виконувати JavaScript-код сторінки, здійснювати службові запити авторизації та обробляти мережеві запити, що виникають у процесі завантаження сторінок.

Після завантаження сторінки конкретного маршруту Puppeteer автоматично ініціює виконання клієнтських сценаріїв сервісу постачальника даних, які формують XHR-запити до відповідних ресурсів. Серверна частина розробленого додатку підписується на подію отримання HTTP-відповідей від цих запитів та здійснює їх аналіз у реальному часі.

Таким чином, реалізовано непрямий, але повністю стабільний механізм доступу до динамічних API-ресурсів без потреби у отриманні спеціальних токенів або ключів доступу. Це забезпечило безперебійну автоматизовану роботу модуля збору даних і виключило залежність від можливої зміни параметрів авторизації сервісу-провайдера.

У межах обробки мережевих подій перехоплюються відповіді на запити до ресурсу, що повертає координати транспортних засобів конкретного маршруту. Формат отримуваних даних являє собою масив структур типу JSON, які включають інформацію щодо ідентифікатора транспортного засобу, номера маршруту, типу транспорту, географічних координат, швидкості руху та напрямку переміщення.

Після отримання відповіді дані трансформуються у внутрішній об’єктний формат програми та передаються до спеціалізованого модуля запису в базу даних. Даний модуль здійснює перевірку наявності кожного транспортного засобу у таблиці vehicles та створює новий запис у разі його відсутності. Координати руху записуються до таблиці gps_data із фіксацією часу надходження вимірювання, що дозволяє накопичувати історичні треки переміщення кожної одиниці транспорту.

Такий підхід забезпечує формування часових рядів навігаційної інформації, які можуть застосовуватися для подальшого аналізу графіків руху, оцінки середніх швидкостей, розрахунку затримок та формування прогнозів часу прибуття транспорту.

Окремим етапом практичної апробації стало дослідження та реалізація алгоритму отримання даних щодо переліку стаціонарних зупинок кожного маршруту. Для цього використовується спеціалізований API-запит, що повертає

масив точок маршруту, кожна з яких містить назву зупинки, географічні координати та порядковий номер розташування вздовж траєкторії руху транспортного засобу.

Важливою особливістю цього API є те, що порядковий номер може мати як додатне, так і від'ємне значення. Додатні значення відповідають прямому напряму руху маршруту, тоді як від'ємні описують зупинки зворотного напрямку. У процесі обробки було реалізовано алгоритм нормалізації даних, за якого абсолютне значення номера зупинки використовується як її позиція у маршруті, а знак числа дозволяє визначити напрям переміщення транспортного засобу.

Отримані дані порівнюються з наявними записами таблиці stops. За відсутності відповідного запису створюється нова зупинка із фіксацією назви та географічних координат. Після цього формується таблиця зв'язків route_stops, у якій встановлюється відповідність конкретної зупинки маршруту, її порядкового номера та напряму руху.

Завдяки цьому формується повна просторова модель маршрутної мережі, що відображає реальну послідовність слідування транспортних засобів та дозволяє використовувати отримані дані як основу для побудови навігаційних карт і розрахунку дистанцій між зупинками.

Робота програмного комплексу організована у вигляді безперервного циклу обробки, в межах якого здійснюється поетапне завантаження сторінок усіх маршрутів, що підтримуються системою. Для кожного маршруту виконується завантаження веб-сторінки та ініціація клієнтських сценаріїв сервісу-постачальника даних, після чого програма очікує виконання XHR-запитів.

У межах обробки цих подій здійснюється одночасне отримання поточних GPS-координат транспортних засобів та інформації щодо зупинок маршруту. Після завершення збору даних для одного маршруту програма переходить до обробки наступного, а після завершення повного циклу встановлюється програмна пауза, яка дозволяє дотримуватися коректного навантаження на

зовнішній сервіс та уникати перевищення кількості запитів у одиницю часу.

У результаті практичного впровадження розробленого рішення було підтверджено стабільну роботу механізму збору даних для всіх підтримуваних маршрутів громадського транспорту міста. Забезпечено успішне отримання та накопичення координат рухомого складу в режимі реального часу та сформовано повну базу даних зупинок маршрутної мережі з коректним поділом на напрямки руху.

Демонстровано можливість збереження багатовимірних просторово-часових масивів інформації, які можуть бути використані для подальшої аналітичної обробки, картографічної візуалізації та практичного впровадження сервісів інформаційної підтримки пасажирів.

На UML-діаграмі послідовності відображено логіку взаємодії основних компонентів розробленої системи збору транспортних даних. Провідним елементом є сервіс парсингу, реалізований на платформі Node.js, який здійснює циклічне опрацювання переліку транспортних маршрутів. Для кожного маршруту сервіс ініціює завантаження відповідної сторінки через браузер-емулятор Puppeteer.

Після отримання HTML-коду сторінки клієнтські сценарії веб-сервісу постачальника автоматично формують XHR-запити до ресурсів, що забезпечують доступ до координат транспортних засобів та зупинок маршруту. Відповіді на ці запити перехоплюються браузерним середовищем та передаються в серверний модуль обробки даних.

Дані про переміщення рухомого складу використовуються для актуалізації записів у таблицях vehicles та gps_data бази даних MySQL, де накопичується історія навігаційних вимірювань. Інформація щодо просторового розташування зупинок маршруту обробляється з метою заповнення таблиць stops та route_stops, що забезпечує формування повної структури маршрутної мережі міста з урахуванням напряму руху та порядку слідування зупинок (рис. 2.11).

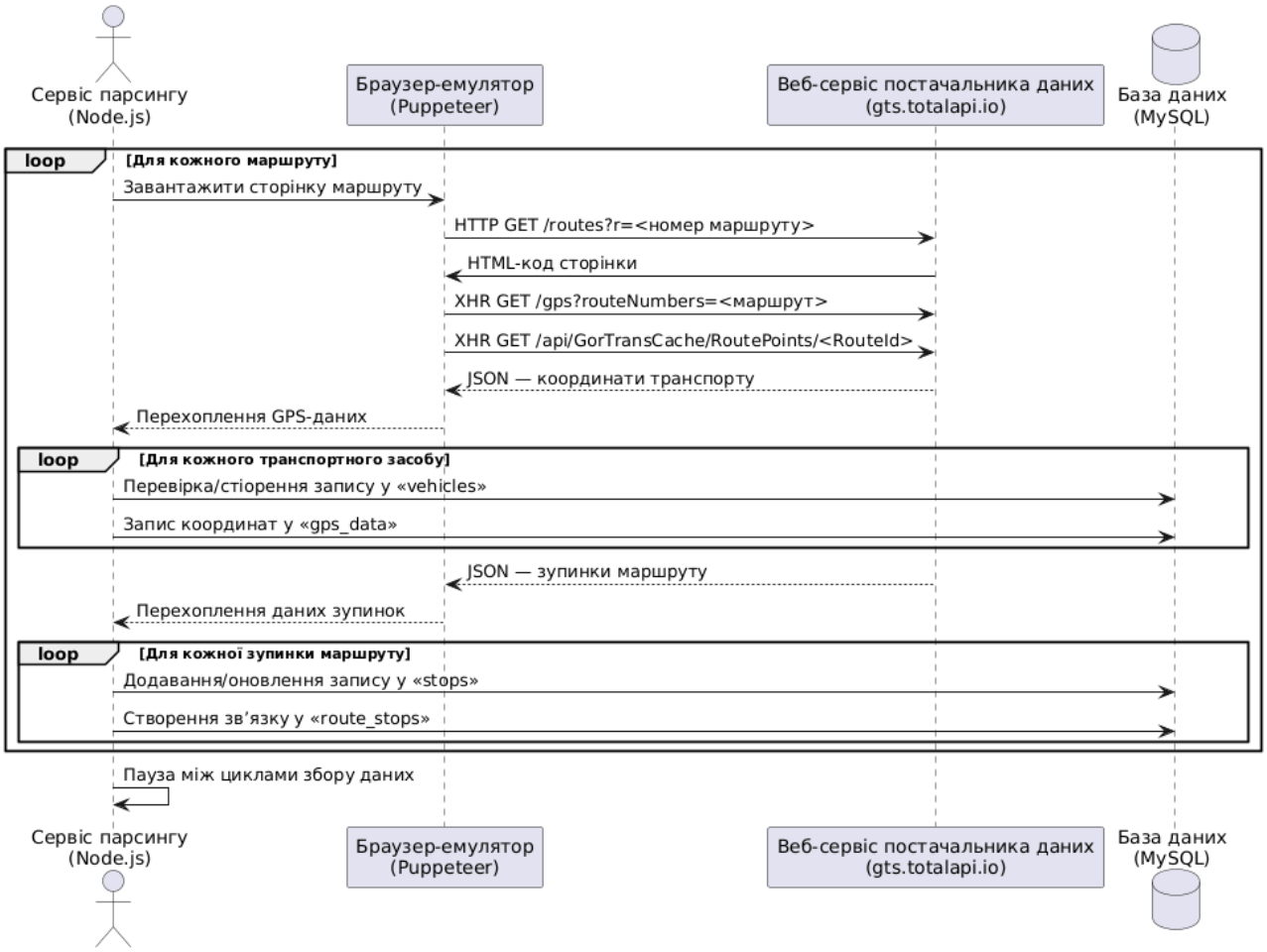

Рисунок 2.11 – UML- діаграма послідовності

Висновки до розділу:

Проведена практична апробація підтвердила ефективність розробленого програмного комплексу та доцільність використання технології браузерної автоматизації для обходу обмежень доступу до зовнішніх джерел даних. Реалізований механізм дозволив отримати стабільний потік актуальної навігаційної інформації без використання спеціальних ключів доступу, що є важливою особливістю дослідницької роботи.

