НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

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

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення моніторингу завантаженості та аварійності на дорогах міст»
Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

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

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

Виконав _____ **Тимур МЕЛЬНИЧЕНКО**
(підпис)

КИЇВ – 2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

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

«___» _____ 2025 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Мельниченку Тимуру Руслановичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
«Програмне забезпечення моніторингу завантаженості та аварійності на дорогах міст»
затверджена наказом від «19» грудня 2025 р. № 3204 «С»

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

Вихідні дані до роботи: опис процесів моніторингу дорожньої ситуації та аналізу аварійності; вимоги до web-орієнтованої системи побудови маршрутів і оцінювання безпечності пересування; структура дорожніх подій, маршрутів та геопросторових даних; технології React, TypeScript, Firebase, Google Maps Platform, REST API та Cloud Firestore; наукові й інформаційні джерела з розробки систем моніторингу транспортної інфраструктури.

Перелік питань, що підлягають дослідженню: дослідження предметної області та вимог; проєктування інформаційної частини; проектування та розробка програмного забезпечення системи; експлуатація та впровадження системи

Перелік графічних документів (за необхідності).

Дата видачі завдання «_22_» ___грудня_____ 2025 р.

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

Завдання взяв до виконання _____ **Тимур МЕЛЬНИЧЕНКО**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface, інтерфейс програмування застосунків

Auth UI – модуль авторизації користувача

BaaS – Backend-as-a-Service, модель надання серверної інфраструктури як сервісу

CMS – Content Management System, система керування контентом

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

CSS – Cascading Style Sheets, каскадні таблиці стилів

DTO – Data Transfer Object, об’єкт передачі даних

ER-модель – Entity Relationship Model, модель «сутність–зв’язок»

Firebase – хмарна платформа для розробки та підтримки веб-застосунків

Firestore – Cloud Firestore, NoSQL база даних реального часу

Figma – веб-сервіс для проєктування інтерфейсів та веб-дизайну

Frontend – клієнтська частина програмного забезпечення

Backend – серверна частина програмного забезпечення

Git – система контролю версій програмного коду

GitHub – веб-платформа для спільної розробки та зберігання програмного коду

HTML – HyperText Markup Language, мова розмітки гіпертексту

HTTP – HyperText Transfer Protocol, протокол передачі гіпертексту

HTTPS – HyperText Transfer Protocol Secure, захищений протокол передачі даних

IDE – Integrated Development Environment, інтегроване середовище розробки

JSON – JavaScript Object Notation, формат обміну даними

JWT – JSON Web Token, токен авторизації користувача

Map UI – модуль картографічного інтерфейсу

MVC – Model-View-Controller, архітектурний шаблон програмного забезпечення

NoSQL – нереляційна база даних

ORM – Object-Relational Mapping, технологія об’єктно-реляційного відображення

REST – Representational State Transfer, архітектурний стиль взаємодії компонентів веб-систем

Route Analysis – модуль аналізу маршрутів

SQL – Structured Query Language, мова структурованих запитів

TypeScript – типізована мова програмування на основі JavaScript

UI – User Interface, користувацький інтерфейс

UML – Unified Modeling Language, уніфікована мова моделювання

URL – Uniform Resource Locator, уніфікований локатор ресурсу

UX – User Experience, користувацький досвід

Web API – програмний інтерфейс для взаємодії веб-компонентів

Zustand Store – модуль централізованого керування станом застосунку

Cloud Firestore – хмарна документно-орієнтована база даних Firebase

Firebase Authentication – сервіс авторизації та автентифікації користувачів Firebase.

ВСТУП

Стрімкий розвиток цифрових технологій та постійне зростання навантаження на транспортну інфраструктуру великих міст формують потребу у створенні сучасних інформаційних систем, здатних оперативно реагувати на зміни дорожньої ситуації. Збільшення кількості транспортних засобів, висока інтенсивність руху, часті дорожньо-транспортні пригоди, затори та тимчасові обмеження руху істотно впливають на ефективність функціонування міського транспорту, рівень безпеки учасників дорожнього руху та загальний комфорт пересування містом. У таких умовах особливої актуальності набувають web-орієнтовані системи моніторингу дорожнього середовища, які забезпечують збір, обробку та візуалізацію інформації у режимі реального часу.

Сучасні навігаційні сервіси переважно орієнтовані на визначення найкоротшого або найшвидшого маршруту, однак не завжди враховують комплексний вплив дорожніх подій, аварійності окремих ділянок, динамічної зміни транспортної ситуації та потенційних ризиків під час пересування. Використання хмарних платформ, картографічних API та realtime-механізмів синхронізації даних дозволяє реалізувати більш гнучкі інформаційні системи, які здатні не лише відображати поточну ситуацію на дорогах, але й забезпечувати аналіз маршрутів з урахуванням дорожніх інцидентів та транспортних обмежень.

Актуальність теми бакалаврської кваліфікаційної роботи полягає у необхідності розробки сучасного веб-застосунку для моніторингу дорожньої ситуації та побудови маршрутів з урахуванням аварій, заторів і транспортних подій у режимі реального часу. Практична цінність таких систем полягає у можливості підвищення безпеки пересування, оптимізації маршрутів та оперативного інформування користувачів про зміни дорожньої ситуації.

Метою роботи є розробка web-орієнтованої системи моніторингу дорожньої ситуації та побудови маршрутів з урахуванням дорожніх подій у міському середовищі.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз предметної області та існуючих систем моніторингу дорожнього руху;
- визначити функціональні та нефункціональні вимоги до програмної системи;
- спроєктувати архітектуру веб-застосунку та структуру збереження даних;
- реалізувати інтеграцію із сервісами Google Maps Platform та Firebase;
- розробити механізми аналізу дорожніх подій і побудови маршрутів;
- провести тестування та оцінити працездатність програмного забезпечення.

Об’єктом дослідження є процеси моніторингу дорожньої ситуації та навігації у міському транспортному середовищі.

Предметом дослідження є методи та програмні засоби розробки web-орієнтованих інформаційних систем моніторингу дорожнього руху з використанням картографічних сервісів і хмарних платформ.

У процесі виконання роботи використано методи аналізу інформаційних систем, web-проєктування, об’єктно-орієнтованого моделювання, клієнтської SPA-розробки, інтеграції API та роботи з хмарними сервісами. Для реалізації програмного забезпечення використано React, TypeScript, Firebase та Google Maps Platform .

Практичне значення одержаних результатів полягає у розробці веб-застосунку RoadGuard City, який забезпечує моніторинг дорожньої ситуації, відображення транспортних подій на інтерактивній карті, аналіз маршрутів та realtime-синхронізацію даних. Розроблена система може бути використана як основа для подальшого розвитку міських інформаційних сервісів та інтеграції з інтелектуальними транспортними системами.

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі виконано аналіз предметної області та сучасних систем моніторингу дорожнього руху. Другий розділ присвячений проєктуванню програмного забезпечення та моделюванню структури системи. У третьому розділі наведено опис програмної реалізації веб-застосунку та використаних технологій. Четвертий розділ містить результати тестування, опис процесу розгортання та демонстрацію роботи системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз предметної області систем моніторингу дорожньої ситуації

Розвиток сучасної транспортної інфраструктури безпосередньо пов’язаний зі зростанням рівня урбанізації, збільшенням кількості транспортних засобів та підвищенням інтенсивності дорожнього руху. Для великих міст характерним є постійне перевантаження окремих ділянок транспортної мережі, що призводить до утворення заторів, збільшення тривалості поїздок, підвищення рівня аварійності та зниження ефективності функціонування міської інфраструктури загалом. Особливо критичною така ситуація є у години пікового навантаження, коли навіть незначні дорожньо-транспортні пригоди або тимчасові перекриття здатні суттєво впливати на стабільність транспортних потоків.

У сучасних умовах важливу роль у вирішенні зазначених проблем відіграють інформаційні системи моніторингу дорожньої ситуації та навігаційні сервіси, що забезпечують користувачів актуальною інформацією про стан транспортної мережі. Подібні системи реалізують функції побудови маршрутів, оцінювання часу прибуття, аналізу транспортного навантаження та відображення дорожніх подій на картографічній основі. Найбільш поширеними прикладами таких платформ є Google Maps, Waze та HERE WeGo, які використовують геопросторові дані, алгоритми маршрутизації та механізми аналізу транспортних потоків для формування оптимальних маршрутів пересування [3–5].

Попри високий рівень розвитку сучасних навігаційних сервісів, більшість із них орієнтована переважно на мінімізацію часу поїздки або довжини маршруту. При цьому недостатньо враховуються фактори аварійності, небезпечності окремих ділянок дороги та вплив транспортних інцидентів на

безпечність пересування. У результаті користувач може отримати маршрут, який є найшвидшим лише на момент побудови, однак проходить через потенційно небезпечні ділянки.

Для систем моніторингу дорожньої ситуації важливою є підтримка роботи з даними у режимі реального часу. Це передбачає використання зовнішніх API та картографічних платформ для отримання інформації про транспортні потоки, дорожні події та маршрути пересування. За таких умов програмне забезпечення повинно забезпечувати швидку обробку даних, динамічне оновлення інформації та можливість масштабування при збільшенні кількості користувачів або дорожніх подій [6].

Предметна область розроблюваного програмного забезпечення охоплює процеси аналізу дорожньої ситуації, побудови маршрутів, оцінювання транспортних подій та визначення рівня безпечності пересування. Основною метою системи є інтеграція картографічних сервісів із механізмами аналізу дорожніх інцидентів для формування маршрутів, що враховують не лише швидкість руху, а й потенційний рівень ризику. Функціональні можливості розроблюваної системи наведено у табл. 1.1.

Таблиця 1.1

Основні функції системи моніторингу дорожньої ситуації

Функція системи	Характеристика
Побудова альтернативних маршрутів	Формування декількох варіантів пересування між заданими точками
Аналіз дорожніх подій	Врахування аварій, перекриттів та інших транспортних інцидентів
Динамічний розрахунок ETA	Обчислення орієнтовного часу прибуття з урахуванням поточної ситуації
Оцінювання безпечності маршруту	Визначення рівня ризику маршруту залежно від кількості та типу подій
Візуалізація дорожньої ситуації	Відображення маршрутів та подій на інтерактивній карті
Синхронізація у реальному часі	Оновлення інформації без перезавантаження сторінки
Розмежування доступу	Підтримка ролей користувача та адміністратора

Однією з ключових особливостей предметної області є необхідність роботи з геопросторовими даними. Програмне забезпечення повинно здійснювати аналіз координат дорожніх подій, визначати їхній вплив на окремі сегменти маршруту, обчислювати відстані між об’єктами та виконувати перевірку перетину маршрутів із потенційно небезпечними зонами. Для реалізації подібного функціоналу використовуються алгоритми геометричного аналізу, методи обробки координат та засоби роботи з картографічними сервісами [7].

Структуру взаємодії основних компонентів системи моніторингу дорожньої ситуації наведено на рисунку 1.

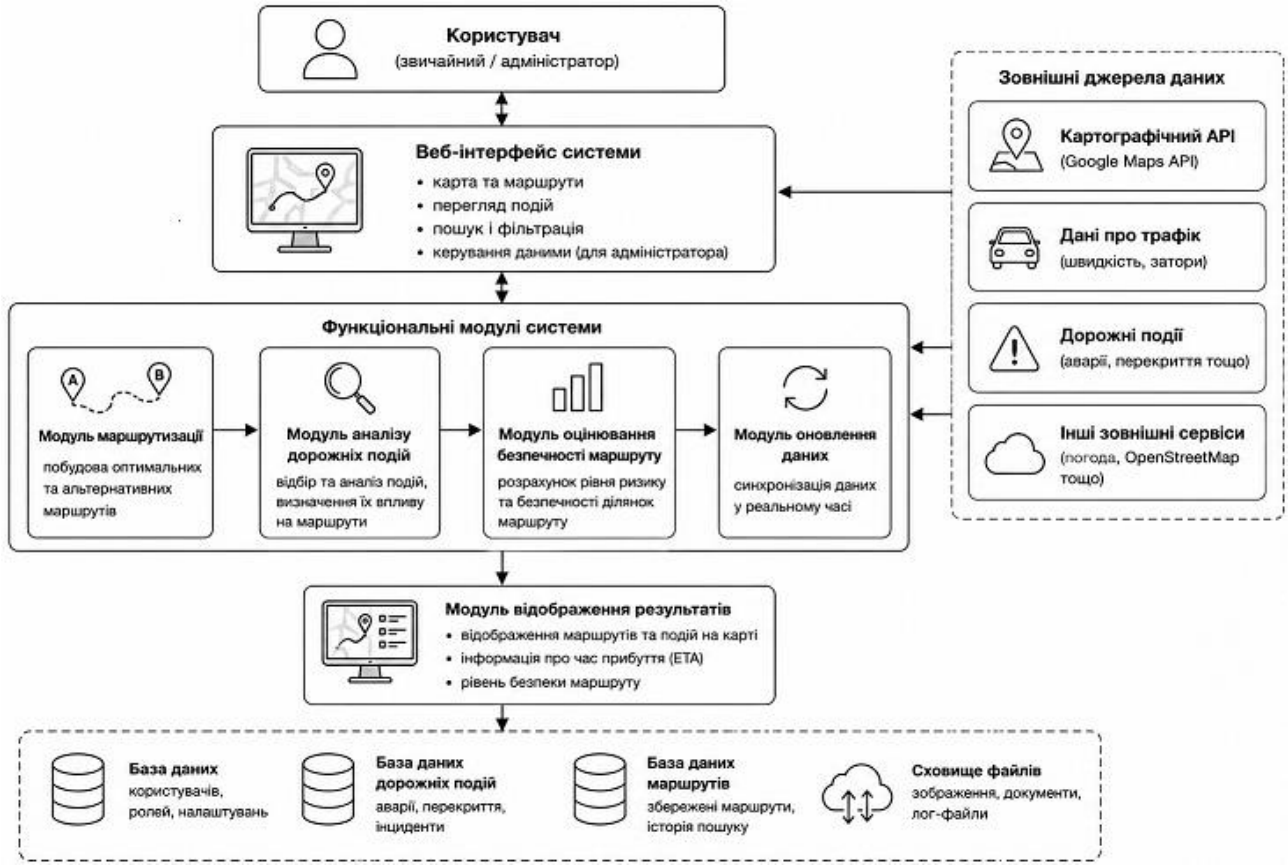

Рис. 1 Загальна структура системи моніторингу дорожньої ситуації

Як показано на рисунку 1.1, центральним елементом системи є модуль аналізу дорожніх подій, який взаємодіє з картографічними сервісами та забезпечує оцінювання безпечності побудованого маршруту. Це дозволяє не

лише відображати поточну дорожню ситуацію, а й виконувати додатковий аналіз транспортних ризиків.

Важливим аспектом предметної області є підтримка різних ролей користувачів. Звичайний користувач отримує можливість перегляду дорожньої ситуації, побудови маршрутів, аналізу альтернативних шляхів пересування та збереження обраних локацій. Адміністратор системи має розширені повноваження, що включають керування дорожніми подіями, редагування рівнів небезпечності, моделювання транспортних ситуацій та адміністрування інформаційної бази системи.

Таким чином, аналіз предметної області показав, що сучасні системи моніторингу дорожньої ситуації потребують поєднання навігаційного функціоналу з механізмами аналізу дорожніх подій та оцінювання безпечності маршрутів. Розроблення програмного забезпечення RoadGuard City спрямоване на підвищення ефективності пересування у міському середовищі, покращення інформативності навігації та зменшення впливу транспортних інцидентів на користувачів транспортної системи.

1.2 Вимоги до програмної системи

На основі проведеного аналізу предметної області та дослідження сучасних навігаційних сервісів було сформовано основні вимоги до програмного забезпечення моніторингу дорожньої ситуації та побудови маршрутів RoadGuard City. Визначення вимог є одним із ключових етапів розроблення програмної системи, оскільки саме на їх основі формується архітектура застосунку, визначаються функціональні можливості та обираються технології реалізації [8].

Розроблювана система повинна забезпечувати не лише побудову маршрутів між двома географічними точками, а й аналіз дорожньої ситуації у режимі реального часу з урахуванням дорожньо-транспортних пригод, транспортних подій та рівня завантаженості доріг. Особливістю програмного

забезпечення є поєднання класичних механізмів навігації із засобами аналізу аварійності маршруту та оцінювання рівня його безпечності.

Під час формування вимог до системи враховувалися особливості сучасних SPA-вебзастосунків, необхідність інтеграції із зовнішніми сервісами, підтримка масштабованості та забезпечення швидкої синхронізації даних. Окрема увага приділялася швидкодії системи, безпечному збереженню інформації та зручності користувацького інтерфейсу.

Функціональні вимоги визначають набір основних можливостей програмної системи та описують функції, які повинні бути реалізовані у застосунку. Основні функціональні вимоги наведено у таблиці 1.2.

Таблиця 1.2

Основні функціональні вимоги системи RoadGuard City

№	Функціональна вимога	Призначення
1	Реєстрація та авторизація користувачів	Забезпечення доступу до персоналізованого функціоналу системи
2	Побудова маршрутів	Формування маршруту між двома географічними точками
3	Альтернативні маршрути	Отримання декількох варіантів пересування
4	Відображення інтерактивної карти	Візуалізація маршрутів та дорожніх подій
5	Аналіз дорожніх подій	Визначення впливу транспортних інцидентів на маршрут
6	Розрахунок ETA	Обчислення орієнтовного часу прибуття
7	Оновлення даних у реальному часі	Актуалізація дорожньої інформації без перезавантаження сторінки
8	Збереження маршрутів	Формування історії маршрутів користувача
9	Робота з обраними місцями	Збереження важливих точок на карті
10	Розмежування ролей	Відокремлення функцій користувача та адміністратора
11	Адміністрування подій	Створення, редагування та видалення дорожніх подій
12	Інтеграція із зовнішніми API	Отримання картографічної та дорожньої інформації
13	Формування рекомендацій	Вибір найбільш безпечного або оптимального маршруту

Однією з ключових функціональних складових системи є модуль аналізу дорожньої ситуації, який виконує оцінювання впливу транспортних подій на швидкість і безпечність пересування. Для цього застосунок повинен підтримувати динамічний розрахунок ETA (Estimated Time Arrival), аналіз рівня небезпечності окремих ділянок дороги та формування рекомендацій щодо вибору маршруту [9].

Крім функціональних можливостей, програмне забезпечення повинно відповідати низці нефункціональних вимог, які визначають характеристики якості системи, її продуктивність, надійність та безпечність. Основні нефункціональні вимоги наведено у таблиці 1.3.

Таблиця 1.3

Нефункціональні вимоги до програмної системи

Категорія	Характеристика
Продуктивність	Швидке оновлення даних та стабільна робота у режимі реального часу
Масштабованість	Можливість збільшення кількості користувачів і дорожніх подій
Сумісність	Підтримка сучасних веббраузерів
Безпечність	Захист даних користувачів та адміністративного функціоналу
Надійність	Стабільна робота при високому навантаженні
Зручність використання	Інтуїтивно зрозумілий інтерфейс користувача
Модульність	Можливість подальшого розширення функціоналу
Геопросторова підтримка	Робота з координатами та геометричним аналізом маршрутів

Для реалізації зазначених вимог у системі використовується клієнт-серверна архітектура, у межах якої клієнтська частина функціонує у вигляді SPA-вебзастосунку. Для автентифікації користувачів та збереження інформації використовуються Firebase Authentication і Cloud Firestore, а побудова маршрутів та робота з геоданими реалізуються за допомогою Google Maps Platform та зовнішніх API дорожнього моніторингу [10].

Таким чином, сформовані функціональні та нефункціональні вимоги визначають основні принципи побудови програмної системи RoadGuard City та є основою для подальшого проєктування архітектури застосунку, структури бази даних і реалізації програмних модулів вебсистеми.

1.3 Моделювання предметної області

Для формалізації процесів функціонування системи моніторингу дорожньої ситуації та подальшого проєктування програмного забезпечення доцільним є використання UML-моделювання. UML (Unified Modeling Language) є універсальною мовою візуального моделювання, яка дозволяє представити структуру системи, взаємодію між її компонентами та логіку виконання основних процесів. Використання UML-діаграм у процесі розроблення програмного забезпечення забезпечує формалізацію функціональних вимог, спрощує аналіз предметної області та дозволяє сформувати цілісне уявлення про логіку роботи системи ще до етапу програмної реалізації [11].

У межах розроблення вебзастосунку RoadGuard City UML-моделі використовуються для опису взаємодії користувачів із системою, аналізу процесу побудови маршрутів, моделювання логіки обробки дорожніх подій та представлення послідовності виконання основних операцій. Такий підхід дозволяє не лише описати функціональні можливості системи, а й визначити взаємозв’язки між окремими підсистемами, порядок обробки даних та принципи функціонування програмних модулів.

Для моделювання предметної області у роботі використано діаграму прецедентів, діаграму послідовності та діаграму діяльності. Кожна з моделей описує окремий аспект функціонування системи та забезпечує комплексне представлення її поведінки.

Основні учасники взаємодії із системою наведені у табл. 1.4.

Таблиця 1.4

Основні актори системи RoadGuard City

Актор	Характеристика
Користувач	Виконує побудову маршрутів, перегляд карти, аналіз дорожньої ситуації та роботу зі збереженими маршрутами
Адміністратор	Керує дорожніми подіями, аваріями та інформацією про транспортні інциденти

На діаграмі прецедентів відображаються основні сценарії взаємодії користувачів із системою, а також функціональні можливості програмного забезпечення (рис. 2).

Рис. 2 Діаграма прецедентів системи RoadGuard City

На діаграмі прецедентів представлено двох основних акторів – користувача та адміністратора. Користувач взаємодіє із системою через вебінтерфейс і використовує функції побудови маршрутів, перегляду дорожньої ситуації, аналізу аварійності та роботи із збереженими точками на карті. Основним сценарієм взаємодії є процес формування маршруту, у межах якого система отримує декілька альтернативних варіантів пересування, аналізує дорожні події та виконує розрахунок часу маршруту.

Адміністратор представлений як користувач із розширеним рівнем доступу. Його функціональні можливості включають створення, редагування та видалення дорожніх подій, зміну рівнів небезпечності окремих ділянок дороги та моделювання транспортних ситуацій. Такий підхід забезпечує підтримку актуальності інформації та дозволяє виконувати тестування логіки аналізу маршрутів.

Для деталізації процесу побудови маршруту та взаємодії між окремими компонентами системи використовується діаграма послідовності (рис. 3).

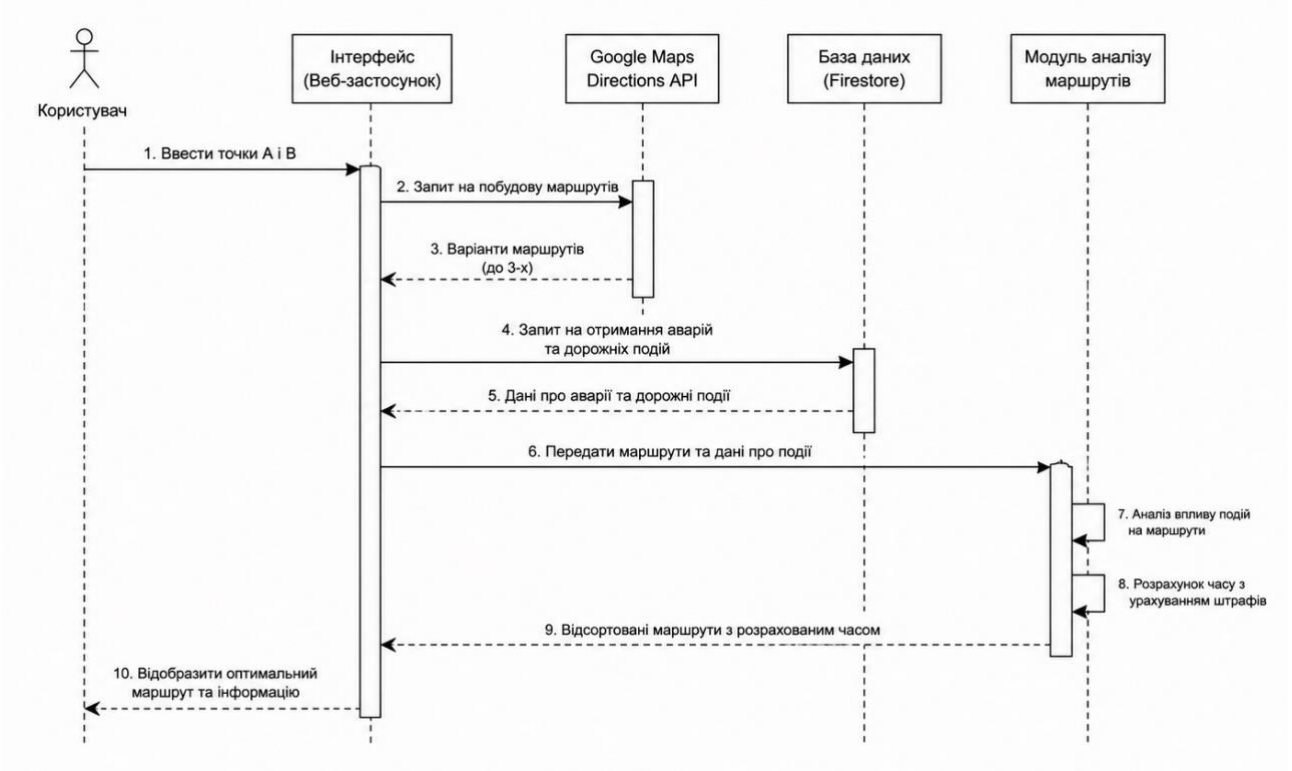

Рис. 3 Діаграма послідовності процесу побудови маршруту

Діаграма послідовності відображає алгоритм взаємодії користувача, вебінтерфейсу, картографічного сервісу та модуля аналізу дорожніх подій під час побудови маршруту. Процес починається із введення користувачем початкової та кінцевої точок маршруту. Після цього система звертається до зовнішнього картографічного API для отримання можливих маршрутів, виконує аналіз транспортної ситуації та визначає вплив дорожніх подій на кожен із маршрутів. На завершальному етапі система обчислює орієнтовний час прибуття, оцінює рівень безпечності пересування та відображає результати на інтерактивній карті.

Логіку виконання основних операцій системи відображає діаграма діяльності (рис. 4).

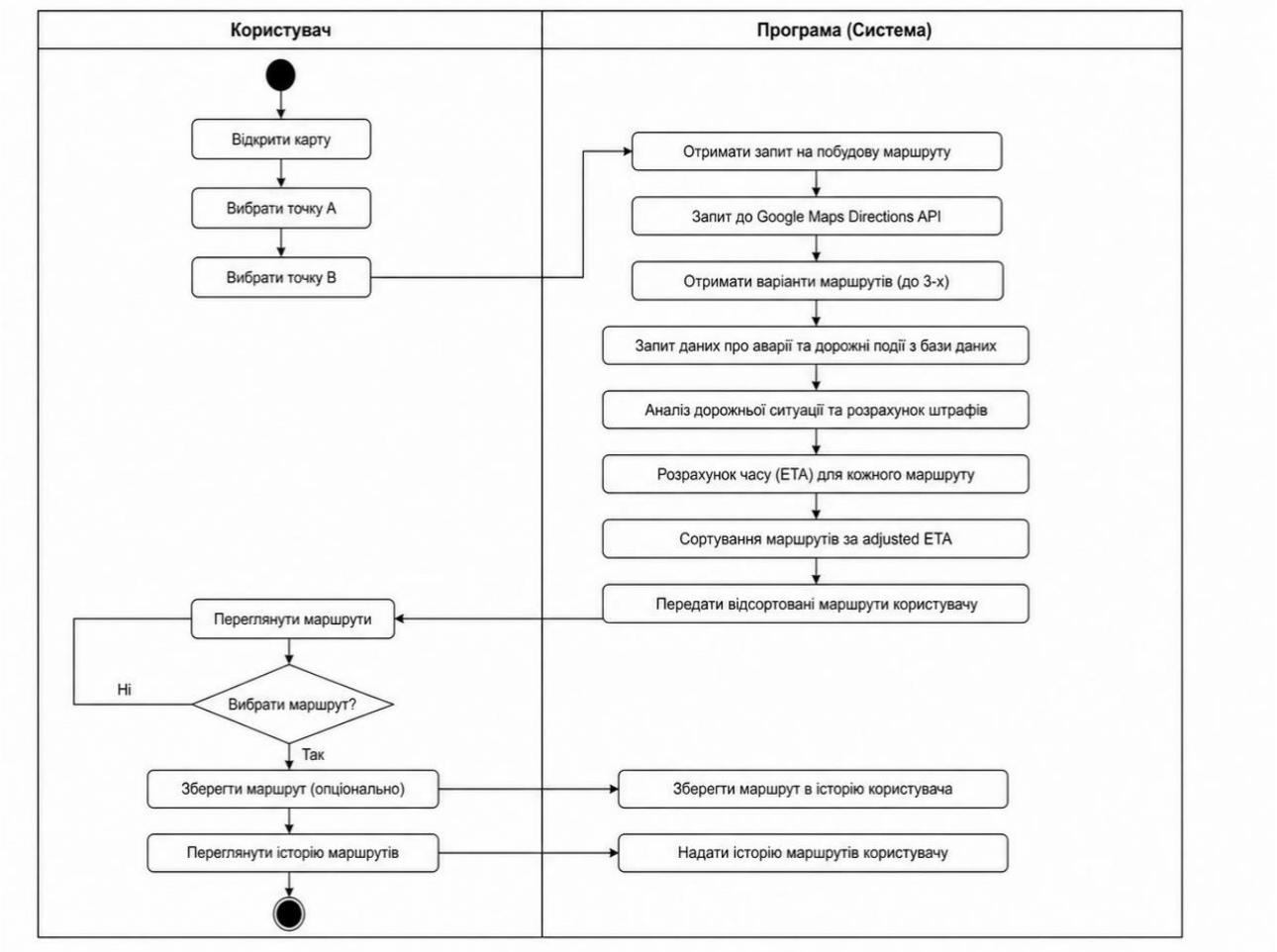

Рис. 4 Діаграма діяльності системи RoadGuard City

Діаграма діяльності демонструє послідовність виконання процесів під час аналізу дорожньої ситуації та побудови маршруту. Модель охоплює етапи

введення параметрів маршруту, отримання геоданих, аналізу транспортних подій, оцінювання небезпечності окремих ділянок та формування рекомендацій щодо вибору маршруту. Використання такої моделі дозволяє виявити логічні залежності між окремими процесами та визначити потенційні вузькі місця у роботі системи.