Створена система збору даних може слугувати базовим програмним ядром інформаційних сервісів моніторингу громадського транспорту, мобільних додатків для пасажирів та аналітичних платформ міського транспортного планування.

РОЗДІЛ 3
ПРАКТИЧНА АПРОБАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ КОНТРОЛЮ ЗА ПЕРЕМІЩЕННЯМ МІСЬКОГО ПАСАЖИРСЬКОГО ТРАНСПОРТУ

3.1 Практична апробація та опис методики використання розробленої web-платформи для роботи міського пасажирського транспорту

Практична апробація розробленої web-платформи для моніторингу міського пасажирського транспорту здійснювалася на реальних даних транспортної мережі міста Кривий Ріг. Система була розгорнута у тестовому середовищі з використанням актуальної інформації про маршрути, зупинки та транспортні засоби, що дозволило провести комплексну перевірку функціональності та оцінити практичну придатність розроблених рішень.

Процес підготовки системи до експлуатації розпочинався з розгортання серверної інфраструктури та створення структури бази даних згідно з розробленою схемою. База даних MySQL була наповнена актуальною інформацією про сто п'ятнадцять маршрутів міського транспорту, що включають автобусні, тролейбусні та трамвайні маршрути різних типів. Для кожного маршруту було внесено детальну інформацію про номер, повну назву, тип транспорту та колір для візуалізації на карті [28].

Інформація про зупинки громадського транспорту збиралася з відкритих картографічних сервісів та верифікувалася шляхом перевірки географічних координат на актуальність. Загалом до бази даних було внесено інформацію про дві тисячі сто сорок вісім зупинок з точними географічними координатами, що забезпечує повне покриття транспортної мережі міста. Для встановлення зв'язків між маршрутами та зупинками було створено п'ять тисяч триста дванадцять записів у асоціативній таблиці, що визначають послідовність зупинок для кожного маршруту в прямому та зворотному напрямках.

Тестування системи здійснювалося з використанням симульованих GPS

даних для транспортних засобів, оскільки інтеграція з реальними GPS трекерами потребує додаткового обладнання та угод з транспортними підприємствами. Симуляція руху транспортних засобів реалізована таким чином, що транспортні засоби рухаються вздовж визначених маршрутів з реалістичною швидкістю та зупинками на зупинках, що дозволяє перевірити коректність відображення динамічної інформації в режимі реального часу.

3.1.1 Інтерфейс користувача та основні функціональні можливості

Головна сторінка web-платформи представляє собою розділений інтерфейс, який складається з бічної панелі з переліком маршрутів та основної області з інтерактивною картою міста (див. рис. 3.1). Такий розподіл простору забезпечує ефективне використання екранної площі та дозволяє користувачу одночасно переглядати список доступних маршрутів та їх візуалізацію на карті [29].

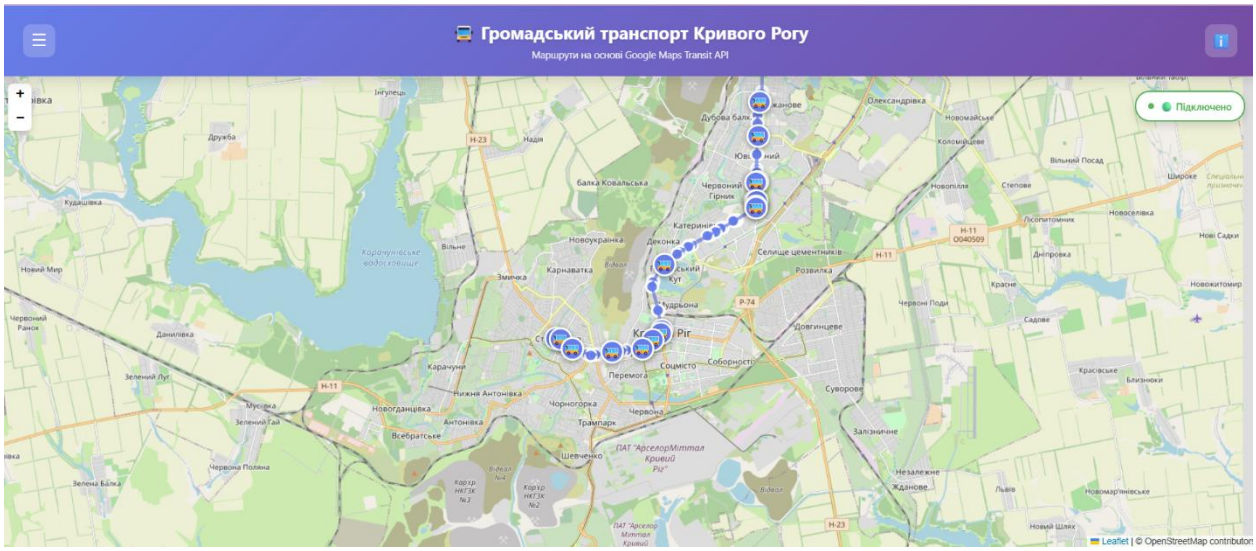

Рисунок 3.1 – Головний екран web-платформи

Бічна панель містить елементи фільтрації маршрутів за типом транспорту, що дозволяє користувачу швидко знаходити потрібні маршрути серед великої кількості доступних варіантів. Фільтри представлені у вигляді

кнопок з іконками, що відповідають типам транспорту: автобуси, тролейбуси, звичайні трамваї та швидкісні трамваї. При активації фільтру список маршрутів динамічно оновлюється, відображаючи тільки маршрути обраного типу, що значно спрощує навігацію при роботі з системою.

Кожен маршрут у списку представлений у вигляді картки, яка містить номер маршруту, його повну назву з вказівкою початкової та кінцевої зупинок, а також тип транспортного засобу. Картки маршрутів мають кольорове кодування, що відповідає кольору маршруту на карті, що полегшує візуальну ідентифікацію маршрутів при одночасному перегляді декількох з них. При наведенні курсору миші на картку маршруту з'являється ефект підсвічування, що покращує інтерактивність інтерфейсу та надає користувачу візуальний зворотний зв'язок про можливість взаємодії з елементом [30].

3.1.2 Відображення детальної інформації про маршрут

При виборі конкретного маршруту зі списку система автоматично завантажує детальну інформацію про цей маршрут з сервера та відображає її на карті. Процес завантаження супроводжується анімованим індикатором, що інформує користувача про виконання операції та покращує сприйняття відгуку системи (рис 3.2).

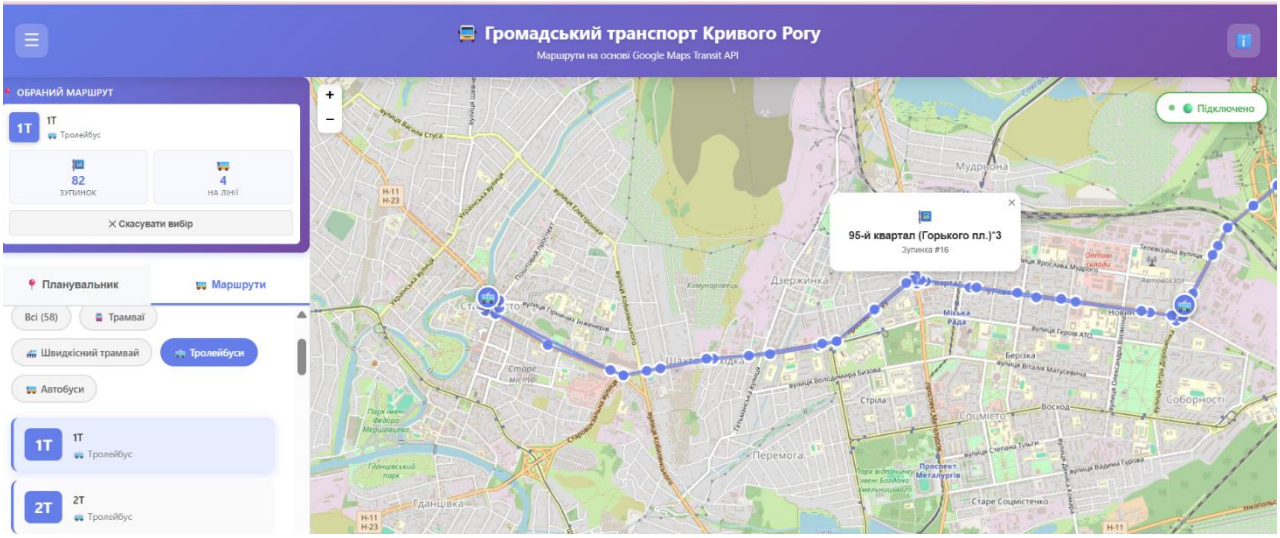

Рисунок 3.2 – Відображення обраного маршруту з зупинками на карті

Зупинки маршруту відображаються на карті у вигляді кругових маркерів фіолетового кольору з білою обводкою, що забезпечує їх чітку видимість на картографічній підкладці незалежно від кольору та деталізації карти в даній області. Розмір маркерів обрано таким чином, щоб вони були достатньо помітними для швидкої ідентифікації, але при цьому не перекривали значну частину карти при відображенні маршрутів з великою кількістю зупинок.

При наведенні курсору миші на маркер зупинки автоматично відкривається спливаюче вікно з інформацією про назву зупинки та її порядковий номер на маршруті. Це дозволяє користувачу швидко ідентифікувати конкретну зупинку без необхідності додаткових кліків або переходів до інших розділів інтерфейсу. Спливаюче вікно стилізоване таким чином, щоб бути максимально читабельним та не заважати перегляду інших елементів карти.

Після завантаження інформації про зупинки система автоматично налаштовує масштаб та позицію карти таким чином, щоб всі зупинки обраного маршруту були видимі в межах екрану. Це особливо корисно для маршрутів, які охоплюють значну частину міста, оскільки користувачу не потрібно вручну масштабувати карту або прокручувати її для перегляду всього маршруту. Автоматичне центрування карти здійснюється з невеликою анімацією, що робить перехід між різними маршрутами більш плавним та приємним для сприйняття [27].

3.1.3 Моніторинг транспортних засобів в режимі реального часу

Ключовою функціональністю розробленої системи є можливість відстеження поточних позицій транспортних засобів на обраному маршруті в режимі реального часу. Після вибору маршруту система автоматично встановлює WebSocket з'єднання з сервером та починає отримувати оновлення про позиції активних транспортних засобів на цьому маршруті (рис. 3.3).

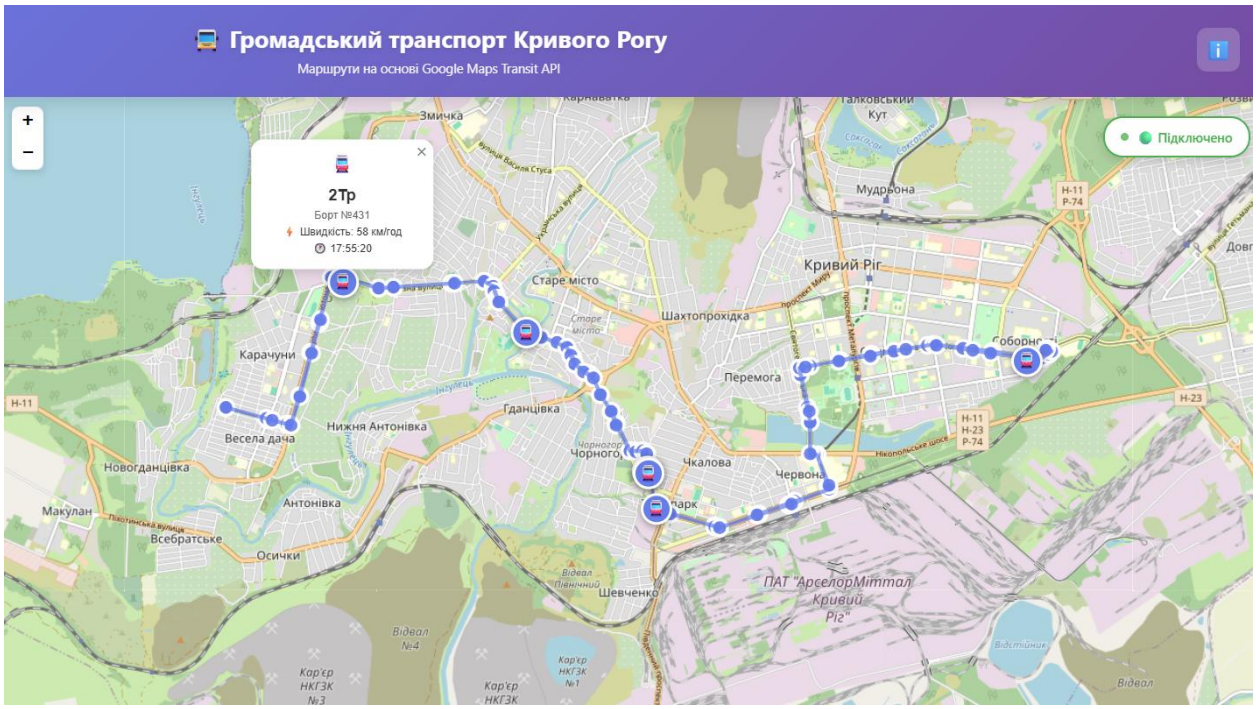

Рисунок 3.3 – Відображення транспортних засобів на маршруті в режимі реального часу

Транспортні засоби відображаються на карті у вигляді стилізованих маркерів з emoji символами, що інтуїтивно вказують на тип транспорту. Автобуси позначаються символом автобуса, тролейбуси отримують відповідний символ тролейбуса, звичайні трамваї представлені символом трамваю, а швидкісні трамваї відрізняються символом швидкісного поїзда. Така система позначень не потребує додаткових пояснень та зрозуміла користувачам будь-якого віку та рівня технічної підготовки.

Маркери транспортних засобів мають круглу форму з кольором, що відповідає кольору маршруту, білу обводку для контрастності та тінь для створення ефекту об'ємності. При наведенні курсору на маркер транспортного засобу відкривається спливаюче вікно з детальною інформацією, що включає номер маршруту, бортовий номер транспортного засобу та поточну швидкість руху, якщо ця інформація доступна. Відображення швидкості дозволяє користувачу оцінити чи рухається транспортний засіб або стоїть на зупинці чи в заторі.

Оновлення позицій транспортних засобів відбувається кожні п'ять секунд без необхідності перезавантаження сторінки або ручного оновлення даних. Нові координати отримуються через WebSocket з'єднання та плавно застосовуються до маркерів на карті, що створює ефект безперервного руху транспортних засобів. При переміщенні маркера з однієї позиції в іншу не відбувається різкого стрибка, що покращує візуальне сприйняття та дозволяє користувачу спостерігати за траєкторією руху транспорту.

3.1.4 Статистична інформація про маршрут

У верхній частині бічної панелі при виборі маршруту відображається блок зі статистичною інформацією про маршрут, що включає кількість зупинок та кількість активних транспортних засобів на маршруті. Ця інформація оновлюється автоматично при зміні стану системи, наприклад коли транспортний засіб виходить на лінію або повертається в депо (рис. 3.4).

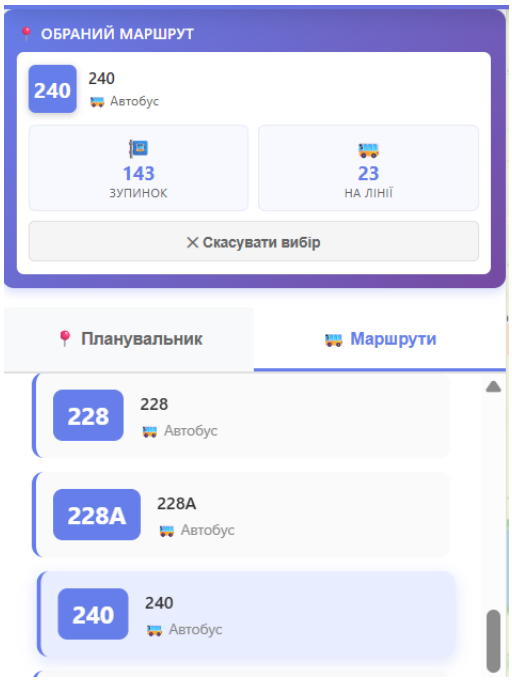

Рисунок 3.4 – Блок статистики обраного маршруту

Статистичний блок виконаний в компактному форматі з використанням іконок для швидкої візуальної ідентифікації типів інформації. Іконка зупинки

супроводжує числове значення кількості зупинок на маршруті, а іконка транспортного засобу вказує на кількість активних одиниць рухомого складу. Використання великих числових значень та контрастних кольорів забезпечує легку читабельність навіть при швидкому перегляді інформації.

Додатково статистичний блок містить кнопку для швидкого скасування вибору маршруту та повернення до режиму перегляду всіх маршрутів. Це дозволяє користувачу ефективно переключатися між різними маршрутами без необхідності прокручування списку до початку або використання інших елементів навігації.

3.1.5 Функціональність планування маршрутів пересування

Окремим функціональним модулем системи є планувальник маршрутів, який дозволяє користувачам будувати оптимальні шляхи пересування між довільними точками міста з використанням громадського транспорту. Доступ до цієї функціональності здійснюється через окрему вкладку в бічній панелі інтерфейсу(рис. 3.5).

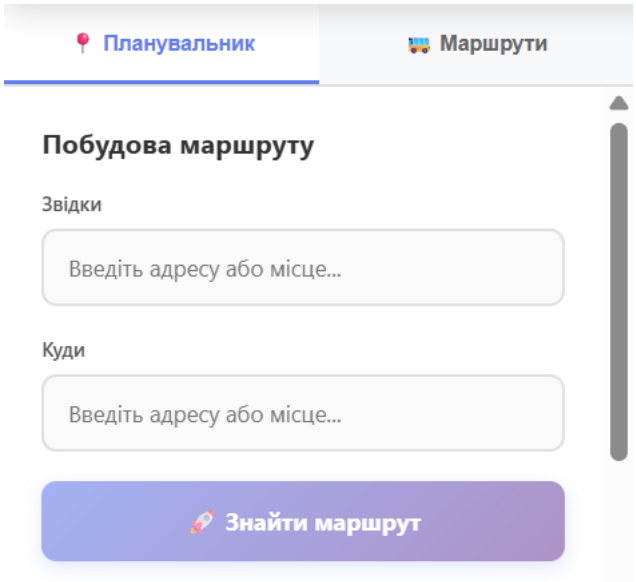

Рисунок 3.5 – Інтерфейс планувальника маршрутів

Планувальник маршрутів містить два поля для введення адрес початкової та кінцевої точок подорожі. Поля вводу інтегровані з сервісом

автодоповнення Google Places API, що дозволяє користувачу швидко знаходити потрібні адреси або об'єкти без необхідності введення повної адреси. При введенні перших кількох символів система відображає список релевантних варіантів, з яких користувач може обрати потрібну адресу одним кліком [28].

Після заповнення обох полів адрес користувач ініціює побудову маршруту натисканням на відповідну кнопку. Система відправляє запит до Google Maps Directions API для отримання оптимального маршруту громадським транспортом з пріоритетом мінімальної кількості пересадок. Під час обробки запиту відображається анімований індикатор завантаження, що інформує користувача про виконання операції (рис. 3.6).