Основні сценарії функціонування програмної системи наведені у табл. 1.5.

Таблиця 1.5

Основні сценарії роботи системи

Сценарій	Результат виконання
Побудова маршруту	Формування одного або декількох маршрутів між точками
Аналіз дорожньої ситуації	Визначення впливу дорожніх подій на маршрут
Оцінювання безпечності	Розрахунок рівня ризику маршруту
Оновлення дорожніх подій	Актуалізація інформації у режимі реального часу
Адміністрування системи	Керування транспортними подіями та аваріями

Таким чином, моделювання предметної області дозволило формалізувати основні процеси функціонування системи RoadGuard City, визначити взаємодію користувачів із програмним забезпеченням та представити логіку роботи основних програмних модулів. Побудовані UML-моделі є основою для подальшого етапу проєктування архітектури системи, структури інформаційної бази та реалізації функціональних компонентів вебзастосунку.

1.4 Огляд наявних рішень

Для розроблення ефективної системи моніторингу дорожньої ситуації та побудови маршрутів необхідно провести аналіз існуючих програмних рішень у даній предметній області. Огляд сучасних навігаційних платформ дозволяє дослідити підходи до реалізації картографічних сервісів, оцінити функціональні можливості наявних систем, визначити їх переваги та недоліки, а також сформувати вимоги до розроблюваного програмного забезпечення [12].

Сучасні системи навігації забезпечують побудову маршрутів, аналіз транспортних потоків, визначення часу прибуття та відображення дорожніх подій у режимі реального часу. Найбільш поширеними рішеннями у цій сфері є Google Maps та Waze, які використовуються для навігації, моніторингу дорожнього руху та аналізу транспортної ситуації.

Проведення порівняльного аналізу дозволяє визначити обмеження існуючих систем та сформувати функціональні особливості програмного забезпечення RoadGuard City, орієнтованого не лише на побудову маршрутів, а й на аналіз аварійності та оцінювання безпечності пересування.

Одним із найпоширеніших навігаційних сервісів є Google Maps, який використовується для побудови маршрутів, роботи з географічними даними, аналізу дорожнього трафіку та визначення орієнтовного часу прибуття. Система підтримує декілька типів транспорту, дозволяє переглядати альтернативні маршрути та надає доступ до картографічних сервісів через Google Maps Platform API (рис. 5).

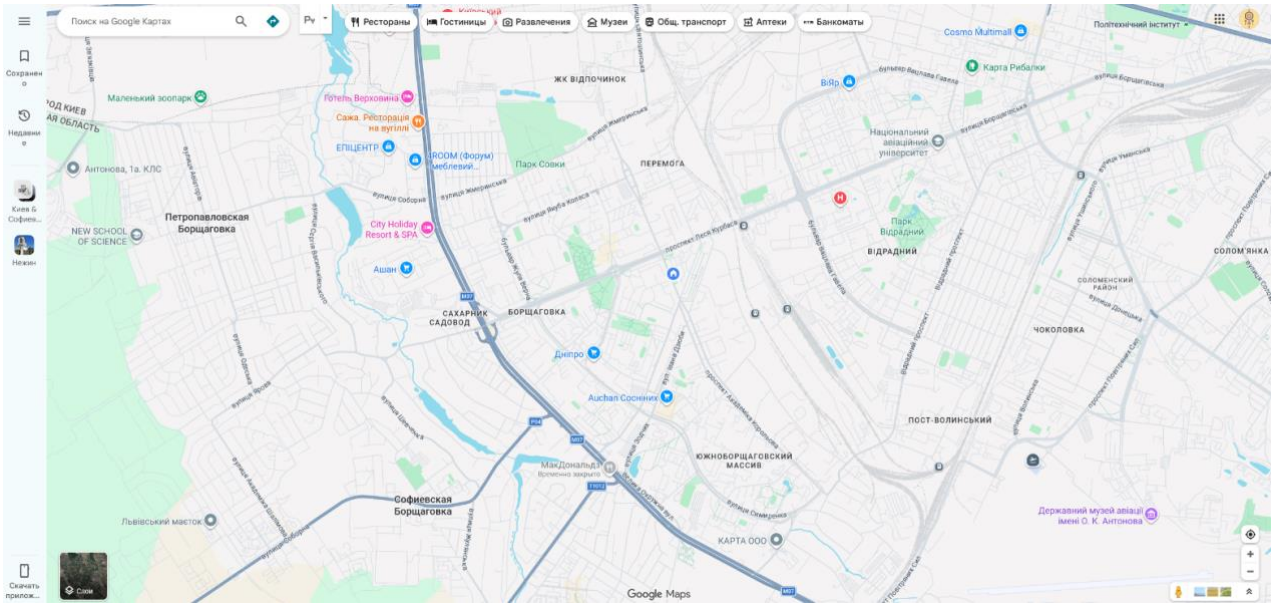

Рис. 5 Інтерфейс сервісу Google Maps

Основними перевагами Google Maps є висока точність картографічних даних, стабільність роботи сервісу, велике географічне покриття та швидке оновлення інформації про транспортну ситуацію. Крім того, система має

зручний користувацький інтерфейс та підтримує роботу на різних програмних платформах.

Разом із тим сервіс має низку обмежень. Google Maps орієнтований переважно на побудову найшвидшого маршруту та аналіз транспортного навантаження, однак не виконує комплексного оцінювання безпечності пересування. Інформація про аварії використовується здебільшого для коригування часу маршруту та не супроводжується детальним аналізом рівня ризику окремих ділянок дороги.

На рисунку 6 наведено приклад побудови маршруту у Google Maps.

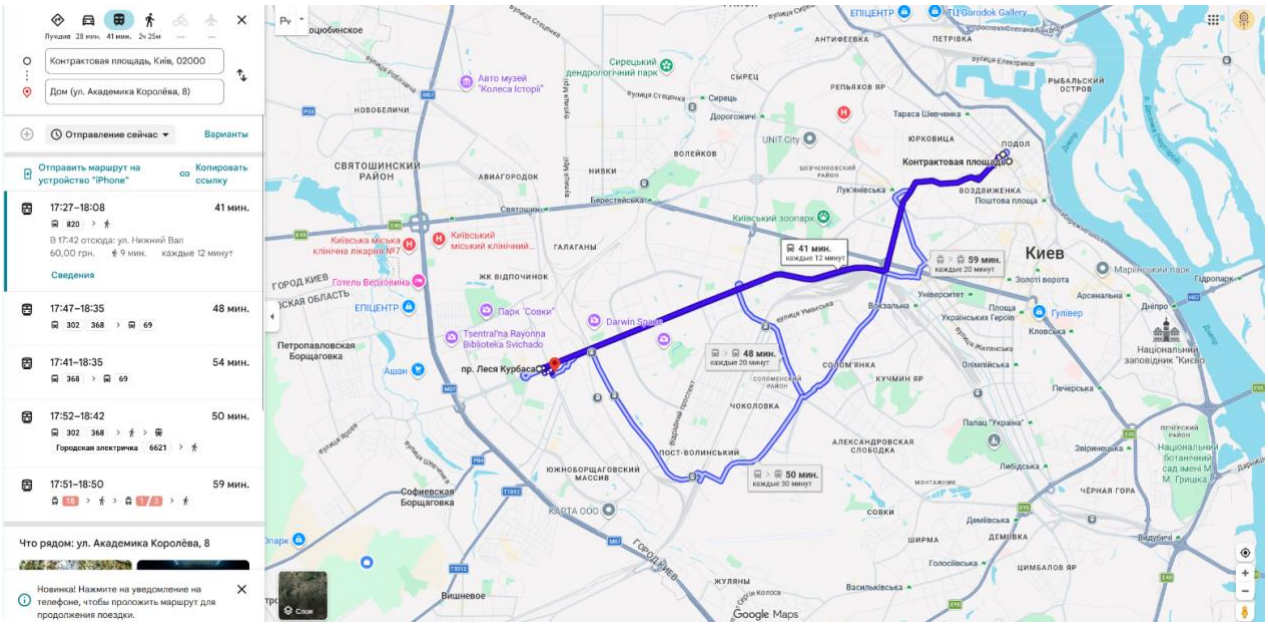

Рис. 6 Приклад побудови маршруту у Google Maps

Система не виконує оцінювання аварійності маршруту, не аналізує рівень небезпечності окремих дорожніх ділянок та не забезпечує механізмів ранжування маршрутів за показниками безпечності. Крім того, розробник має обмежений доступ до внутрішніх алгоритмів аналізу транспортної ситуації, а частина функціоналу API є платною та має тарифні обмеження.

Ще одним популярним навігаційним сервісом є Waze, який побудований на принципах краудсорсингу та активної участі користувачів у формуванні дорожньої інформації. Користувачі системи можуть повідомляти про аварії,

перекриття руху, ремонтні роботи, затори та інші транспортні події у режимі реального часу (рис. 7).

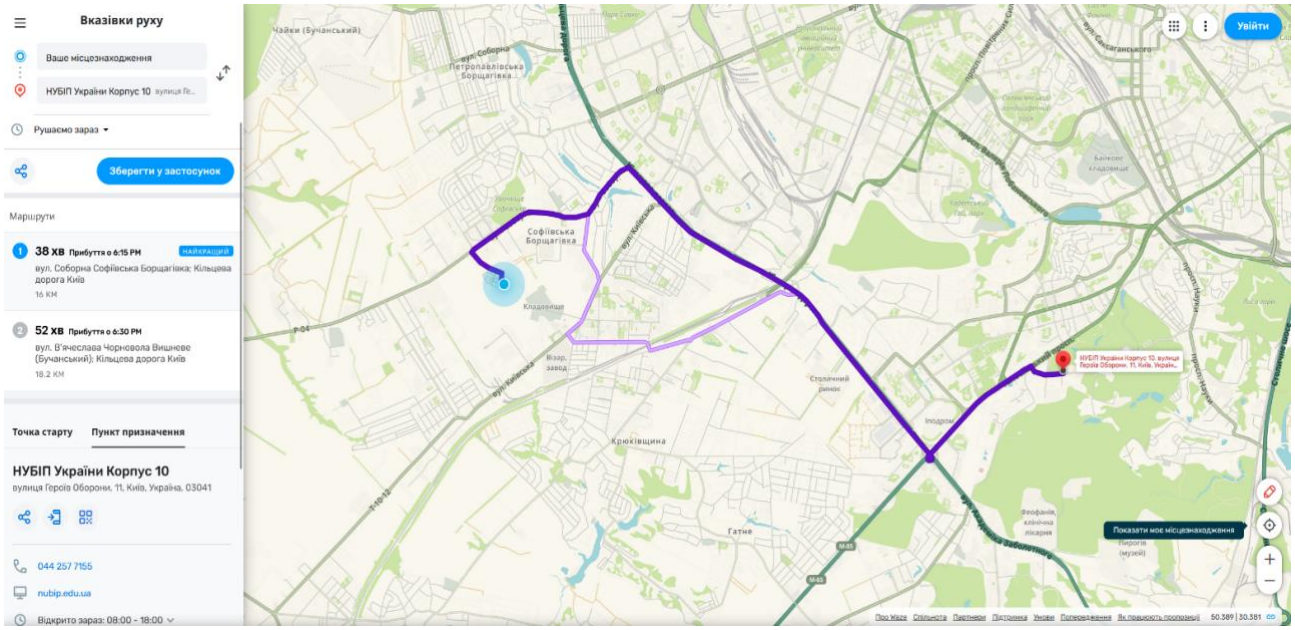

Рис. 7 Інтерфейс сервісу Waze

Перевагою Waze є оперативне оновлення дорожньої інформації та можливість швидкого реагування на зміни транспортної ситуації. Завдяки великій кількості активних користувачів система здатна автоматично пропонувати альтернативні маршрути при виникненні заторів або дорожніх інцидентів.

Водночас функціонування Waze значною мірою залежить від активності користувачів. У регіонах із низькою кількістю активних учасників дорожні події можуть оновлюватися із затримкою або не фіксуватися взагалі. Крім того, інформація, отримана від користувачів, не завжди проходить достатню перевірку достовірності, що може призводити до появи помилкових або неактуальних повідомлень.

Суттєвим недоліком Waze є також відсутність комплексного аналізу впливу дорожніх подій на маршрут користувача. Система відображає факт наявності аварії чи затору, однак не виконує оцінювання рівня ризику маршруту та не аналізує ступінь небезпечності окремих ділянок дороги.

Порівняльну характеристику розглянутих систем наведено у табл. 1.6.

Таблиця 1.6

Порівняння сучасних навігаційних систем

Характеристика	Google Maps	Waze	RoadGuard City
Побудова маршрутів	+	+	+
Альтернативні маршрути	+	+	+
Аналіз дорожнього трафіку	+	+	+
Аналіз аварійності маршруту	–	–	+
Оцінювання безпечності маршруту	–	–	+
Робота у режимі реального часу	+	+	+
Механізм керування подіями	Обмежений	Частковий	+
Інтеграція із зовнішніми API	+	Частково	+
Аналіз впливу подій на маршрут	Частковий	Частковий	+

Проведений аналіз показав, що сучасні навігаційні системи мають широкий набір функціональних можливостей, однак більшість із них орієнтована переважно на побудову найшвидшого маршруту та аналіз транспортного навантаження. При цьому недостатньо уваги приділяється оцінюванню безпечності пересування, аналізу аварійності маршрутів та визначенню впливу дорожніх подій на рівень ризику.

На відміну від існуючих рішень, система RoadGuard City поєднує навігаційні функції із механізмами аналізу дорожніх інцидентів, оцінюванням безпечності маршруту та підтримкою роботи у режимі реального часу. Такий підхід дозволяє підвищити інформативність навігації, покращити якість аналізу дорожньої ситуації та забезпечити більш безпечне пересування користувачів транспортної системи.

1.5 Постановка задачі дослідження

Проведений аналіз предметної області систем моніторингу дорожньої ситуації, дослідження сучасних навігаційних платформ та моделювання основних процесів функціонування програмного забезпечення дозволили визначити ключові проблеми, характерні для існуючих систем побудови маршрутів. Більшість сучасних навігаційних сервісів орієнтовані переважно на мінімізацію часу пересування або довжини маршруту, однак недостатньо враховують рівень безпечності дорожніх ділянок, вплив транспортних подій та аварійність маршруту. У результаті користувач отримує маршрут, який є оптимальним лише за часовими показниками, але не забезпечує достатнього рівня безпечності пересування.

В умовах зростання транспортного навантаження та збільшення кількості дорожньо-транспортних пригод актуальним є створення програмного забезпечення, здатного поєднувати функції навігації, моніторингу дорожньої ситуації та аналізу транспортних подій у режимі реального часу. Така система повинна забезпечувати не лише побудову маршрутів, а й оцінювання впливу дорожніх інцидентів на безпечність пересування користувачів.

Метою бакалаврської кваліфікаційної роботи є розроблення веборієнтованої системи моніторингу дорожньої ситуації RoadGuard City, яка забезпечує побудову маршрутів, аналіз транспортних подій та оцінювання безпечності пересування у міському середовищі.

Для досягнення поставленої мети у роботі необхідно вирішити такі задачі:

- провести аналіз предметної області та сучасних систем навігації;
- визначити функціональні та нефункціональні вимоги до програмного забезпечення;
- виконати моделювання предметної області та основних процесів функціонування системи;
- спроєктувати структуру інформаційної бази та архітектуру вебзастосунку;

- реалізувати механізми побудови маршрутів, аналізу дорожніх подій та оцінювання безпечності маршруту;
- забезпечити інтеграцію із зовнішніми картографічними сервісами та API дорожнього моніторингу;
- виконати тестування та перевірку працездатності програмної системи.