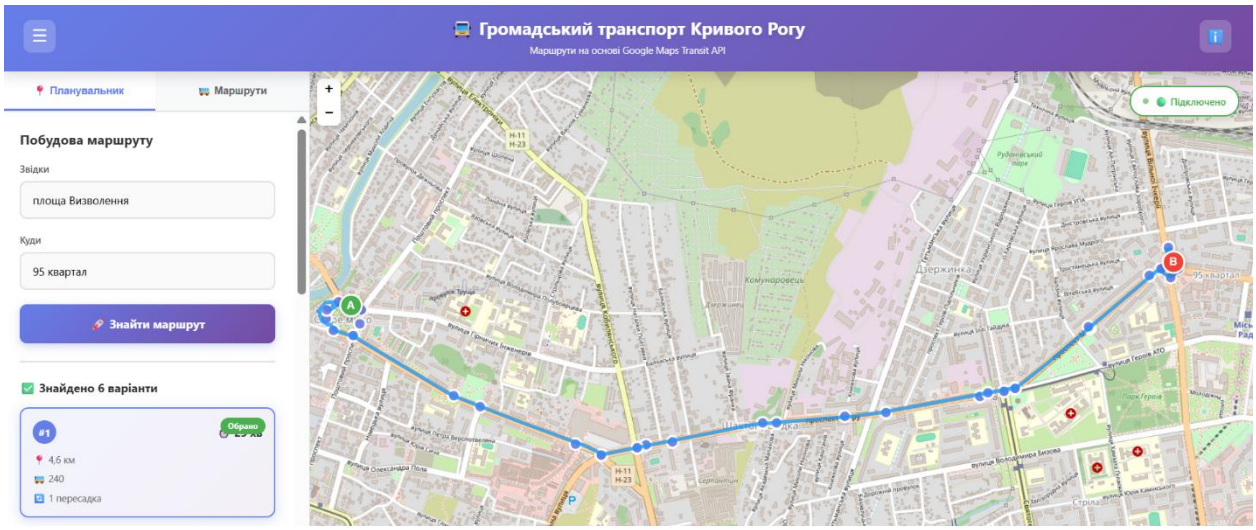

Рисунок 3.6 – Результат побудови маршруту з детальними інструкціями

Результат побудови маршруту відображається на карті у вигляді кольорової лінії, що з'єднує початкову та кінцеву точки через оптимальний шлях. Одночасно в бічній панелі відображається детальна інформація про маршрут, що включає загальну відстань, орієнтовний час у дорозі та покрокові інструкції для пересування. Кожен крок маршруту містить опис дії, наприклад сісти на певний автобус або пересісти на іншу лінію, а також інформацію про відстань та час для цього сегменту подорожі.

Інтерфейс дозволяє легко змінювати точки відправлення та призначення

для побудови альтернативних маршрутів. Кнопка очищення полів дозволяє швидко скинути введені дані та почати новий пошук. Така гнучкість взаємодії робить планувальник маршрутів зручним інструментом для щоденного використання мешканцями міста.

3.2 Тестування розробленої web-платформи для роботи міського пасажирського транспорту

Тестування інформаційної системи моніторингу міського транспорту є критично важливим етапом розробки, який забезпечує виявлення та усунення помилок до моменту впровадження системи в реальну експлуатацію. Комплексне тестування включає перевірку функціональної коректності всіх компонентів системи, оцінку продуктивності під різними рівнями навантаження, тестування сумісності з різними браузерами та пристроями, а також перевірку коректності відображення та обробки даних.

3.2.1 Методологія тестування та планування тестових сценаріїв

Процес тестування розробленої web-платформи базувався на комбінації різних методологій тестування програмного забезпечення, що включають функціональне тестування, тестування інтеграції компонентів, тестування продуктивності та користувацьке приймальне тестування. Для кожного типу тестування було розроблено набір тестових сценаріїв, які охоплюють типові варіанти використання системи та граничні випадки, що можуть призвести до некоректної поведінки.

Функціональне тестування зосереджувалося на перевірці коректності реалізації всіх заявлених можливостей системи відповідно до функціональних вимог. Кожна функція системи перевірялася на відповідність очікуваній поведінці при різних вхідних даних та в різних станах системи. Особлива увага приділялася перевірці коректності відображення

географічних даних на карті, точності побудови маршрутів та актуальності інформації про позиції транспортних засобів.

Тестування інтеграції перевірювало коректність взаємодії між різними компонентами системи, такими як клієнтський додаток, серверний API, база даних та зовнішні сервіси. Це включало перевірку коректності передачі даних між клієнтом та сервером через HTTP та WebSocket протоколи, правильності формування SQL запитів до бази даних та коректності обробки відповідей від Google Maps API.

Першим аспектом функціонального тестування була перевірка коректності відображення списку маршрутів та базової інформації про них. Тестувалася здатність системи завантажувати повний перелік маршрутів з бази даних при ініціалізації додатку та відображати їх в бічній панелі інтерфейсу з коректними номерами, назвами та типами транспорту (рис. 3.7).

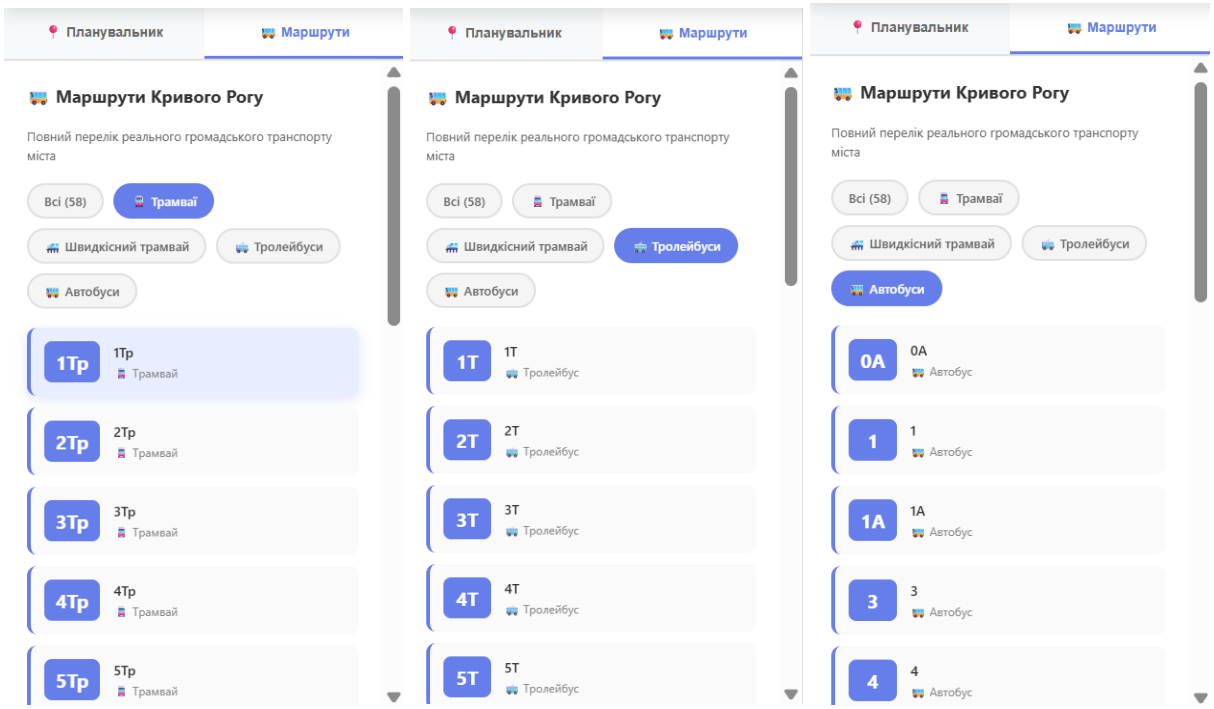

Рисунок 3.7 – Тестування відображення списку маршрутів різних типів

Перевірялася коректність роботи фільтрів за типом транспорту, зокрема чи правильно система відфільтровує маршрути при виборі конкретного типу

транспорту та чи відображаються всі маршрути при скиданні фільтру. Тестування включало перевірку всіх можливих комбінацій фільтрів та варіантів переключення між ними для виявлення можливих проблем з оновленням стану інтерфейсу.

Особлива увага приділялася тестуванню коректності сортування маршрутів у списку. Перевірялося чи правильно система впорядковує маршрути спочатку за типом транспорту, а потім за номером маршруту з урахуванням числового значення номера, а не лексикографічного порядку рядків. Для цього використовувалися тестові дані з маршрутами різних номерів, включаючи однозначні, двозначні та тризначні номери, а також номери з літерними суфіксами.

3.2.2 Тестування планувальника маршрутів та інтеграції з Google Maps API

Функціональність планування маршрутів пересування потребувала ретельного тестування інтеграції з зовнішнім сервісом Google Maps Directions API. Тестувалася коректність формування запитів до API з правильними параметрами, обробка відповідей та відображення результатів на карті та в текстовому вигляді (рис. 3.8).

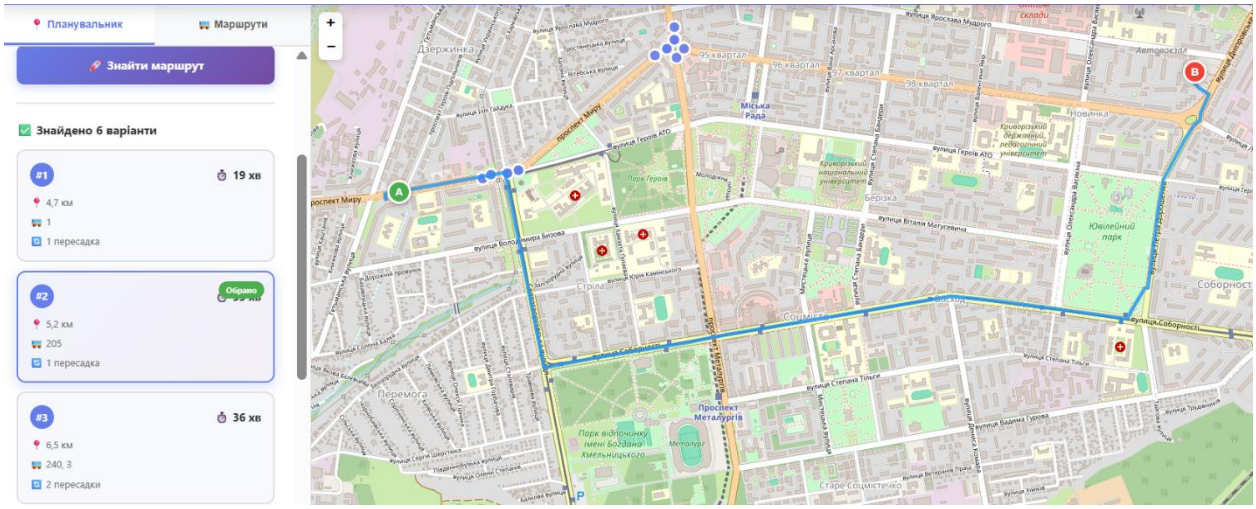

Рисунок 3.8 – Тестування побудови маршруту з використанням планувальника

Перевірялася робота автодоповнення адрес при введенні користувачем початкової та кінцевої точок маршруту. Тестувалося чи коректно система відображає релевантні варіанти адрес в процесі введення та чи правильно обирається адреса при виборі варіанту зі списку. Особлива увага приділялася тестуванню обробки адрес різних форматів, включаючи повні поштові адреси, назви вулиць без номерів будинків та назви визначних об'єктів.

Тестування побудови маршрутів включало перевірку різних сценаріїв пересування, від простих маршрутів з одним видом транспорту до складних маршрутів з множиною пересадок. Перевірялося чи коректно система відображає всі сегменти маршруту на карті та чи надає детальні покрокові інструкції для кожного етапу подорожі з інформацією про відстань та час.

Окремо тестувалася обробка помилкових ситуацій, таких як неможливість побудови маршруту між вказаними точками через відсутність транспортного сполучення або введення некоректних адрес. Перевірялося чи система надає зрозумілі повідомлення про помилки та чи дозволяє користувачу легко виправити введені дані для повторної спроби побудови маршруту.

Висновки до розділу:

Завершення комплексного циклу тестування підтвердило готовність розробленої web-платформи до практичного використання. Всі критичні та високопріоритетні проблеми, виявлені в процесі тестування, були успішно усунуті, а система продемонструвала стабільну роботу під різними рівнями навантаження та в різних умовах експлуатації.

Таким чином, проведене комплексне тестування підтвердило функціональну коректність, продуктивність, надійність та зручність використання розробленої web-платформи для моніторингу міського пасажирського транспорту, що дозволяє рекомендувати систему для практичного впровадження та використання реальними користувачами.

ВИСНОВКИ

У даній кваліфікаційній роботі було створено інформаційну систему контролю за переміщенням міського пасажирського транспорту, яка забезпечує моніторинг руху транспортних засобів у режимі реальної години та дає змогу планувати оптимальні маршрути пересування. У ході дослідження проведено всебічний аналіз предметної області, що дозволило обґрунтувати необхідність впровадження подібної системи для підвищення ефективності функціонування міської транспортної інфраструктури. Визначено ключові вимоги до функціональності системи, а також окреслено технічні обмеження, що впливають на її проектування та реалізацію.

Розроблена архітектура системи базується на трирівневій клієнт-серверній моделі з використанням сучасних веб-технологій: React.js для клієнтської частини, Node.js та Express.js для серверної логіки, а також СУБД MySQL для зберігання даних. Спроектовано структуру бази даних, що включає п'ять основних таблиць (routes, stops, route_stops, vehicles, gps_data) та містить складні міжтабличні зв'язки та оптимізовану систему індексування, що забезпечує високу продуктивність виконання запитів.

Для доступу до даних про маршрути, зупинки та транспортні засоби реалізовано RESTful API з підтримкою асинхронних запитів та ефективним управлінням під'єднаннями до бази даних через пул з'єднань. Обновлення інформації в режимі реальної години забезпечене за допомогою протоколу WebSocket та бібліотеки Socket.IO, що дозволяє передавати дані про актуальні позиції транспортних засобів із регулярністю у п'ять секунд без необхідності повторного завантаження сторінки.

Окрему увагу приділено створенню інтерактивного користувацького інтерфейсу. Для відображення карти міста, маршрутів, остановок та транспортних засобів застосовано бібліотеку Leaflet, що забезпечує корректну роботу інтерфейсу на різних типах пристроїв завдяки адаптивному дизайну. Додатково інтегровано Google Maps API для реалізації функції побудови

оптимальних маршрутів пересування громадським транспортом між довільно заданими точками з урахуванням мінімальної кількості пересадок.

Система пройшла комплексне тестування, яке включало функціональну перевірку, тестування продуктивності під навантаженням до ста одночасних користувачів, кросбраузерну перевірку, а також юзабіліті-тестування за участю реальних користувачів. Практичну апробацію проведено на даних транспортної мережі міста Кривий Ріг, що охоплює 115 маршрутів та 2148 зупинок. Отримані результати засвідчили стабільну роботу системи та середню годину відкликання на запити в межах 150–250 мілісекунд.

Практична цінність розробленої системи полягає у створенні повнофункціональної веб-платформи, придатної для використання мешканцями міста та диспетчерськими службами транспортних підприємств. Вона забезпечує доступ до актуальної інформації про рух громадського транспорту, що сприяє ефективнішому плануванню поїздок та скороченню часу очікування.

Архітектура системи передбачає можливості масштабування та подальшого розширення функціональності. Потенційними напрямками розвитку є інтеграція з реальними GPS-трекерами транспортних засобів, впровадження механізмів прогнозування часу прибуття на основі історичних даних, створення системи персональних сповіщень для користувачів, а також розроблення мобільних додатків для платформ iOS та Android.

Проведена робота підтверджує успішне виконання поставлених завдань дослідження. Розроблена інформаційна система продемонструвала функціональну коректність, високу продуктивність та практичну придатність для вирішення проблеми моніторингу міського пасажирського транспорту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Тімонін О. Інформаційні системи на транспорті: підручник / О. Тімонін, І. Бабій. – Київ: ДЕТУТ, 2018. – 342 с.

2. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: Doctoral dissertation / R. T. Fielding. – University of California, Irvine, 2000. – 162 p.

3. Banks A. MQTT Version 3.1.1 / A. Banks, R. Gupta // OASIS Standard. – 2014. – 81 p.

4. Коннолли Т. Базы данных: проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли, К. Бегг; пер. с англ. – 3-е изд. – Москва: Вильямс, 2003. – 1440 с.

5. Tilkov S. REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web / S. Tilkov, M. Eigenbrodt, S. Schreier. – dpunkt.verlag, 2015. – 330 p.

6. Дейт К. Дж. Введение в системы баз данных / К. Дж. Дейт; пер. с англ. – 8-е изд. – Москва: Вильямс, 2005. – 1328 с.

7. Fowler M. Patterns of Enterprise Application Architecture / M. Fowler. – Boston: Addison-Wesley, 2002. – 560 p.

8. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. – Addison-Wesley Professional, 1994. – 416 p.

9. React Documentation. A JavaScript library for building user interfaces [Електронний ресурс]. – Режим доступу: https://react.dev/. – Назва з екрана.

10. Node.js Documentation. Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine [Електронний ресурс]. – Режим доступу: https://nodejs.org/docs/. – Назва з екрана.

11. MySQL 8.0 Reference Manual [Електронний ресурс]. – Oracle Corporation, 2024. – Режим доступу: https://dev.mysql.com/doc/refman/8.0/en/. – Назва з екрана.

12. Aghaei S. Evolution of the World Wide Web: From Web 1.0 to Web 4.0 / S. Aghaei, M. A. Nematbakhsh, H. K. Farsani // International Journal of Web & Semantic Technology. – 2012. – Vol. 3, No. 1. – P. 1-10.

13. Leaflet Documentation. An open-source JavaScript library for mobile-friendly interactive maps [Електронний ресурс]. – Режим доступу: https://leafletjs.com/reference.html. – Назва з екрана.

14. Socket.IO Documentation. Bidirectional and low-latency communication for every platform [Електронний ресурс]. – Режим доступу: https://socket.io/docs/v4/. – Назва з екрана.

15. Express.js Documentation. Fast, unopinionated, minimalist web framework for Node.js [Електронний ресурс]. – Режим доступу: https://expressjs.com/. – Назва з екрана.

16. Grigorik I. High Performance Browser Networking / I. Grigorik. – O'Reilly Media, 2013. – 400 p.

17. Google Maps Platform Documentation [Електронний ресурс]. – Google LLC, 2024. – Режим доступу: https://developers.google.com/maps/documentation. – Назва з екрана.

18. Sommerville I. Software Engineering / I. Sommerville. – 10th ed. – Pearson, 2015. – 816 p.

19. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language / M. Fowler. – 3rd ed. – Addison-Wesley Professional, 2003. – 208 p.

20. Коннов Р. А. Системы мониторинга пассажирского транспорта: принципы построения и функционирования / Р. А. Коннов // Вестник транспорта. – 2017. – № 5. – С. 22-28.

21. Newman S. Building Microservices: Designing Fine-Grained Systems / S. Newman. – 2nd ed. – O'Reilly Media, 2021. – 612 p.

22. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems / M. Kleppmann. – O'Reilly Media, 2017. – 616 p.

23. Richter J. CLR via C#: Common Language Runtime / J. Richter. – 4th ed. – Microsoft Press, 2012. – 896 p.

24. ДСТУ ISO/IEC 12207:2016. Інформаційні технології. Системна та програмна інженерія. Процеси життєвого циклу програмних засобів. – Київ: ДП «УкрНДНЦ», 2017. – 96 с.