Об’єктом дослідження є процеси моніторингу дорожньої ситуації та побудови маршрутів у міському транспортному середовищі.

Предметом дослідження є методи, моделі та програмні засоби аналізу дорожніх подій, оцінювання безпечності маршрутів і реалізації веборієнтованих навігаційних систем.

Практичне значення отриманих результатів полягає у створенні програмної системи, яка може бути використана для аналізу дорожньої ситуації, побудови альтернативних маршрутів та підвищення ефективності пересування користувачів у міському середовищі. Реалізований вебзастосунок забезпечує підтримку роботи у режимі реального часу, інтеграцію із картографічними сервісами та можливість подальшого розширення функціональних можливостей системи.

Таким чином, постановка задачі дослідження визначає основні напрями проєктування та програмної реалізації системи RoadGuard City, орієнтованої на підвищення ефективності моніторингу дорожньої ситуації та покращення безпечності пересування користувачів.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ ЧАСТИНИ

2.1 Проєктування інформаційного забезпечення та вибір СУБД

Одним із ключових етапів розроблення системи моніторингу завантаженості та аварійності на дорогах міст є проєктування інформаційного забезпечення, оскільки саме від структури зберігання даних, способів їх обробки та механізмів синхронізації залежить стабільність роботи програмного забезпечення, швидкість реагування на зміну дорожньої ситуації та можливість масштабування системи у процесі подальшого розвитку. Для веборієнтованого застосунку RoadGuard City особливого значення набуває підтримка роботи з геопросторовими даними, оперативне оновлення інформації про дорожні події та забезпечення одночасної взаємодії великої кількості користувачів із системою.

Предметна область дослідження характеризується значною динамічністю даних. Інформація про дорожньо-транспортні пригоди, затори, перекриття руху та інші транспортні інциденти постійно змінюється, а тому система повинна забезпечувати можливість оперативного оновлення даних без необхідності повторного завантаження сторінок або ручної синхронізації інформації. Крім того, структура даних у системі не є повністю фіксованою, оскільки у процесі розвитку програмного забезпечення передбачається розширення функціональності, додавання нових типів транспортних подій, аналітичних параметрів та механізмів оцінювання дорожньої ситуації.

З урахуванням зазначених особливостей для реалізації інформаційного забезпечення системи було обрано документно-орієнтовану NoSQL СУБД Cloud Firestore, яка входить до складу платформи Firebase. Використання Firestore дозволяє забезпечити роботу із напівструктурованими даними, підтримку realtime-синхронізації та гнучке масштабування вебзастосунку без необхідності розгортання окремої серверної інфраструктури. Такий підхід є доцільним для

систем моніторингу дорожньої ситуації, де основна частина інформації представлена у вигляді подій, координат, маршрутів та тимчасових записів.

На відміну від традиційних реляційних систем керування базами даних, у яких структура таблиць визначається заздалегідь і потребує складних процедур модифікації, Firestore використовує документну модель зберігання даних, що дозволяє змінювати структуру документів без перебудови всієї бази даних. Це є важливою перевагою для системи RoadGuard City, оскільки у процесі її розвитку може виникати необхідність розширення набору параметрів дорожніх подій, модифікації механізмів аналізу маршрутів або інтеграції із новими зовнішніми сервісами.

У межах Firestore дані організовуються у вигляді колекцій та документів. Для системи RoadGuard City основними колекціями є users, accidents, trafficEvents, routesHistory та favoritePlaces. Кожна з них відповідає окремій функціональній підсистемі та використовується для зберігання певного типу інформації.

Колекція users призначена для зберігання інформації про користувачів системи. У ній містяться ідентифікатор користувача, адреса електронної пошти, роль доступу та часові мітки створення облікового запису. Наявність ролей дозволяє реалізувати механізм розмежування доступу між звичайними користувачами та адміністраторами системи.

Колекція accidents використовується для зберігання інформації про дорожньо-транспортні пригоди. Для кожного запису зберігаються координати події, опис інциденту, рівень небезпечності, джерело отримання інформації та часові параметри створення або оновлення запису. Дані цієї колекції застосовуються під час аналізу маршрутів та відображення аварій на інтерактивній карті.

Для підтримки механізмів синхронізації дорожніх подій у режимі реального часу використовується колекція liveAccidents, яка містить актуальні транспортні інциденти, отримані із зовнішніх API або через механізми автоматичного оновлення. Використання окремої колекції для live-даних

дозволяє розділити статичні та динамічні транспортні події та підвищити ефективність обробки інформації.

Колекція trafficEvents використовується для зберігання дорожніх подій, представлених у вигляді точок, ліній або полігональних зон. Для кожної події зберігаються геометричні параметри об’єкта, тип транспортної ситуації, рівень ризику та часові межі дії події. Такий підхід дозволяє виконувати аналіз перетину маршруту з небезпечними ділянками дороги та оцінювати вплив дорожніх інцидентів на процес пересування користувача.

Колекція routesHistory забезпечує збереження історії побудованих маршрутів. У ній містяться дані про початкову та кінцеву точки маршруту, тип транспорту, тривалість пересування, скоригований час маршруту та ідентифікатор користувача. Наявність цієї колекції дозволяє реалізувати функціональність перегляду історії маршрутів і створює основу для подальшого аналізу поведінки користувачів системи.

Для реалізації персоналізованого функціоналу використовується колекція favoritePlaces, у якій зберігаються обрані користувачем місця, зокрема дім, робота або інші важливі точки. Для кожного запису фіксуються координати, назва місця, тип мітки та ідентифікатор власника.

Структуру основних колекцій інформаційної бази наведено у таблиці 2.1.

Таблиця 2.1

Основні колекції бази даних системи RoadGuard City

Колекція	Призначення
users	Зберігання інформації про користувачів та ролі доступу
accidents	Дані про дорожньо-транспортні пригоди
liveAccidents	Актуальні дорожні інциденти у режимі реального часу
trafficEvents	Інформація про транспортні події та небезпечні ділянки
routesHistory	Історія маршрутів користувачів
favoritePlaces	Збережені користувачем географічні точки

Важливою перевагою Firestore є підтримка синхронізації даних у режимі реального часу через механізм onSnapshot(), який дозволяє автоматично отримувати оновлення при зміні документів у колекціях. Завдяки цьому

користувачі системи отримують актуальну інформацію про дорожню ситуацію без необхідності перезавантаження вебсторінки. Для системи моніторингу дорожнього руху така функціональність є критично важливою, оскільки оперативність оновлення даних безпосередньо впливає на точність аналізу маршрутів та ефективність навігації.

Ще однією важливою причиною вибору Firestore стала інтеграція із сервісом Firebase Authentication, який забезпечує механізми реєстрації, авторизації та контролю доступу користувачів. Для захисту даних використовуються Firebase Security Rules, що дозволяють обмежити доступ до редагування дорожніх подій лише для користувачів із роллю адміністратора. Такий підхід підвищує рівень безпеки системи та запобігає несанкціонованій зміні критично важливої інформації.

На рисунку 8 наведено загальну структуру інформаційного забезпечення системи RoadGuard City.

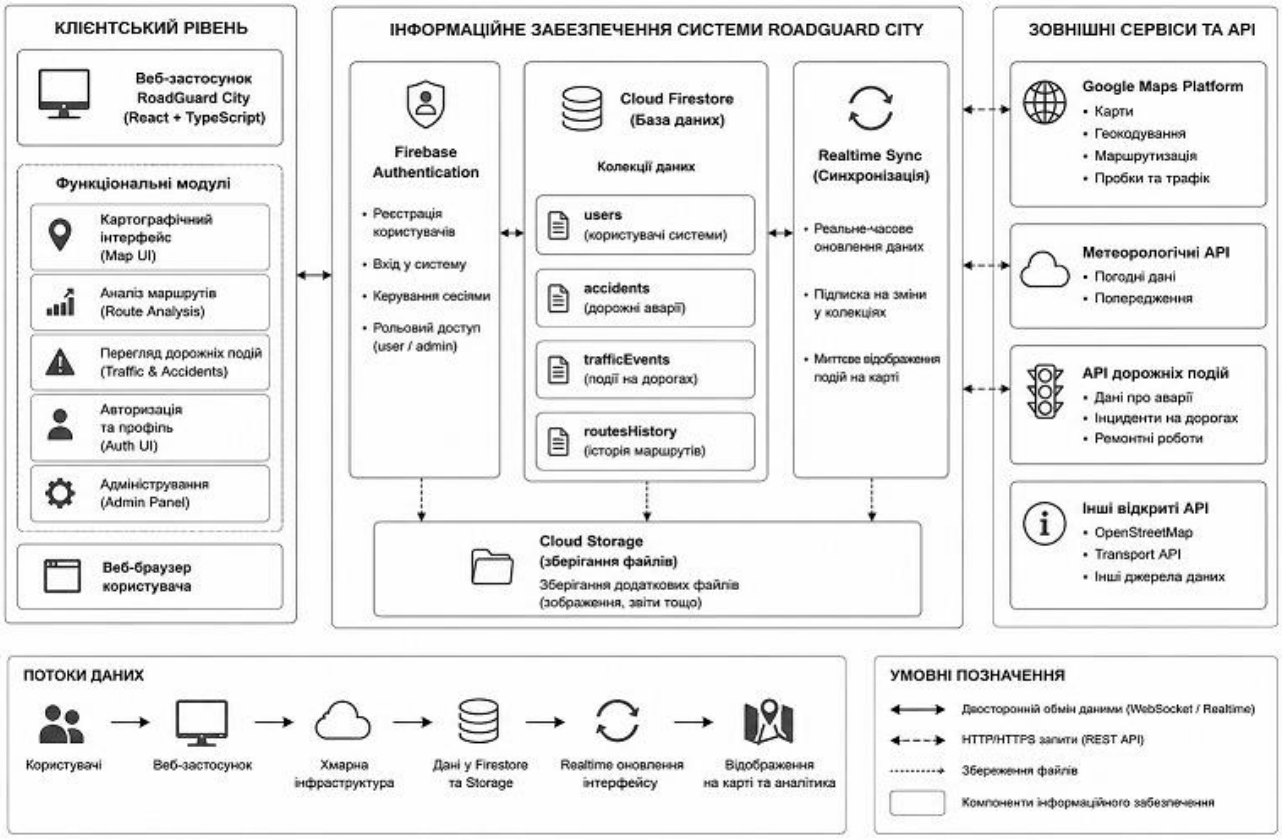

Рис. 8 Загальна структура інформаційного забезпечення системи RoadGuard City

Використання Firestore також дозволило спростити архітектуру програмного забезпечення, оскільки взаємодія клієнтської частини із базою даних здійснюється безпосередньо через Firebase SDK без необхідності створення окремого серверного API для виконання базових CRUD-операцій. Це дозволило зосередити основну увагу на реалізації механізмів аналізу дорожньої ситуації, побудови маршрутів та інтеграції із картографічними сервісами.

Для адміністрування інформаційної бази використовується Firebase Console, яка забезпечує графічний інтерфейс для перегляду колекцій, редагування документів, налаштування правил безпеки та контролю роботи системи. Використання даного середовища спрощує процес тестування структури бази даних та дозволяє оперативно перевіряти актуальність синхронізації дорожніх подій.

Таким чином, використання Cloud Firestore як системи керування базою даних є обґрунтованим рішенням для вебзастосунку RoadGuard City, оскільки забезпечує підтримку роботи у режимі реального часу, гнучкість структури даних, масштабованість, безпечне зберігання інформації та ефективну інтеграцію із сучасними вебтехнологіями. Обрана архітектура інформаційного забезпечення відповідає функціональним вимогам системи та створює основу для подальшого розвитку програмного забезпечення.

2.2 Структура бази даних, таблиці та зв’язки

Після вибору системи зберігання даних наступним етапом проєктування інформаційної частини є формування структури бази даних та визначення взаємозв’язків між основними сутностями системи. Для системи моніторингу дорожнього руху RoadGuard City структура бази даних повинна забезпечувати швидку обробку дорожніх подій, підтримку роботи у режимі реального часу та зручне зберігання геопросторових даних.

Оскільки у проєкті використовується Cloud Firestore, структура бази даних реалізована у вигляді колекцій та документів. Такий підхід дозволяє гнучко

організувати дані та спростити масштабування системи у майбутньому. Документно-орієнтована модель забезпечує ефективну роботу з напівструктурованими даними, масивами координат та динамічними об’єктами дорожньої інфраструктури, що є важливим для систем моніторингу транспортної ситуації.

Основною колекцією системи є users (рис. 9). Вона використовується для зберігання інформації про користувачів веб-застосунку.

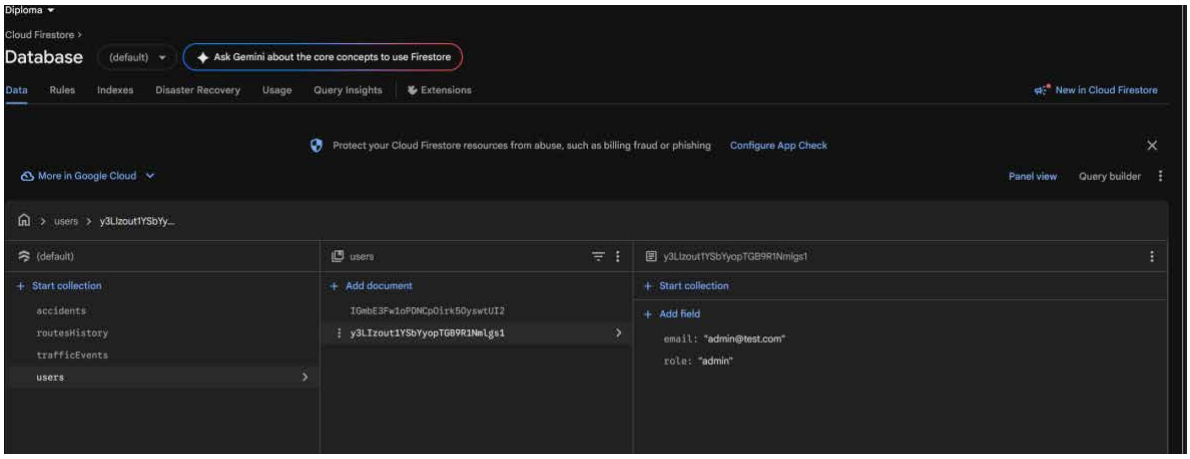

Рис. 9 Структура колекції users у Cloud Firestore

Для кожного користувача у базі даних зберігається унікальний ідентифікатор, електронна пошта, роль користувача та дата створення облікового запису (рис. 10). Роль визначає рівень доступу до функціоналу системи.

```
import { doc, getDoc, setDoc } from 'firebase/firestore';
import { db } from './firebase';

export type UserRole = 'user' | 'admin';

export const resolveRoleByEmail = (email: string | null | undefined): UserRole =>
  email === 'admin@test.com' ? 'admin' : 'user';

export const ensureUserProfile = async (uid: string, email: string) => {
  const ref = doc(db, 'users', uid);
  const snap = await getDoc(ref);
  const role = resolveRoleByEmail(email);
  if (!snap.exists()) {
    await setDoc(ref, { email, role });
    return role;
  }
  return (snap.data().role as UserRole) ?? role;
};
```

Рис. 10 Профілі Users

Користувачі з роллю user можуть будувати маршрути, переглядати дорожню ситуацію та зберігати обрані місця, тоді як користувачі з роллю admin мають доступ до адміністративного функціоналу та керування дорожніми подіями. Підключення бази даних та механізму автентифікації Firebase Authentication наведено на рис. 11.

```
aimport { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
import { getFirestore } from 'firebase/firestore';

const firebaseConfig = {
  apiKey: import.meta.env.VITE_FIREBASE_API_KEY,
  authDomain: import.meta.env.VITE_FIREBASE_AUTH_DOMAIN,
  projectId: import.meta.env.VITE_FIREBASE_PROJECT_ID,
  storageBucket: import.meta.env.VITE_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: import.meta.env.VITE_FIREBASE_MESSAGING_SENDER_ID,
  appId: import.meta.env.VITE_FIREBASE_APP_ID,
};

const requiredKeys = ['apiKey', 'authDomain', 'projectId', 'appId'] as const;
for (const key of requiredKeys) {
  if (!firebaseConfig[key]) {
    // Throw early to make missing env setup obvious.
    throw new Error(`Missing Firebase config: ${key}`);
  }
}

export const firebaseApp = initializeApp(firebaseConfig);
export const auth = getAuth(firebaseApp);
export const db = getFirestore(firebaseApp);
```

Рис. 11 Підключення БД і AUTH

Колекція accidents використовується для зберігання інформації про дорожньо-транспортні пригоди (рис. 12). Кожен документ містить координати аварії, опис події, рівень небезпечності та часові параметри створення запису.