25. Маринич І. А., Тронь В. В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 151 “Автоматизація та комп’ютерно-інтегровані технології”. Кривий Ріг : Видавничий центр КНУ, 2022. 50с.

26. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

27. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

28. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові.

29. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація)

30. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с. (Інформація та документація).

ДОДАТОК А

Лістинг сторінки index.js

```
const express = require('express');
const mysql = require('mysql2/promise');
const cors = require('cors');
const http = require('http');
const socketIo = require('socket.io');
require('dotenv').config();

const app = express();
const server = http.createServer(app);
const io = socketIo(server, {
  cors: {
    origin: process.env.CLIENT_URL || 'http://localhost:5173',
    methods: ['GET', 'POST']
  }
});

// Middleware
app.use(cors());
app.use(express.json());

// ЛОГУВАННЯ
app.use((req, res, next) => {
  console.log(`📥 ${req.method} ${req.path}`);
  next();
});

// ПЕРЕВІРКА ЩО ENDPOINTS ЗАРЕЄСТРОВАНІ
console.log('📋 Registered routes:');
app._router.stack.forEach((r) => {
  if (r.route && r.route.path) {
    console.log(`  ${Object.keys(r.route.methods)[0].toUpperCase()}
${r.route.path}`);
  }
});
// MySQL connection pool
const pool = mysql.createPool({
  host: process.env.DB_HOST || 'localhost',
  user: process.env.DB_USER || 'root',
  password: process.env.DB_PASSWORD || '',
  database: process.env.DB_NAME || 'kryvyi_rih_transport',
  waitForConnections: true,
  connectionLimit: 10,
  queueLimit: 0,
  charset: 'utf8mb4'
});
```

```
// Test database connection
pool.getConnection()
  .then(connection => {
    console.log('☑ Database connected successfully');
    connection.release();
  })
  .catch(err => {
    console.error('❌ Database connection failed:', err.message);
  });

// ======================================================
// API Endpoints
// ======================================================

// Health check
app.get('/api/health', (req, res) => {
  res.json({ status: 'ok', timestamp: new Date().toISOString() });
});

// Отримати всі маршрути
app.get('/api/routes', async (req, res) => {
  try {
    const [routes] = await pool.query(
      'SELECT * FROM routes WHERE is_active = TRUE ORDER BY transport_type,
CAST(route_number AS UNSIGNED), route_number'
    );
    res.json({ success: true, data: routes });
  } catch (error) {
    console.error('Error fetching routes:', error);
    res.status(500).json({ success: false, error: 'Failed to fetch routes' });
  }
});

// Отримати конкретний маршрут
app.get('/api/routes/:routeId', async (req, res) => {
  try {
    const { routeId } = req.params;
    const [routes] = await pool.query(
      'SELECT * FROM routes WHERE id = ? AND is_active = TRUE',
      [routeId]
    );

    if (routes.length === 0) {
      return res.status(404).json({ success: false, error: 'Route not found' });
    }

    res.json({ success: true, data: routes[0] });
  } catch (error) {
    console.error('Error fetching route:', error);
    res.status(500).json({ success: false, error: 'Failed to fetch route' });
  }
```

```
});

// Отримати зупинки конкретного маршруту
app.get('/api/routes/:routeId/stops', async (req, res) => {
  try {
    const { routeId } = req.params;

    const [stops] = await pool.query(`
      SELECT
        s.id,
        s.stop_name,
        s.latitude,
        s.longitude,
        rs.stop_sequence,
        rs.direction
      FROM route_stops rs
      JOIN stops s ON rs.stop_id = s.id
      WHERE rs.route_id = ?
      ORDER BY rs.direction, rs.stop_sequence
    `, [routeId]);

    res.json({ success: true, data: stops });
  } catch (error) {
    console.error('Error fetching route stops:', error);
    res.status(500).json({ success: false, error: 'Failed to fetch route stops' });
  }
});

// Отримати транспортні засоби на маршруті
app.get('/api/routes/:routeId/vehicles', async (req, res) => {
  try {
    const { routeId } = req.params;

    const [vehicles] = await pool.query(`
      SELECT
        v.id,
        v.vehicle_number,
        v.transport_type,
        v.is_active,
        g.latitude,
        g.longitude,
        g.speed,
        g.heading,
        g.timestamp
      FROM vehicles v
      LEFT JOIN gps_data g ON v.id = g.vehicle_id
      WHERE v.route_id = ? AND v.is_active = TRUE
      AND g.id = (
        SELECT id FROM gps_data
        WHERE vehicle_id = v.id
        ORDER BY timestamp DESC
```

```
        LIMIT 1
      )
    `, [routeId]);

    res.json({ success: true, data: vehicles });
  } catch (error) {
    console.error('Error fetching vehicles:', error);
    res.status(500).json({ success: false, error: 'Failed to fetch vehicles' });
  }
});

// Отримати деталі маршруту (інфо + зупинки + транспорт)
app.get('/api/routes/:routeId/details', async (req, res) => {
  try {
    const { routeId } = req.params;

    // Інформація про маршрут
    const [routes] = await pool.query(
      'SELECT * FROM routes WHERE id = ?',
      [routeId]
    );

    if (routes.length === 0) {
      return res.status(404).json({ success: false, error: 'Route not found' });
    }

    const route = routes[0];

    // Зупинки
    const [stops] = await pool.query(`
      SELECT
        s.id,
        s.stop_name,
        s.latitude,
        s.longitude,
        rs.stop_sequence,
        rs.direction
      FROM route_stops rs
      JOIN stops s ON rs.stop_id = s.id
      WHERE rs.route_id = ?
      ORDER BY rs.direction, rs.stop_sequence
    `, [routeId]);

    // Транспорт
    const [vehicles] = await pool.query(`
      SELECT
        v.id,
        v.vehicle_number,
        v.transport_type,
        v.is_active,
        g.latitude,
```

```
        g.longitude,
        g.speed,
        g.heading,
        g.timestamp
      FROM vehicles v
      LEFT JOIN gps_data g ON v.id = g.vehicle_id
      WHERE v.route_id = ? AND v.is_active = TRUE
      AND g.id = (
        SELECT id FROM gps_data
        WHERE vehicle_id = v.id
        ORDER BY timestamp DESC
        LIMIT 1
      )
    `, [routeId]);

    res.json({
      success: true,
      data: {
        route,
        stops,
        vehicles
      }
    });
  } catch (error) {
    console.error('Error fetching route details:', error);
    res.status(500).json({ success: false, error: 'Failed to fetch route details'
});
  }
});

// Отримати всі зупинки
app.get('/api/stops', async (req, res) => {
  try {
    const [stops] = await pool.query('SELECT * FROM stops ORDER BY stop_name');
    res.json({ success: true, data: stops });
  } catch (error) {
    console.error('Error fetching stops:', error);
    res.status(500).json({ success: false, error: 'Failed to fetch stops' });
  }
});


// ВИВЕДЕННЯ ЗАРЕЄСТРОВАНИХ ROUTES
console.log('\n📋 Registered API routes:');
app._router.stack.forEach((r) => {
  if (r.route && r.route.path) {
    console.log(`  ${Object.keys(r.route.methods)[0].toUpperCase()}
${r.route.path}`);
  }
});
```

```
// ====================================================
// WebSocket для real-time оновлень
// ====================================================

io.on('connection', (socket) => {
  console.log('🔌 Client connected:', socket.id);

  socket.on('subscribe:route', (routeId) => {
    console.log(`📍 Client subscribed to route ${routeId}`);
    socket.join(`route:${routeId}`);
  });

  socket.on('unsubscribe:route', (routeId) => {
    console.log(`📍 Client unsubscribed from route ${routeId}`);
    socket.leave(`route:${routeId}`);
  });

  socket.on('disconnect', () => {
    console.log('🔌 Client disconnected:', socket.id);
  });
});

// Емуляція оновлень GPS (кожні 5 секунд)
// В реальному проекті це буде отримувати дані з GPS трекерів
setInterval(async () => {
  try {
    const [activeRoutes] = await pool.query(
      'SELECT DISTINCT route_id FROM vehicles WHERE is_active = TRUE'
    );

    for (const { route_id } of activeRoutes) {
      const [vehicles] = await pool.query(`
        SELECT
          v.id,
          v.vehicle_number,
          v.transport_type,
          g.latitude,
          g.longitude,
          g.speed,
          g.heading
        FROM vehicles v
        LEFT JOIN gps_data g ON v.id = g.vehicle_id
        WHERE v.route_id = ? AND v.is_active = TRUE
        AND g.id = (
          SELECT id FROM gps_data
          WHERE vehicle_id = v.id
          ORDER BY timestamp DESC
          LIMIT 1
        )
      `, [route_id]);
```

```
      if (vehicles.length > 0) {
        io.to(`route:${route_id}`).emit('vehicles:update', vehicles);
      }
    }
  } catch (error) {
    console.error('Error in GPS update interval:', error);
  }
}, 5000);

// ========================================================
// Server Start
// ========================================================

const PORT = process.env.PORT || 3000;

server.listen(PORT, () => {
  console.log('🚀 Server running on port', PORT);
  console.log('📍 API available at http://localhost:' + PORT + '/api');
  console.log('🔌 WebSocket available at http://localhost:' + PORT);
});

module.exports = { app, server, io };
```