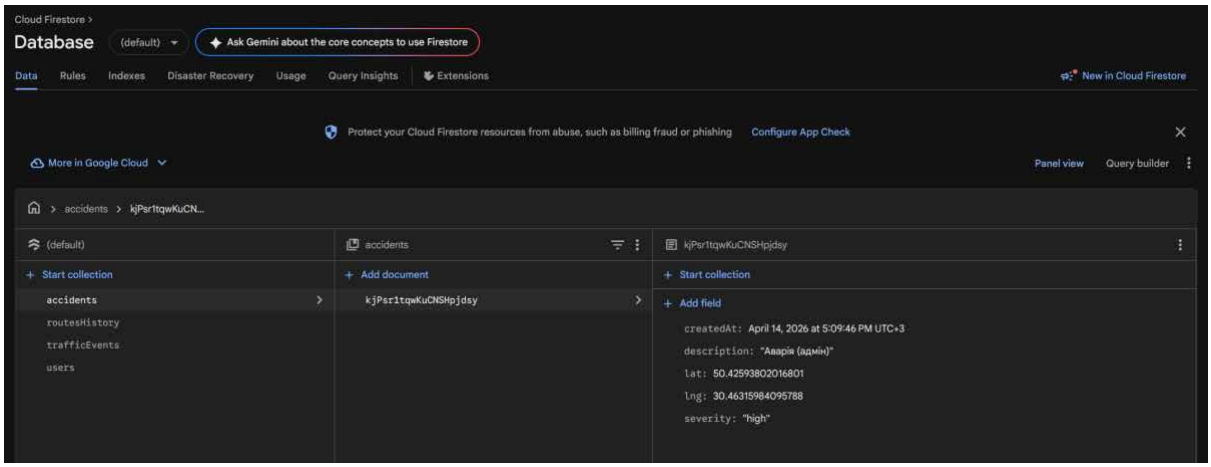

Рис. 12 Структура документа колекції accidents

Дані цієї колекції використовуються під час відображення аварій на інтерактивній карті та аналізу їх впливу на побудований маршрут. Інформація про аварії враховується модулем аналізу маршрутів під час визначення безпечності пересування та розрахунку скоригованого часу маршруту.

Для підтримки роботи системи у режимі реального часу використовується колекція liveAccidents. Вона містить актуальні дорожні інциденти, які надходять через механізми realtime-синхронізації або зовнішні API. У документах колекції зберігаються координати події, опис інциденту, рівень небезпечності, джерело даних та час оновлення інформації. Завдяки використанню Firestore система автоматично синхронізує зміни між усіма активними користувачами без необхідності перезавантаження сторінки.

Колекція trafficEvents використовується для зберігання дорожніх подій, представлених у вигляді точок, ліній або полігональних зон. Для кожного документа зберігаються тип події, геометричні параметри, масив координат, рівень ризику та часові мітки створення. Такий підхід дозволяє виконувати перевірку перетину маршруту із небезпечними ділянками дороги та визначати вплив транспортних подій на процес пересування користувача. Приклад структури документа колекції trafficEvents наведено на рис. 13.

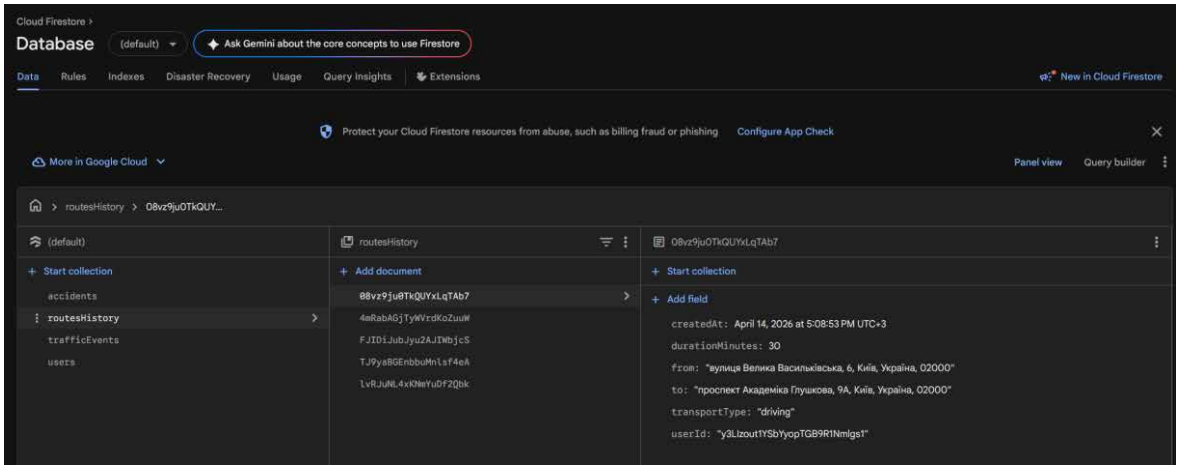

Рис. 13 Структура документа колекції routesHistory

Для збереження історії маршрутів використовується колекція routesHistory. У ній містяться дані про початкову та кінцеву точки маршруту, тип транспорту, тривалість маршруту, adjusted ETA та часові параметри створення

запису. Наявність даної колекції дозволяє реалізувати функціонал перегляду попередніх маршрутів, аналізу історії поїздок та повторного використання маршрутів користувачем. Структуру документа колекції routesHistory наведено на рис. 14.

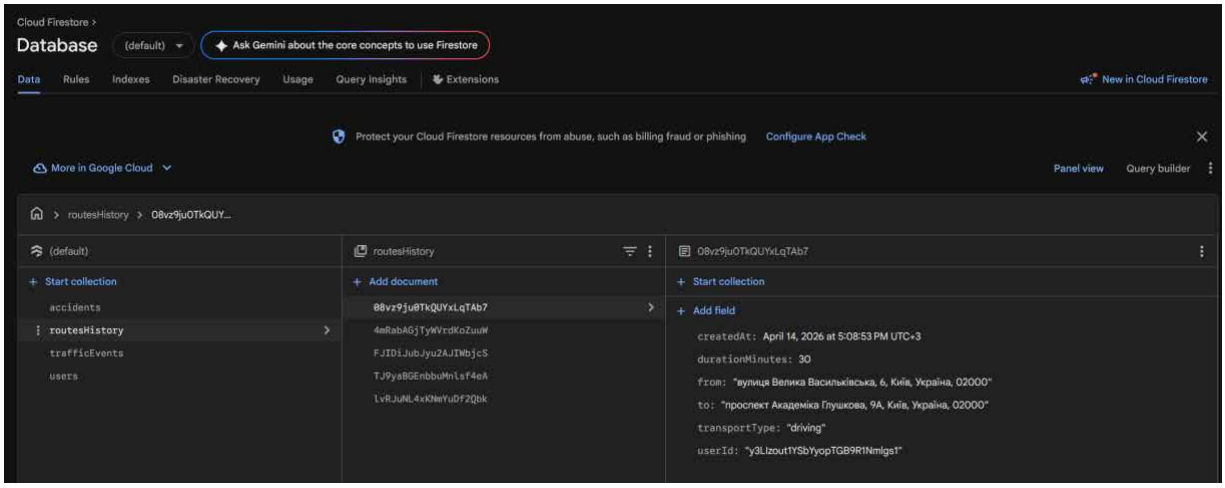

Рис. 14 Структура документа колекції routesHistory

Колекція favoritePlaces використовується для збереження обраних користувачем місць. Для кожного документа зберігаються назва точки, географічні координати, тип місця та ідентифікатор користувача. Це дозволяє реалізувати механізм швидкого доступу до найбільш використовуваних маршрутів та забезпечує персоналізацію роботи веб-застосунку.

Основні характеристики колекцій інформаційної бази наведено у табл. 2.2.

Таблиця 2.2

Основні колекції бази даних системи RoadGuard City

Колекція	Призначення	Основні поля
users	Дані користувачів та ролей доступу	email, role, createdAt
accidents	Інформація про дорожньо-транспортні пригоди	coordinates, severity, description
liveAccidents	Актуальні дорожні інциденти у realtime-режимі	coordinates, updatedAt, source
trafficEvents	Дані про затори, перекриття та небезпечні зони	geometry, riskLevel, eventType
routesHistory	Історія побудованих маршрутів	startPoint, endPoint, duration
favoritePlaces	Обрані користувачем місця	title, coordinates, userId

Між колекціями системи існують логічні взаємозв’язки. Колекції routesHistory та favoritePlaces пов’язані із колекцією users через ідентифікатор користувача. Колекції accidents, liveAccidents та trafficEvents використовуються спільно модулем аналізу маршрутів для оцінювання дорожньої ситуації та визначення рівня безпечності пересування. Загальну структуру зв’язків між основними сутностями системи наведено на рис. 15.

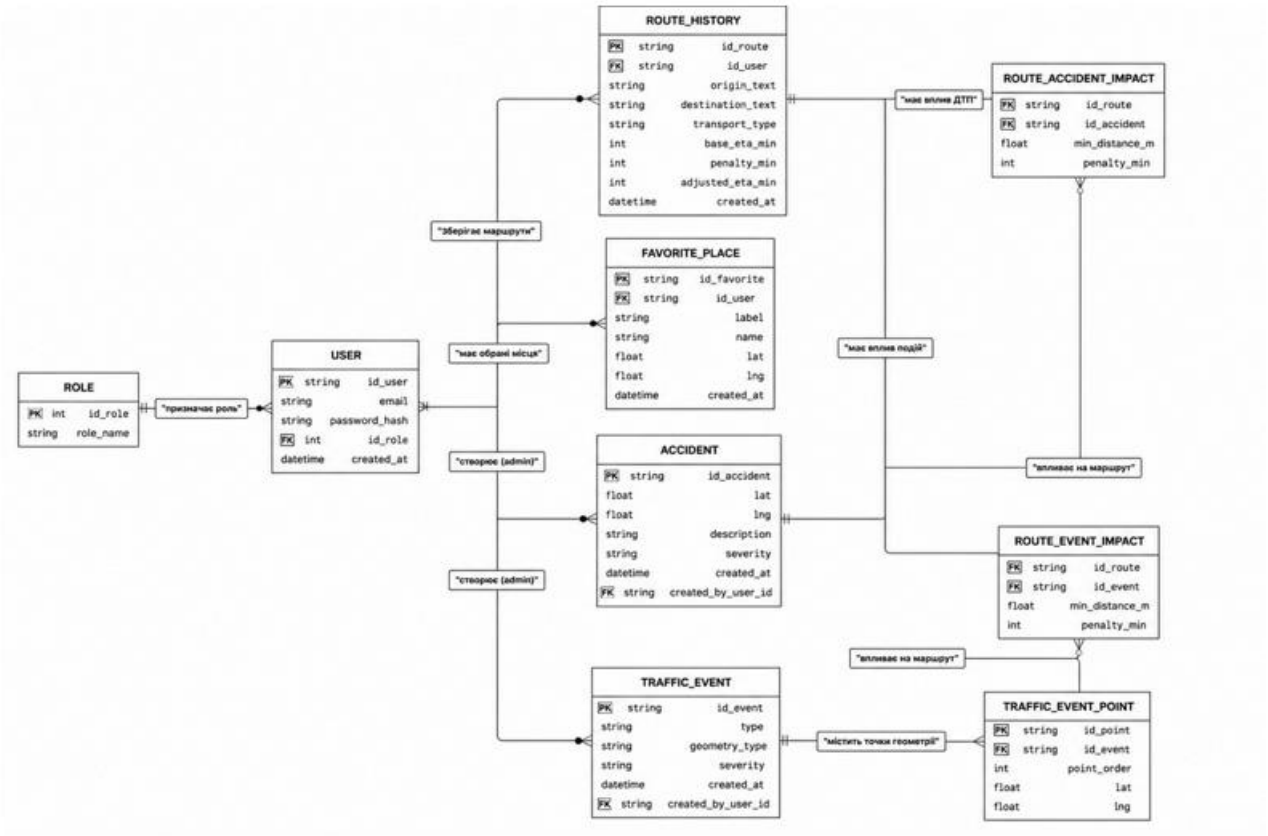

Рис. 15 ER-модель структури бази даних системи RoadGuard City

Під час побудови маршруту система одночасно взаємодіє з декількома джерелами даних. Після отримання маршруту через Google Maps Directions API виконується завантаження інформації про аварії та дорожні події із Firestore. Далі модуль аналізу маршрутів перевіряє перетин маршруту із небезпечними зонами, виконує оцінювання впливу дорожніх подій на маршрут та розрахунок скоригованого ETA. У разі виявлення критичних подій система може рекомендувати альтернативний маршрут або попередити користувача про підвищений рівень ризику.

Використання документної структури Firestore дозволяє швидко змінювати формат даних та додавати нові поля без необхідності модифікації всієї структури бази даних. Це є важливою перевагою для системи RoadGuard City, оскільки у майбутньому передбачається розширення функціоналу та інтеграція з додатковими сервісами дорожнього моніторингу. Крім того, використання Firebase Security Rules забезпечує контроль доступу до даних та обмежує можливість створення або редагування дорожніх подій лише користувачами з адміністративними правами.

Таким чином, структура бази даних системи RoadGuard City забезпечує ефективне зберігання інформації про користувачів, маршрути та дорожні події, підтримує роботу у режимі реального часу та дозволяє реалізувати механізми аналізу транспортної ситуації. Сформована інформаційна модель є основою для подальшої реалізації модулів синхронізації даних, аналізу маршрутів та інтеграції веб-застосунку із зовнішніми картографічними сервісами.

2.3 Реалізація механізмів обробки та синхронізації даних

Однією з ключових особливостей веб-застосунку RoadGuard City є необхідність безперервного оновлення інформації про дорожню ситуацію та забезпечення узгодженості даних між клієнтською частиною, серверними сервісами і хмарною базою даних. Оскільки система орієнтована на моніторинг дорожніх подій у режимі реального часу, механізми обробки та синхронізації даних безпосередньо впливають на коректність побудови маршрутів, точність прогнозування часу пересування та стабільність роботи програмного забезпечення. Для реалізації даного функціоналу у проєкті використано документо-орієнтовану базу даних Firebase Firestore, яка підтримує realtime-синхронізацію та асинхронний обмін інформацією між усіма активними клієнтами [13].