Лістинг сторінки server.js

```
// ========================================================
// Backend Server - Громадський транспорт Кривого Рогу
// ========================================================

const express = require('express');
const mysql = require('mysql2/promise');
const cors = require('cors');
const http = require('http');
const { Server } = require('socket.io');
const path = require('path');
require('dotenv').config();

const app = express();
const server = http.createServer(app);
const io = new Server(server, {
    cors: { origin: "*", methods: ["GET", "POST"] }
});

// Middleware
app.use(cors());
app.use(express.json());
app.use(express.static(path.join(__dirname, 'client', 'dist')));

// MySQL підключення
const pool = mysql.createPool({
```

```
    host: process.env.DB_HOST || 'localhost',
    user: process.env.DB_USER || 'root',
    password: process.env.DB_PASSWORD || '',
    database: process.env.DB_NAME || 'kryvyi_rih_transport',
    waitForConnections: true,
    connectionLimit: 10,
    queueLimit: 0
});

// Перевірка підключення
pool.getConnection()
    .then(connection => {
        console.log('☑ Підключено до MySQL');
        connection.release();
    })
    .catch(err => {
        console.error('❌ Помилка підключення до БД:', err.message);
    });

// ========================================================
// API ENDPOINTS
// ========================================================

// 1. Отримати список маршрутів
app.get('/api/routes', async (req, res) => {
    try {
        const { type } = req.query;
        let query = 'SELECT * FROM routes WHERE is_active = TRUE';
        let params = [];

        if (type && type !== 'all') {
            query += ' AND transport_type = ?';
            params.push(type);
        }

        query += ' ORDER BY route_number';
        const [routes] = await pool.query(query, params);

        res.json({ success: true, data: routes });
    } catch (error) {
        console.error('Error getting routes:', error);
        res.status(500).json({ success: false, error: error.message });
    }
});

// 2. Отримати деталі маршруту зі зупинками
app.get('/api/routes/:id', async (req, res) => {
    try {
        const [routes] = await pool.query('SELECT * FROM routes WHERE id = ?',
[req.params.id]);
```

```
        if (routes.length === 0) {
            return res.status(404).json({
                success: false,
                error: 'Маршрут не знайдено'
            });
        }

        const [stops] = await pool.query(`
            SELECT
                s.id,
                s.stop_name,
                s.latitude,
                s.longitude,
                rs.stop_sequence,
                rs.direction
            FROM route_stops rs
            INNER JOIN stops s ON rs.stop_id = s.id
            WHERE rs.route_id = ?
            ORDER BY rs.direction, rs.stop_sequence
        `, [req.params.id]);

        res.json({
            success: true,
            data: {
                route: routes[0],
                stops
            }
        });
    } catch (error) {
        console.error('Error getting route details:', error);
        res.status(500).json({ success: false, error: error.message });
    }
});

// 3. Отримати поточні позиції всіх транспортних засобів
app.get('/api/vehicles/positions', async (req, res) => {
    try {
        const [positions] = await pool.query('CALL GetCurrentVehiclePositions()');
        res.json({ success: true, data: positions[0] || [] });
    } catch (error) {
        console.error('Error getting positions:', error);
        res.json({ success: true, data: [] });
    }
});

// 4. Отримати всі зупинки
app.get('/api/stops', async (req, res) => {
    try {
        const [stops] = await pool.query('SELECT * FROM stops ORDER BY stop_name');
        res.json({ success: true, data: stops });
    } catch (error) {
```

```
        console.error('Error getting stops:', error);
        res.status(500).json({ success: false, error: error.message });
    }
});

// 5. Оновити GPS дані (для GPS-пристроїв)
app.post('/api/gps/update', async (req, res) => {
    try {
        const { gps_device_id, latitude, longitude, speed, heading, accuracy } =
req.body;

        if (!gps_device_id || !latitude || !longitude) {
            return res.status(400).json({
                success: false,
                error: 'Необхідні параметри: gps_device_id, latitude, longitude'
            });
        }

        const [vehicles] = await pool.query(
            'SELECT id FROM vehicles WHERE gps_device_id = ?',
            [gps_device_id]
        );

        if (vehicles.length === 0) {
            return res.status(404).json({
                success: false,
                error: 'Транспортний засіб не знайдено'
            });
        }

        const vehicle_id = vehicles[0].id;

        await pool.query(`
            INSERT INTO gps_data (vehicle_id, latitude, longitude, speed, heading,
accuracy)
            VALUES (?, ?, ?, ?, ?, ?)
        `, [vehicle_id, latitude, longitude, speed || 0, heading || null, accuracy
|| null]);

        io.emit('gps_update', {
            vehicle_id,
            gps_device_id,
            latitude,
            longitude,
            speed: speed || 0,
            heading: heading || null,
            timestamp: new Date()
        });

        res.json({ success: true, message: 'GPS дані оновлено' });
    } catch (error) {
```

```
        console.error('Error updating GPS:', error);
        res.status(500).json({ success: false, error: error.message });
    }
});

// 6. Отримати транспорт конкретного маршруту
app.get('/api/routes/:id/vehicles', async (req, res) => {
    try {
        const [vehicles] = await pool.query(`
            SELECT
                v.id,
                v.vehicle_number,
                v.transport_type,
                r.route_number,
                r.route_name,
                r.color,
                g.latitude,
                g.longitude,
                g.speed,
                g.heading,
                g.timestamp
            FROM vehicles v
            INNER JOIN routes r ON v.route_id = r.id
            LEFT JOIN (
                SELECT vehicle_id, latitude, longitude, speed, heading, timestamp
                FROM gps_data
                WHERE timestamp > DATE_SUB(NOW(), INTERVAL 5 MINUTE)
                GROUP BY vehicle_id
                ORDER BY timestamp DESC
            ) g ON v.id = g.vehicle_id
            WHERE v.route_id = ? AND v.is_active = TRUE
        `, [req.params.id]);

        res.json({ success: true, data: vehicles });
    } catch (error) {
        console.error('Error getting route vehicles:', error);
        res.status(500).json({ success: false, error: error.message });
    }
});

// ======================================================
// ROUTES - SPA підтримка
// ======================================================

app.get('*', (req, res) => {
    res.sendFile(path.join(__dirname, 'client', 'dist', 'index.html'));
});

// ======================================================
// WEBSOCKET
// ======================================================
```

```
io.on('connection', (socket) => {
    console.log('🔌 Клієнт підключено:', socket.id);

    // Відправити початкові позиції
    pool.query('CALL GetCurrentVehiclePositions()')
        .then(([positions]) => {
            socket.emit('initial_positions', positions[0] || []);
        })
        .catch(err => console.error('Error sending initial positions:', err));

    socket.on('disconnect', () => {
        console.log('❌ Клієнт від\'єднано:', socket.id);
    });
});

// Автоматична трансляція позицій кожні 5 секунд
setInterval(async () => {
    try {
        const [positions] = await pool.query('CALL GetCurrentVehiclePositions()');
        io.emit('positions_update', positions[0] || []);
    } catch (error) {
        console.error('Error broadcasting positions:', error.message);
    }
}, 5000);
// ======================================================
// GPS СИМУЛЯЦІЯ (тільки для розробки)
// ======================================================

async function simulateGPSData() {
    if (process.env.NODE_ENV !== 'development') return;

    try {
        const [vehicles] = await pool.query(`
            SELECT v.id, v.route_id
            FROM vehicles v
            WHERE v.is_active = TRUE
            ORDER BY RAND()
            LIMIT 1
        `);

        if (vehicles.length > 0) {
            const vehicle = vehicles[0];

            const [stops] = await pool.query(`
                SELECT s.latitude, s.longitude
                FROM route_stops rs
                INNER JOIN stops s ON rs.stop_id = s.id
                WHERE rs.route_id = ?
                ORDER BY RAND()
                LIMIT 1
```

```
            `, [vehicle.route_id]);

            if (stops.length > 0) {
                const stop = stops[0];
                const lat = parseFloat(stop.latitude) + (Math.random() - 0.5) *
0.001;
                const lon = parseFloat(stop.longitude) + (Math.random() - 0.5) *
0.001;
                const speed = Math.random() * 40 + 20;
                const heading = Math.floor(Math.random() * 360);

                await pool.query(`
                    INSERT INTO gps_data (vehicle_id, latitude, longitude, speed,
heading)
                    VALUES (?, ?, ?, ?, ?)
                `, [vehicle.id, lat, lon, speed, heading]);

                console.log(` 📍 GPS симуляція: ТЗ #${vehicle.id}`);
            }
        }
    } catch (error) {
        console.error('GPS симуляція помилка:', error.message);
    }
}

// Запуск симуляції кожні 10 секунд (тільки для розробки)
if (process.env.NODE_ENV === 'development') {
    setInterval(simulateGPSData, 10000);
}

// ========================================================
// ЗАПУСК СЕРВЕРА
// ========================================================
const PORT = process.env.PORT || 3000;

server.listen(PORT, () => {
    console.log('');
    console.log('========================================');
    console.log(`🚀 Сервер запущено на порті ${PORT}`);
    console.log(` 📍 http://localhost:${PORT}`);
    console.log('========================================');
    console.log('');
});
// Graceful shutdown
process.on('SIGTERM', () => {
    console.log('SIGTERM отримано, зупинка сервера...');
    server.close(() => {
        console.log('Сервер зупинено');
        pool.end();
    });
});
```