На відміну від класичних реляційних систем, де оновлення інформації часто реалізується через SQL-запити, тригери або серверні процедури, у Firestore

механізм синхронізації базується на постійному відстеженні змін у колекціях даних. Для цього застосовуються realtime-слухачі onSnapshot(), що входять до складу Firebase SDK. Після підключення до відповідної колекції клієнтська частина автоматично отримує оновлені дані при зміні документів, додаванні нових записів або видаленні існуючих елементів. Такий підхід дозволяє мінімізувати затримки між появою нової дорожньої події та її відображенням на карті користувача.

У процесі роботи застосунок підтримує одночасну синхронізацію декількох основних колекцій бази даних: accidents, liveAccidents, trafficEvents, routesHistory та favoritePlaces. Колекції accidents і liveAccidents використовуються для зберігання інформації про дорожньо-транспортні пригоди та інші критичні події, які можуть впливати на безпечність маршруту. Колекція trafficEvents містить інформацію про затори, перекриття руху, ремонтні роботи та інші фактори дорожньої інфраструктури. Дані про побудовані маршрути та обрані користувачем точки зберігаються у routesHistory і favoritePlaces відповідно. Завдяки realtime-механізмам Firestore зміни в зазначених колекціях автоматично синхронізуються між усіма активними користувачами системи без необхідності повторного завантаження сторінки.

Архітектура обробки та синхронізації даних у системі наведена на рис. 16.

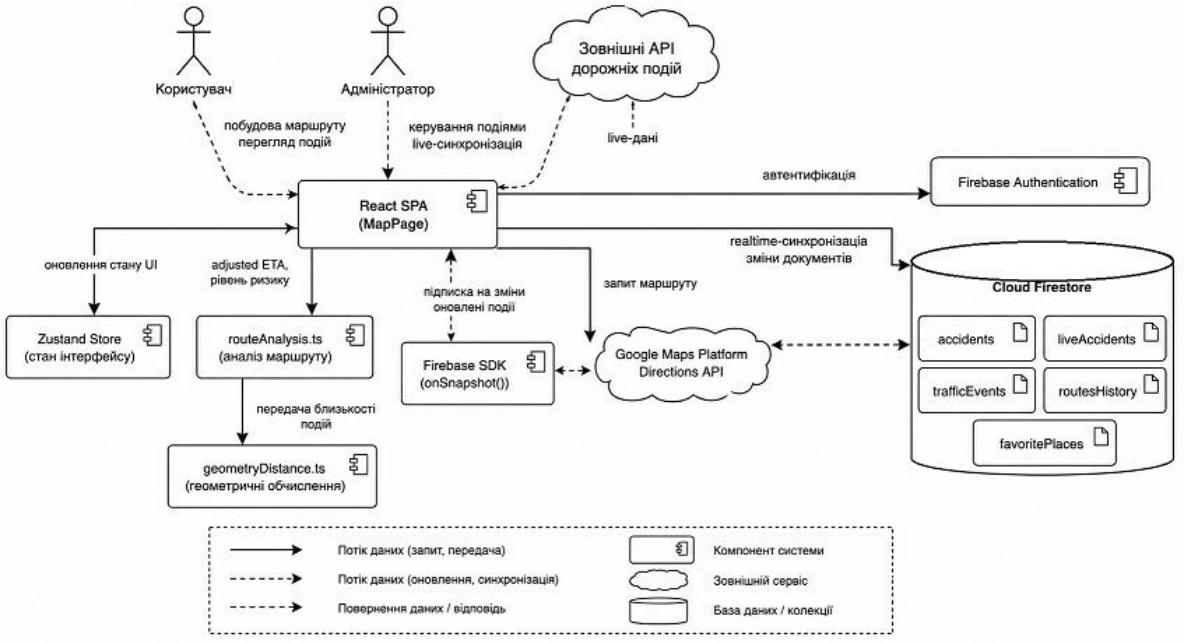

Рис. 16 Архітектура обробки та синхронізації даних у системі RoadGuard City

У структурі взаємодії компонентів основним джерелом даних виступають Firebase Firestore та зовнішні API дорожнього моніторингу. Клієнтська частина застосунку отримує дані через Firebase SDK та Google Maps Directions API, після чого виконується локальна обробка інформації у функціональних модулях routeAnalysis.ts і geometryDistance.ts. Отримані результати використовуються для візуалізації подій на інтерактивній карті, аналізу небезпечності маршруту та формування оновленого ETA.

Для синхронізації інформації про дорожні інциденти у системі реалізовано механізм інтеграції із зовнішніми API. Після активації синхронізації через адміністративну панель застосунок періодично виконує асинхронні HTTP-запити до зовнішнього сервісу дорожнього моніторингу. Отримані дані проходять декілька етапів обробки: перевірку структури відповіді, нормалізацію координат, усунення дублікатів, визначення типу події та приведення часових параметрів до єдиного формату. Після завершення обробки записи автоматично зберігаються у колекції liveAccidents. Такий підхід дозволяє підтримувати актуальність інформації без ручного введення дорожніх подій адміністратором.

Важливою складовою роботи системи є повторний аналіз маршрутів після надходження нових дорожніх подій. Після отримання маршруту через Google Maps Directions API модуль routeAnalysis.ts виконує аналіз координат маршруту та визначає відстань між сегментами шляху і дорожніми подіями. Для цього використовується допоміжний модуль geometryDistance.ts, який реалізує обчислення геометричної близькості між координатами маршруту та точками дорожніх інцидентів. Якщо відстань до події не перевищує встановленого порогового значення, система вважає, що подія впливає на маршрут користувача.

Після визначення критичних точок маршруту система виконує оцінювання рівня небезпечності дорожньої ситуації. Для кожної події визначається коефіцієнт впливу залежно від типу інциденту, рівня затору або ступеня перекриття руху. На основі отриманих коефіцієнтів формується скоригований показник ETA, який враховує додаткові затримки пересування. Таким чином,

користувач отримує не лише найкоротший маршрут, але й маршрут із урахуванням поточної дорожньої ситуації та потенційних ризиків пересування.

Основні механізми синхронізації та обробки даних наведено у табл. 2.4.

Таблиця 2.4

Основні механізми синхронізації даних у системі RoadGuard City

Механізм	Призначення
onSnapshot()	Автоматичне отримання оновлень із Firestore
Firebase Firestore	Зберігання та синхронізація даних у realtime-режимі
Google Maps Directions API	Отримання маршрутів та координат пересування
routeAnalysis.ts	Аналіз впливу дорожніх подій на маршрут
geometryDistance.ts	Обчислення відстаней між координатами
Firebase Authentication	Контроль автентифікації та ролей користувачів
Zustand Store	Локальна синхронізація стану інтерфейсу

Окрему увагу у системі приділено забезпеченню цілісності та безпеки даних. Для реалізації контролю доступу використовуються Firebase Security Rules, які визначають права користувачів на читання, створення, редагування та видалення документів. Незареєстровані користувачі мають доступ лише до перегляду дорожніх подій та побудови маршрутів. Авторизовані користувачі можуть додатково працювати з власними колекціями favoritePlaces і routesHistory. Адміністратори системи отримують розширені права на керування дорожніми подіями, виконання live-синхронізації та редагування інформації про аварії [14].

У процесі обробки даних широко використовуються асинхронні механізми JavaScript та TypeScript. Отримання маршрутів, обробка realtime-подій і синхронізація з Firestore виконуються через Promise-based запити та асинхронні функції async/await. Такий підхід дозволяє уникнути блокування користувацького інтерфейсу під час виконання мережевих операцій, забезпечує стабільну роботу інтерактивної карти та підтримує коректне оновлення компонентів React у режимі реального часу [15].

Застосування Firestore, Firebase SDK та асинхронної моделі взаємодії компонентів дозволило реалізувати у системі RoadGuard City ефективний

механізм обробки й синхронізації даних, який забезпечує актуальність дорожньої інформації, швидке оновлення маршрутів, узгодженість даних між клієнтами та стабільну роботу веб-застосунку при одночасній взаємодії великої кількості користувачів.

2.4 Модель даних та організація збереження інформації системи

Модель даних системи RoadGuard City сформована відповідно до особливостей веб-орієнтованого застосунку моніторингу дорожньої ситуації та аналізу маршрутів у режимі реального часу. Під час проєктування структури збереження даних основна увага приділялась забезпеченню швидкої обробки геопросторової інформації, підтримці realtime-синхронізації та мінімізації затримок під час взаємодії клієнтської частини із хмарною базою даних. Для реалізації системи використано документно-орієнтовану базу даних Cloud Firestore платформи Firebase, що забезпечує масштабоване зберігання інформації та автоматичне оновлення даних між клієнтами [12], [18].

Структура даних організована у вигляді окремих колекцій, кожна з яких відповідає певній сутності предметної області. Основними сутностями системи є користувач, дорожньо-транспортна пригода, дорожня подія, історія маршрутів та обрані місця користувача. Сутність User використовується для збереження інформації про зареєстрованих користувачів системи та містить унікальний ідентифікатор, електронну пошту, роль і дату створення облікового запису. Рольова модель дозволяє розмежувати доступ до функціональних можливостей системи між користувачем та адміністратором.

Сутність Accident призначена для представлення дорожньо-транспортних пригод. Для кожного запису зберігаються координати події, опис ситуації, рівень небезпечності, часові параметри та джерело отримання інформації. Дані використовуються під час аналізу маршруту та оцінювання ризику дорожньої ситуації.

Для представлення дорожніх подій використовується сутність TrafficEvent. Вона містить тип події, координати, геометрію об’єкта, часові межі дії та рівень ризику. Події можуть бути представлені у вигляді точкових, лінійних або полігональних об’єктів, що забезпечує підтримку просторового аналізу та інтеграції із картографічними сервісами [7], [15].

Історія побудованих маршрутів зберігається у сутності RouteHistory. У ній фіксуються початкова та кінцева точки маршруту, тип транспорту, тривалість маршруту, adjusted ETA та дата створення запису. Додатково зберігається ідентифікатор користувача, що дозволяє формувати персональну історію маршрутів.

Сутність FavoritePlace використовується для збереження обраних користувачем місць. Вона містить назву точки, координати, тип місця та посилання на користувача-власника.

Особливістю моделі даних є використання денормалізованого підходу до збереження інформації. Оскільки Firestore належить до документно-орієнтованих баз даних, частина інформації дублюється у документах для зменшення кількості запитів та прискорення отримання даних у режимі реального часу [12]. Взаємозв’язки між сутностями реалізуються через ідентифікатори користувачів, часові мітки та посилання на документи колекцій.

Під час функціонування системи модель даних активно використовується модулем аналізу маршрутів. Інформація про аварії та дорожні події поєднується з даними маршруту користувача для визначення рівня ризику та розрахунку скоригованого часу прибуття. Оновлення даних виконується через механізми realtime-синхронізації Firebase SDK, що забезпечує автоматичне оновлення інформації в інтерфейсі користувача [18].

Таким чином, модель даних системи RoadGuard City забезпечує ефективне зберігання та обробку інформації, підтримує роботу із геопросторовими об’єктами та створює основу для функціонування механізмів аналізу дорожньої ситуації у режимі реального часу.

3 РОЗРОБКА ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ МОНІТОРИНГУ

3.1 Вибір технологій та засобів програмної реалізації

Під час розробки системи моніторингу дорожнього руху RoadGuard City основна увага приділялася швидкодії, підтримці роботи у режимі реального часу, масштабованості та ефективній роботі з геопросторовими даними. Для реалізації вебзастосунку було використано сучасний технологічний стек, який забезпечує інтеграцію із картографічними сервісами, підтримку realtime-синхронізації та зручну організацію клієнтської частини.

Клієнтська частина системи реалізована за допомогою React та TypeScript. Використання React дозволило побудувати застосунок за компонентним принципом, що спрощує підтримку та розширення функціоналу системи. TypeScript забезпечує статичну типізацію, підвищує надійність програмного коду та дозволяє зменшити кількість помилок під час роботи із маршрутами, координатами та API-запитами [13].

Для маршрутизації між сторінками використовується React Router, а для централізованого керування станом застосунку – бібліотека Zustand. Її використання дозволило організувати збереження даних про маршрути, дорожні події та параметри інтерфейсу без складної конфігурації.

Однією з основних технологій системи є Google Maps Platform. У проєкті використовуються Google Maps JavaScript API, Directions API, Places API та Geometry Library. Дані сервіси забезпечують відображення інтерактивної карти, побудову маршрутів, пошук геолокацій та аналіз дорожніх подій [14].

Для авторизації користувачів та збереження даних використовується Firebase. У системі застосовуються Firebase Authentication та Cloud Firestore. Firebase Authentication забезпечує автентифікацію користувачів та підтримку ролей user і admin. Cloud Firestore використовується як основна NoSQL база даних для зберігання інформації про аварії, маршрути та дорожні події.

Realtime-синхронізація даних реалізована за допомогою механізмів onSnapshot(), що дозволяє автоматично оновлювати інформацію про дорожню ситуацію без перезавантаження сторінки. Для виконання HTTP-запитів до зовнішніх API використовується fetch().

Модулі аналізу маршрутів routeAnalysis.ts, geometryDistance.ts та directions.ts забезпечують аналіз дорожніх подій, перевірку перетину маршруту із небезпечними зонами та розрахунок adjusted ETA.

Розроблення програмного забезпечення здійснювалося у середовищі Visual Studio Code із використанням системи контролю версій Git та платформи GitHub [15].

Основні технології, використані у системі RoadGuard City, наведені у табл. 3.1.

Таблиця 3.1

Технології програмної реалізації системи RoadGuard City

Технологія	Призначення
React	Побудова клієнтського SPA-інтерфейсу
TypeScript	Типізація програмного коду
React Router	Маршрутизація сторінок
Zustand	Керування станом застосунку
Firebase Authentication	Авторизація користувачів
Cloud Firestore	Зберігання та синхронізація даних
Google Maps API	Робота з інтерактивною картою
Directions API	Побудова маршрутів
Places API	Пошук геолокацій
Geometry Library	Аналіз геометрії маршрутів
fetch()	Виконання HTTP-запитів
Git / GitHub	Контроль версій
Visual Studio Code	Середовище розроблення

Отже, використаний набір технологій забезпечує підтримку роботи системи RoadGuard City у режимі реального часу, інтеграцію із зовнішніми картографічними сервісами та ефективну реалізацію функцій аналізу дорожньої ситуації.

3.2 Архітектура програмної системи

Архітектура системи RoadGuard City побудована за модульним принципом із розподілом функціональності між клієнтською частиною, підсистемою аналітики маршрутів, модулем доступу до даних та зовнішніми сервісами. Такий підхід забезпечує масштабованість застосунку, спрощує підтримку програмного коду та дозволяє ізолювати окремі функціональні компоненти системи. Загальну структуру взаємодії підсистем наведено на рисунку 17.

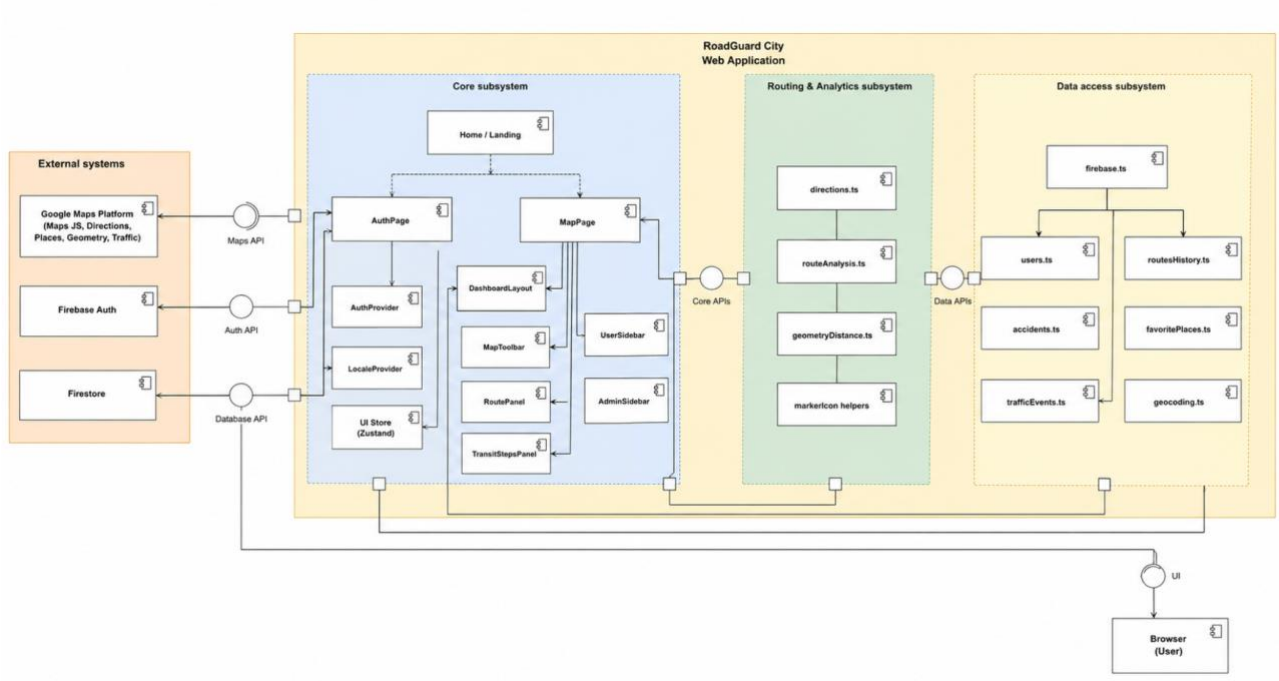

Рис. 17 Діаграма компонентів проєкту

Як показано на рисунку 17, центральною частиною системи є SPA-застосунок, реалізований на основі React та TypeScript. Клієнтська частина забезпечує взаємодію користувача із системою, побудову маршрутів, відображення дорожніх подій та роботу з інтерактивною картою. Основною сторінкою системи є MapPage, у межах якої реалізовано побудову маршрутів, аналіз дорожньої ситуації та взаємодію з адміністративними модулями.

Для організації інтерфейсу використовується компонентний підхід. Основні елементи інтерфейсу представлені компонентами DashboardLayout, MapSidebar, RoutePanel та TransitStepsPanel. Компонент DashboardLayout

забезпечує загальну структуру сторінки, а MapSidebar використовується для введення параметрів маршруту та роботи із дорожніми подіями. Компонент RoutePanel відповідає за відображення альтернативних маршрутів та їх характеристик, зокрема часу пересування та рівня впливу дорожніх подій.

Логіка авторизації реалізована через Firebase Authentication. Для централізованого керування станом застосунку використовується Zustand, що дозволяє зберігати параметри інтерфейсу, інформацію про маршрути та повідомлення системи без надмірного ускладнення архітектури [13].

Важливу роль у системі відіграє підсистема аналітики маршрутів, структура якої наведена на рисунку 18.

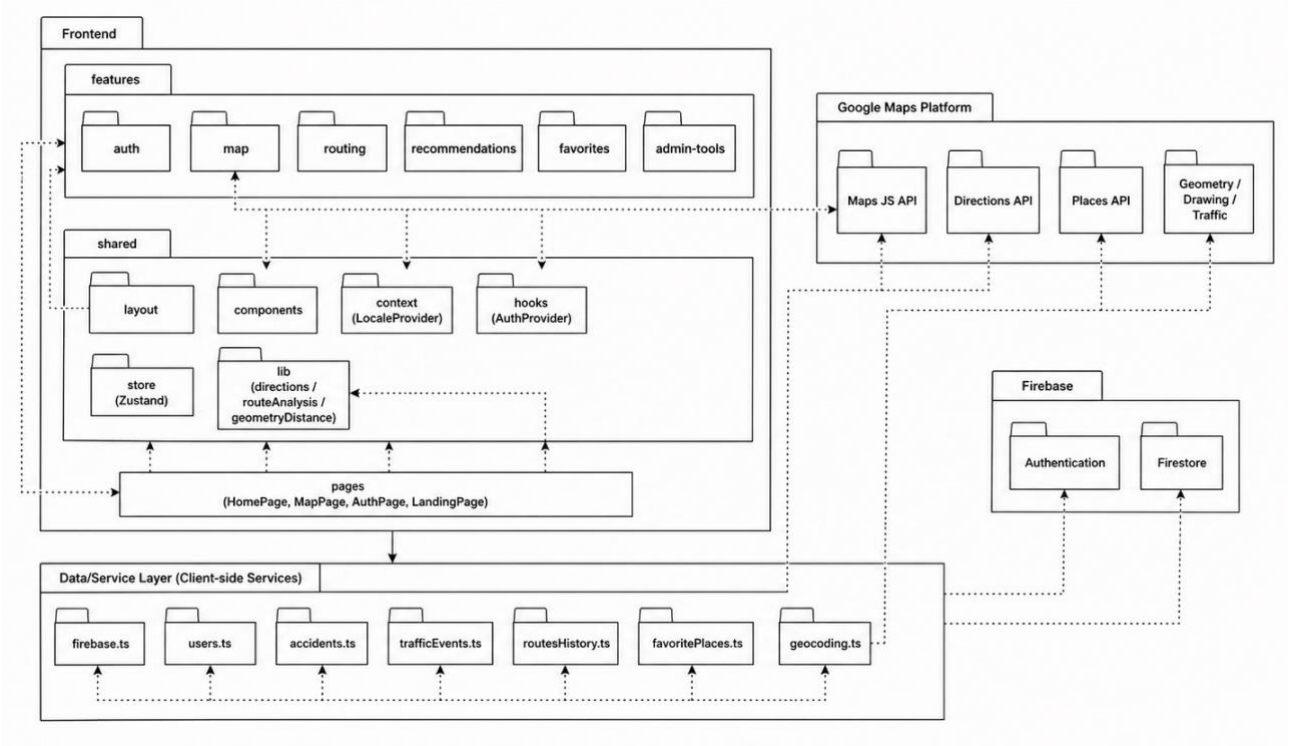

Рис. 18 Архітектура програмної системи RoadGuard City у вигляді діаграми пакетів проєкту

Підсистема аналітики реалізована у модулях directions.ts, routeAnalysis.ts та geometryDistance.ts. Модуль directions.ts забезпечує взаємодію з Google Maps Directions API та отримання альтернативних маршрутів між заданими точками. Модуль routeAnalysis.ts виконує аналіз маршрутів, визначає вплив аварій та дорожніх подій, а також розраховує adjusted ETA з урахуванням поточної

дорожньої ситуації. Для виконання геометричних обчислень використовується geometryDistance.ts, який визначає відстані між маршрутом та дорожніми подіями [14].

Концептуальна модель взаємодії основних сутностей системи наведена на рисунку 19.

Рис. 19 Концептуальна модель взаємодії основних сутностей системи

У системі основними сутностями є користувач, маршрут, аварія та система моніторингу. Користувач формує маршрут та взаємодіє із системою через вебінтерфейс. Маршрут містить інформацію про початкову та кінцеву точки, базовий час пересування та ETA. Дані про аварії використовуються підсистемою аналітики для оцінювання рівня небезпечності маршруту та формування рекомендацій щодо пересування.

Для деталізації взаємозв’язків між сутностями використовується UML-діаграма класів, наведена на рисунку 20.

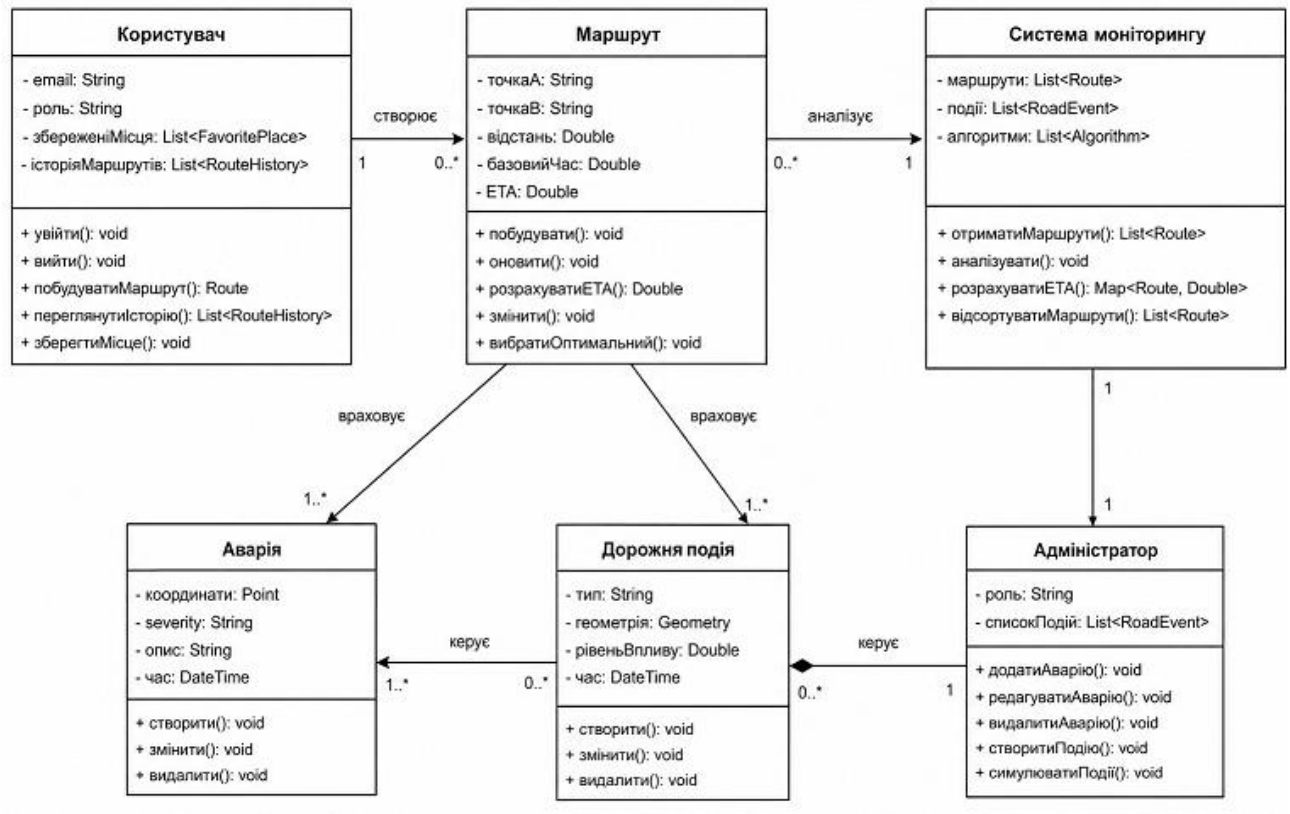

Рис. 20 Діаграма класів системи RoadGuard City

Як показано на рисунку 20, система підтримує взаємодію між користувачами, маршрутами, дорожніми подіями та адміністративним модулем. Адміністратор має розширені права доступу та може створювати, редагувати або видаляти дорожні події. Маршрути аналізуються системою моніторингу з урахуванням аварій та дорожніх інцидентів.

Для зберігання даних використовується Cloud Firestore. Робота з базою даних реалізована через окремі модулі users.ts, accidents.ts, trafficEvents.ts, routesHistory.ts та favoritePlaces.ts. Такий підхід дозволяє ізолювати доступ до даних від інших частин системи та спрощує підтримку програмного коду. Realtime-синхронізація інформації реалізована за допомогою механізму onSnapshot(), що дозволяє автоматично оновлювати дані про дорожню ситуацію без перезавантаження сторінки.

Інтеграція із Google Maps Platform забезпечує роботу з інтерактивною картою, геокодуванням адрес, побудовою маршрутів та аналізом дорожньої ситуації. У проєкті використовуються Google Maps JavaScript API, Directions API,

Places API та Geometry Library. Firebase забезпечує автентифікацію користувачів, керування ролями та підтримку роботи із хмарною базою даних.

Таким чином, архітектура системи RoadGuard City побудована із використанням модульного підходу та сучасних вебтехнологій, що забезпечує підтримку роботи у режимі реального часу, ефективну взаємодію із зовнішніми сервісами та можливість подальшого розвитку програмного забезпечення.

3.3 Алгоритми взаємодії компонентів та організація потоків даних у системі RoadGuard City

Функціонування веб-застосунку RoadGuard City базується на постійній взаємодії між клієнтською частиною, сервісами Firebase, зовнішніми API Google Maps Platform та внутрішніми модулями аналізу дорожньої ситуації. Побудова алгоритмів взаємодії компонентів здійснювалася з урахуванням вимог до роботи системи у режимі реального часу, мінімізації затримок під час обробки маршрутів, підтримки адаптивної логіки аналізу дорожніх подій та забезпечення стабільної синхронізації даних між користувачами.

Одним із центральних сценаріїв роботи системи є процес авторизації та автентифікації користувачів. Після відкриття сторінки застосунку користувач переходить до форми входу або реєстрації, де взаємодія здійснюється через Firebase Authentication. У випадку авторизації модуль AuthProvider передає введені користувачем облікові дані до Firebase SDK із використанням методу signInWithEmailAndPassword(). Після успішного входу система автоматично отримує JWT-токен користувача, а також виконує перевірку наявності профілю у колекції users бази Cloud Firestore. На основі отриманої ролі користувача активуються відповідні елементи інтерфейсу та дозволені функції системи. Для користувачів із роллю administrator додатково відкривається доступ до адміністративних компонентів керування дорожніми подіями та live-синхронізацією інцидентів. Алгоритм авторизації представлено на рис. 21.