ДОДАТОК Б

Лістинг сторінки populate-routes.js

```
// ======================================================
// Скрипт заповнення БД реальними маршрутами Кривого Рогу
// Дані зібрані з публічних джерел: eWay, OpenStreetMap
// ======================================================

const mysql = require('mysql2/promise');
require('dotenv').config();

// Реальні маршрути громадського транспорту Кривого Рогу
const ROUTES_DATA = {
  // ===== ТРАМВАЇ =====
  trams: [
    { number: '1', name: 'пл. Визволення - Тернівка', color: '#FF6B6B' },
    { number: '2', name: 'Вокзал - Тернівка', color: '#4ECDC4' },
    { number: '3', name: 'пл. Визволення - Соцмісто', color: '#45B7D1' },
    { number: '4', name: 'Вокзал - Соцмісто', color: '#FFA07A' },
    { number: '5', name: 'Вокзал - мкр. Дніпровський', color: '#98D8C8' },
    { number: '6', name: 'пл. Визволення - мкр. Дніпровський', color: '#F7DC6F' },
    { number: '11', name: 'пл. Визволення - мкр. Соцмісто (кільцевий)', color:
'#BB8FCE' }
  ],

  // ===== ШВИДКІСНИЙ ТРАМВАЙ =====
  speedTrams: [
    { number: '1Ш', name: 'пл. Визволення - Інгулецький', color: '#E74C3C' },
    { number: '2Ш', name: 'Вокзал - Інгулецький', color: '#3498DB' }
  ],

  // ===== ТРОЛЕЙБУСИ =====
  trolleybuses: [
    { number: '1', name: 'Вокзал - мкр. Соцмісто', color: '#95E1D3' },
    { number: '2', name: 'Вокзал - вул. Кірова', color: '#F38181' },
    { number: '3', name: 'Довгинцеве - Соцмісто', color: '#AA96DA' },
    { number: '4', name: 'Вокзал - Тернівка', color: '#FCBAD3' },
    { number: '5', name: 'пл. Визволення - мкр. Сонячний', color: '#FFFFD2' },
    { number: '6', name: 'пл. Визволення - Соцмісто', color: '#A8D8EA' },
    { number: '7', name: 'Вокзал - Тернівка (через Соцмісто)', color: '#FFBF86' },
    { number: '8', name: 'пл. Визволення - Тернівка', color: '#FF8C94' },
    { number: '9', name: 'Вокзал - Тернівка (через центр)', color: '#C4FAF8' }
  ],

  // ===== АВТОБУСИ (основні маршрути) =====
  buses: [
    { number: '11', name: 'Автовокзал - мкр. Соцмісто', color: '#FFB6C1' },
    { number: '12', name: 'Вокзал - Тернівка', color: '#DDA0DD' },
    { number: '41', name: 'Автовокзал - Жовтневий район', color: '#87CEEB' },
```

```
    { number: '42', name: 'пл. Визволення - Червоний Камінь', color: '#90EE90' },
    { number: '43', name: 'Вокзал - мкр. Сонячний', color: '#FFE4B5' },
    { number: '44', name: 'пл. Визволення - Інгулець', color: '#F0E68C' },
    { number: '45', name: 'Автовокзал - Довгинцеве', color: '#D8BFD8' },
    { number: '52', name: 'Центр - Інгулецький район', color: '#FFA07A' },
    { number: '55', name: 'Автовокзал - Червоний Камінь', color: '#20B2AA' },
    { number: '61', name: 'пл. Визволення - Жовтневий район', color: '#87CEFA' },
    { number: '62', name: 'Вокзал - Інгулець', color: '#778899' },
    { number: '101', name: 'Автовокзал - Тернівка (експрес)', color: '#B0C4DE' },
    { number: '102', name: 'Автовокзал - Соцмісто (експрес)', color: '#FFDAB9' },
    { number: '201', name: 'Центральний ринок - мкр. Сонячний', color: '#EEE8AA' },
    { number: '222', name: 'Вокзал - Червоний Камінь', color: '#98FB98' },
    { number: '224', name: 'пл. Визволення - Інгулецький', color: '#AFEEEE' }
  ]
};

// Основні зупинки Кривого Рогу з реальними координатами
const MAIN_STOPS = [
  { name: 'пл. Визволення', lat: 47.9104, lng: 33.3909 },
  { name: 'Вокзал', lat: 47.9084, lng: 33.3847 },
  { name: 'Автовокзал', lat: 47.9110, lng: 33.3920 },
  { name: 'Соцмісто', lat: 47.9205, lng: 33.4110 },
  { name: 'Тернівка', lat: 47.8980, lng: 33.3750 },
  { name: 'Інгулецький район', lat: 47.9300, lng: 33.3600 },
  { name: 'Червоний Камінь', lat: 47.8850, lng: 33.3500 },
  { name: 'Жовтневий район', lat: 47.9400, lng: 33.4200 },
  { name: 'мкр. Сонячний', lat: 47.9150, lng: 33.4300 },
  { name: 'Довгинцеве', lat: 47.9350, lng: 33.3400 },
  { name: 'Центральний ринок', lat: 47.9120, lng: 33.3880 },
  { name: 'мкр. Дніпровський', lat: 47.9250, lng: 33.4000 },
  { name: 'вул. Кірова', lat: 47.9050, lng: 33.3950 },
  { name: 'Інгулець', lat: 47.9320, lng: 33.3580 }
];

async function populateDatabase() {
  let connection;

  try {
    console.log('🔌 Підключення до БД...');

    connection = await mysql.createConnection({
      host: process.env.DB_HOST || 'localhost',
      user: process.env.DB_USER || 'root',
      password: process.env.DB_PASSWORD || '',
      database: process.env.DB_NAME || 'kryvyi_rih_transport',
      charset: 'utf8mb4'
    });

    console.log('☑ Підключено до БД');

    // Очистити існуючі дані
```

```
    console.log('\n🗑️  Очищення старих даних...');
    await connection.query('DELETE FROM route_stops');
    await connection.query('DELETE FROM vehicles');
    await connection.query('DELETE FROM gps_data');
    await connection.query('DELETE FROM routes');
    await connection.query('DELETE FROM stops');

    // Скинути AUTO_INCREMENT
    await connection.query('ALTER TABLE routes AUTO_INCREMENT = 1');
    await connection.query('ALTER TABLE stops AUTO_INCREMENT = 1');

    console.log('☑️ Стара база очищена');

    // Додати зупинки
    console.log('\n📍 Додавання зупинок...');
    for (const stop of MAIN_STOPS) {
      await connection.query(
        'INSERT INTO stops (stop_name, latitude, longitude) VALUES (?, ?, ?)',
        [stop.name, stop.lat, stop.lng]
      );
    }
    console.log(`☑️ Додано ${MAIN_STOPS.length} зупинок`);

    // Додати маршрути
    console.log('\n🚊 Додавання маршрутів...');

    let totalRoutes = 0;

    // Трамваї
    for (const route of ROUTES_DATA.trams) {
      await connection.query(
        'INSERT INTO routes (route_number, route_name, transport_type, color)
VALUES (?, ?, ?, ?)',
        [route.number, route.name, 'tram', route.color]
      );
      totalRoutes++;
    }
    console.log(`  ✓ Додано ${ROUTES_DATA.trams.length} трамвайних маршрутів`);

    // Швидкісні трамваї
    for (const route of ROUTES_DATA.speedTrams) {
      await connection.query(
        'INSERT INTO routes (route_number, route_name, transport_type, color)
VALUES (?, ?, ?, ?)',
        [route.number, route.name, 'speed_tram', route.color]
      );
      totalRoutes++;
    }
    console.log(`  ✓ Додано ${ROUTES_DATA.speedTrams.length} маршрутів швидкісного
трамваю`);
```

```
    // Тролейбуси
    for (const route of ROUTES_DATA.trolleybuses) {
      await connection.query(
        'INSERT INTO routes (route_number, route_name, transport_type, color)
VALUES (?, ?, ?, ?)',
        [route.number, route.name, 'trolleybus', route.color]
      );
      totalRoutes++;
    }
    console.log(`  ✓ Додано ${ROUTES_DATA.trolleybuses.length} тролейбусних
маршрутів`);

    // Автобуси
    for (const route of ROUTES_DATA.buses) {
      await connection.query(
        'INSERT INTO routes (route_number, route_name, transport_type, color)
VALUES (?, ?, ?, ?)',
        [route.number, route.name, 'bus', route.color]
      );
      totalRoutes++;
    }
    console.log(`  ✓ Додано ${ROUTES_DATA.buses.length} автобусних маршрутів`);

    console.log(`\n☑ ВСЬОГО додано ${totalRoutes} маршрутів`);

    // Статистика
    console.log('\n📊 Статистика бази даних:');
    const [routesCount] = await connection.query('SELECT COUNT(*) as count FROM
routes');
    const [stopsCount] = await connection.query('SELECT COUNT(*) as count FROM
stops');
    const [tramCount] = await connection.query("SELECT COUNT(*) as count FROM
routes WHERE transport_type = 'tram'");
    const [speedTramCount] = await connection.query("SELECT COUNT(*) as count FROM
routes WHERE transport_type = 'speed_tram'");
    const [trolleyCount] = await connection.query("SELECT COUNT(*) as count FROM
routes WHERE transport_type = 'trolleybus'");
    const [busCount] = await connection.query("SELECT COUNT(*) as count FROM routes
WHERE transport_type = 'bus'");

    console.log(`  🚊 Трамваї: ${tramCount[0].count}`);
    console.log(`  🚈 Швидкісний трамвай: ${speedTramCount[0].count}`);
    console.log(`  🚎 Тролейбуси: ${trolleyCount[0].count}`);
    console.log(`  🚌 Автобуси: ${busCount[0].count}`);
    console.log(`  📍 Зупинок: ${stopsCount[0].count}`);
    console.log(`  📊 ВСЬОГО маршрутів: ${routesCount[0].count}`);

    console.log('\n🎉 База даних успішно заповнена реальними маршрутами Кривого
Рогу!');

  } catch (error) {
```

```
    console.error('❌ Помилка:', error.message);
    throw error;
  } finally {
    if (connection) {
      await connection.end();
      console.log('\n👋 З\'єднання закрито');
    }
  }
}

// Запуск
if (require.main === module) {
  populateDatabase()
    .then(() => {
      console.log('\n☑️ Скрипт завершено успішно!');
      process.exit(0);
    })
    .catch(error => {
      console.error('\n❌ Критична помилка:', error);
      process.exit(1);
    });
}

module.exports = { populateDatabase, ROUTES_DATA, MAIN_STOPS };
```