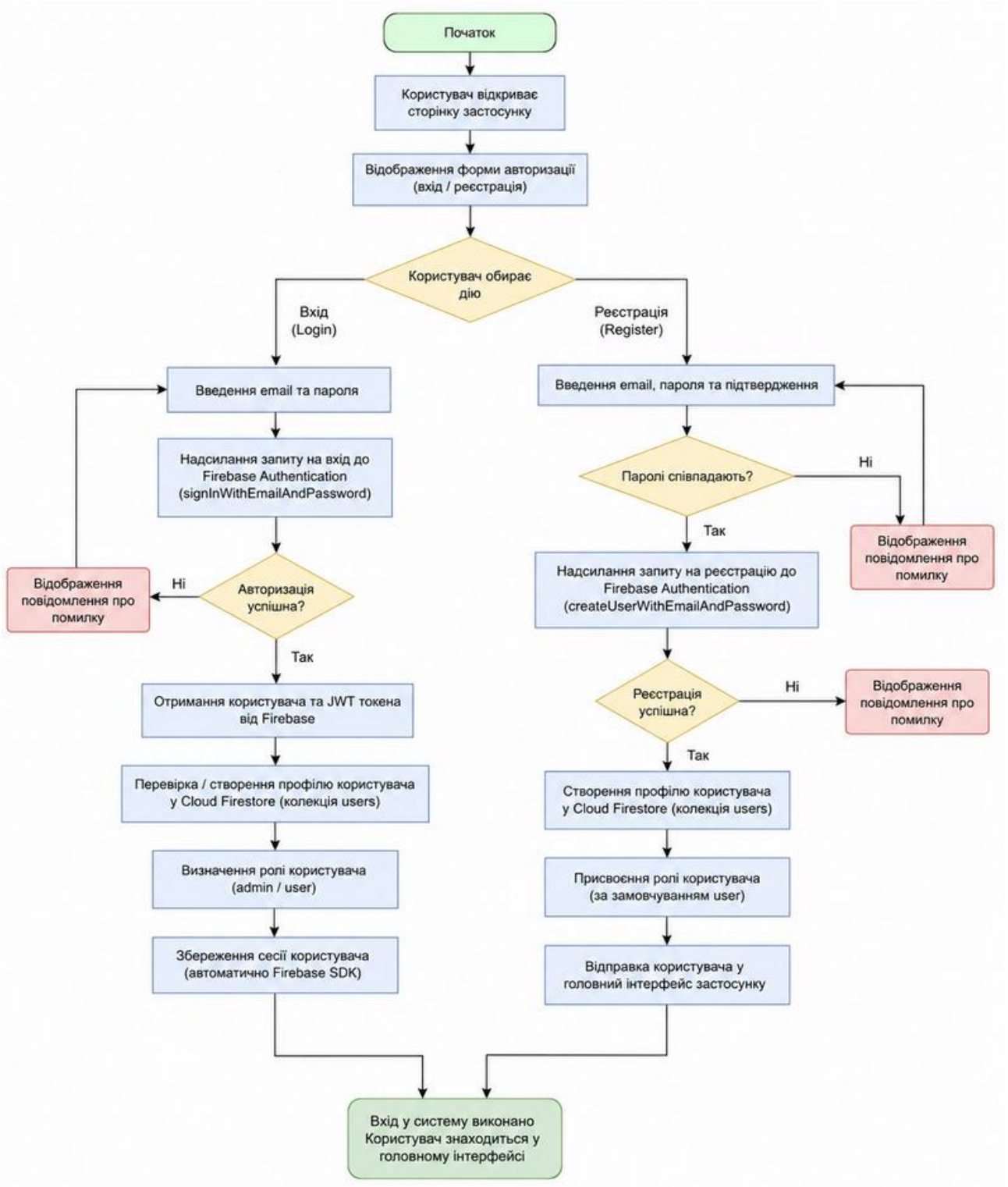

Рис. 21 Алгоритм процесу авторизації користувача в системі RoadGuard City

Після завершення авторизації основним середовищем взаємодії стає сторінка карти MapPage, яка забезпечує роботу з маршрутами, дорожніми подіями та історією переміщень користувача. У процесі побудови маршруту користувач вводить початкову та кінцеву точки через компонент

PlaceAutocompleteInput, після чого клієнтська частина формує запит до Google Places API для геокодування адрес. У випадку успішного перетворення текстових адрес у координати система передає параметри маршруту до Google Directions API через модуль directions.ts.

У відповідь Google Directions API повертає набір альтернативних маршрутів, які містять полігон маршруту, орієнтовний час пересування, дистанцію, проміжні точки та навігаційні кроки. Отримані результати надходять до модуля routeAnalysis.ts, де виконується додатковий аналіз безпечності маршруту та оцінка дорожньої ситуації. На цьому етапі система одночасно звертається до колекцій accidents, liveAccidents та trafficEvents у Firestore через сервіси accidents.ts, liveAccidents.ts і trafficEvents.ts.

Подальша обробка виконується модулем geometryDistance.ts, який аналізує просторову близькість дорожніх подій до траєкторії маршруту. Для кожного маршруту здійснюється перевірка перетину з аваріями, перекриттями руху, ремонтними роботами або іншими небезпечними ділянками дороги. Якщо відстань між координатами маршруту та дорожньою подією не перевищує визначений поріг, система застосовує додатковий коефіцієнт впливу на маршрут. У результаті формується скоригований показник adjusted ETA, який використовується для ранжування альтернативних маршрутів за рівнем безпечності та прогнозованим часом пересування.

Особливістю реалізації є використання алгоритмів асинхронної обробки даних. Оскільки запити до Google Maps Platform та Firestore виконуються незалежно один від одного, система використовує механізми Promise-based обробки та реактивного оновлення стану інтерфейсу. Це дозволяє мінімізувати затримки під час побудови маршрутів навіть при значній кількості дорожніх подій.

Алгоритм побудови та аналізу маршруту представлено на рис. 22.

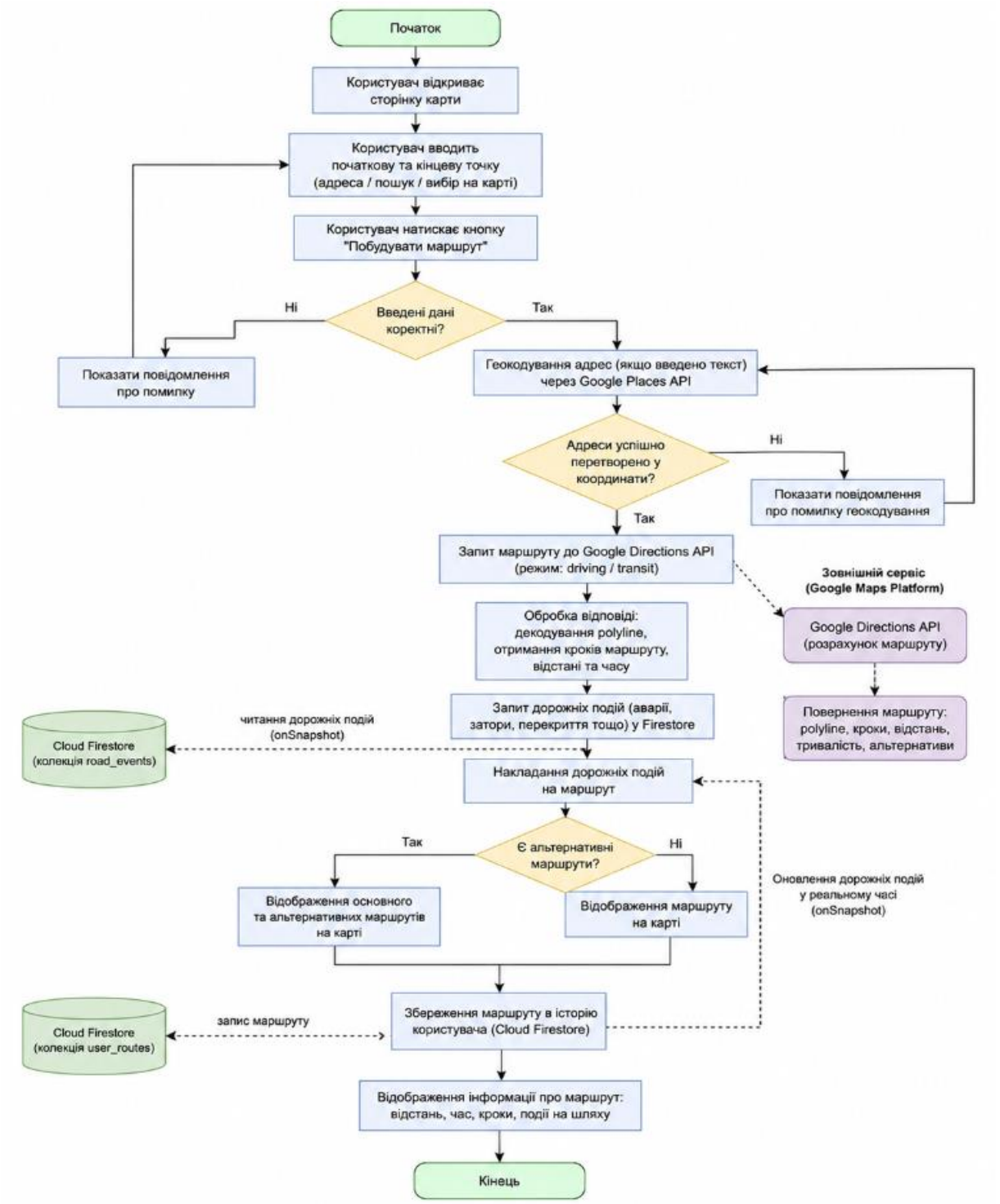

Рис. 22 Алгоритм побудови та аналізу маршруту в системі RoadGuard City

При додаванні нової аварії або зміні статусу події карта миттєво оновлює візуальні елементи та повторно виконує аналіз маршрутів. Live-синхронізація реалізована через модуль AdminSidebar та сервіс liveAccidents.ts, який отримує дані із зовнішнього API, виконує їх нормалізацію та зберігає у колекції

liveAccidents бази Firestore. Завдяки realtime-підписці нові дорожні події одразу відображаються у всіх активних користувачів.

Для збереження історії маршрутів використовується сервіс routesHistory.ts, через який у Firestore записуються параметри маршруту, тип транспорту, значення ETA та інформація про дорожні події. Аналогічний механізм застосовується для роботи з обраними місцями через сервіс favoritePlaces.ts. Важливою складовою системи є також механізм обробки помилок. У випадку помилок під час роботи з Google Maps API, Firebase Authentication або Firestore система використовує централізоване перехоплення винятків із передачею повідомлень користувачу через компоненти Toasts та AlertMessage. Додатково реалізовано перевірку коректності координат, валідності маршрутів та доступності зовнішніх сервісів.

Організація потоків даних у RoadGuard City побудована за принципом реактивної взаємодії між компонентами, що забезпечує узгоджену роботу клієнтської частини, хмарної бази даних та зовнішніх геоінформаційних сервісів. Такий підхід дозволяє підтримувати актуальність дорожньої інформації у режимі реального часу, оперативно реагувати на зміни дорожньої ситуації та забезпечувати стабільну роботу системи навіть при високій інтенсивності оновлення даних.

3.4 Фізична архітектура та інфраструктура розгортання системи

Фізична архітектура веб-застосунку RoadGuard City побудована за принципами serverless-архітектури із використанням Firebase та Google Maps Platform. Такий підхід дозволив реалізувати масштабовану систему моніторингу дорожньої ситуації без необхідності розгортання окремого серверного програмного забезпечення. Взаємодія між компонентами здійснюється через клієнтський React-застосунок, який працює із зовнішніми API та хмарними сервісами через HTTPS-з’єднання.

Архітектурно система складається із трьох основних рівнів: клієнтської частини, хмарної інфраструктури Firebase та зовнішніх сервісів Google Maps Platform. Клієнтська частина реалізована як SPA-застосунок на основі React і TypeScript. Для маршрутизації між сторінками використовується React Router, а інтерактивне відображення карти та дорожніх подій забезпечується Maps JavaScript API.

Інтеграція із Google Maps Platform здійснюється через Maps JavaScript API, Directions API, Places API та Geometry Library. Directions API використовується для побудови маршрутів та визначення часу пересування, Places API – для автодоповнення адрес і геокодування, а Geometry Library – для просторового аналізу маршрутів та дорожніх подій.

Для зберігання інформації використовується Firebase Cloud Firestore – документоорієнтована NoSQL база даних із підтримкою realtime-синхронізації. У Firestore зберігаються користувачі, маршрути, аварії, дорожні події та обрані місця. Авторизація користувачів реалізована через Firebase Authentication із підтримкою входу за email і паролем та автоматичного відновлення сесії.

Однією з ключових особливостей системи є realtime-синхронізація через onSnapshot(), що дозволяє автоматично оновлювати інформацію про дорожню ситуацію без перезавантаження сторінки. Для отримання live-даних використовується модуль liveAccidents.ts, який отримує JSON-дані із зовнішніх API та зберігає їх у Firestore. Доступ до функціональних можливостей системи реалізований через role-based access control із ролями user та admin. Для забезпечення безпеки взаємодія між компонентами виконується через HTTPS, а доступ до даних контролюється за допомогою Firebase Security Rules.

Розгортання клієнтської частини застосунку здійснюється через Firebase Hosting, який забезпечує підтримку HTTPS, CDN-кешування та стабільну доставку ресурсів користувачам.

Таким чином, фізична архітектура RoadGuard City забезпечує масштабованість системи, підтримку роботи у режимі реального часу та ефективну інтеграцію із зовнішніми сервісами дорожнього моніторингу.

3.5 Інтеграція із зовнішніми сервісами

Для забезпечення функціонування системи RoadGuard City у застосунку реалізовано інтеграцію із зовнішніми сервісами, що відповідають за роботу картографічного модуля, побудову маршрутів, авторизацію користувачів, зберігання даних та синхронізацію дорожньої інформації у режимі реального часу. Використання cloud-based платформ дозволило спростити серверну архітектуру та зосередити процес розробки на прикладній логіці системи [18, 24].

Основним зовнішнім компонентом є Google Maps Platform, яка використовується для відображення інтерактивної карти, побудови маршрутів та роботи з геоданими. Інтеграція реалізована через Maps JavaScript API, Directions API, Places API та Geometry Library. Maps JavaScript API забезпечує візуалізацію карти, дорожніх подій та маршрутів у браузері користувача. Directions API використовується для побудови альтернативних маршрутів і визначення часу пересування. Places API реалізує пошук та автодоповнення адрес, а Geometry Library використовується для просторових обчислень, зокрема аналізу близькості дорожніх подій до маршруту.

Для реалізації серверної інфраструктури використовується Firebase. Авторизація користувачів реалізована через Firebase Authentication із підтримкою реєстрації, входу в систему та автоматичного відновлення сесії. У застосунку передбачено ролі user та admin для розмежування доступу до функціональних можливостей системи.

Зберігання інформації виконується у Cloud Firestore – NoSQL базі даних із підтримкою realtime-синхронізації. У базі зберігаються маршрути, дорожні події, історія навігації та дані користувачів. Завдяки realtime listeners інтерфейс автоматично оновлюється після зміни інформації у базі даних.

У системі також реалізовано механізм інтеграції із зовнішніми джерелами live-даних про дорожні інциденти. Для цього використовується модуль liveAccidents.ts, який отримує JSON-дані із зовнішнього API, виконує їх нормалізацію та автоматично зберігає у Firestore. Такий підхід забезпечує

сумісність із різними сервісами дорожнього моніторингу та спрощує масштабування системи.

Передача даних між клієнтським застосунком, Firebase та зовнішніми API здійснюється через HTTPS із використанням TLS-шифрування, що забезпечує захист інформації під час взаємодії компонентів системи. Для розміщення клієнтської частини застосунку використовується Firebase Hosting, який забезпечує підтримку HTTPS, кешування ресурсів та стабільну роботу SPA-застосунку.

Крім цього, у системі реалізовано механізм збереження маршрутів користувачів із використанням Firestore та realtime-синхронізації. Після побудови маршруту система перевіряє стан авторизації користувача, формує структуру маршруту та створює документ у колекції routesHistory. Алгоритм взаємодії користувача із системою під час збереження маршруту наведено на рисунку 23.

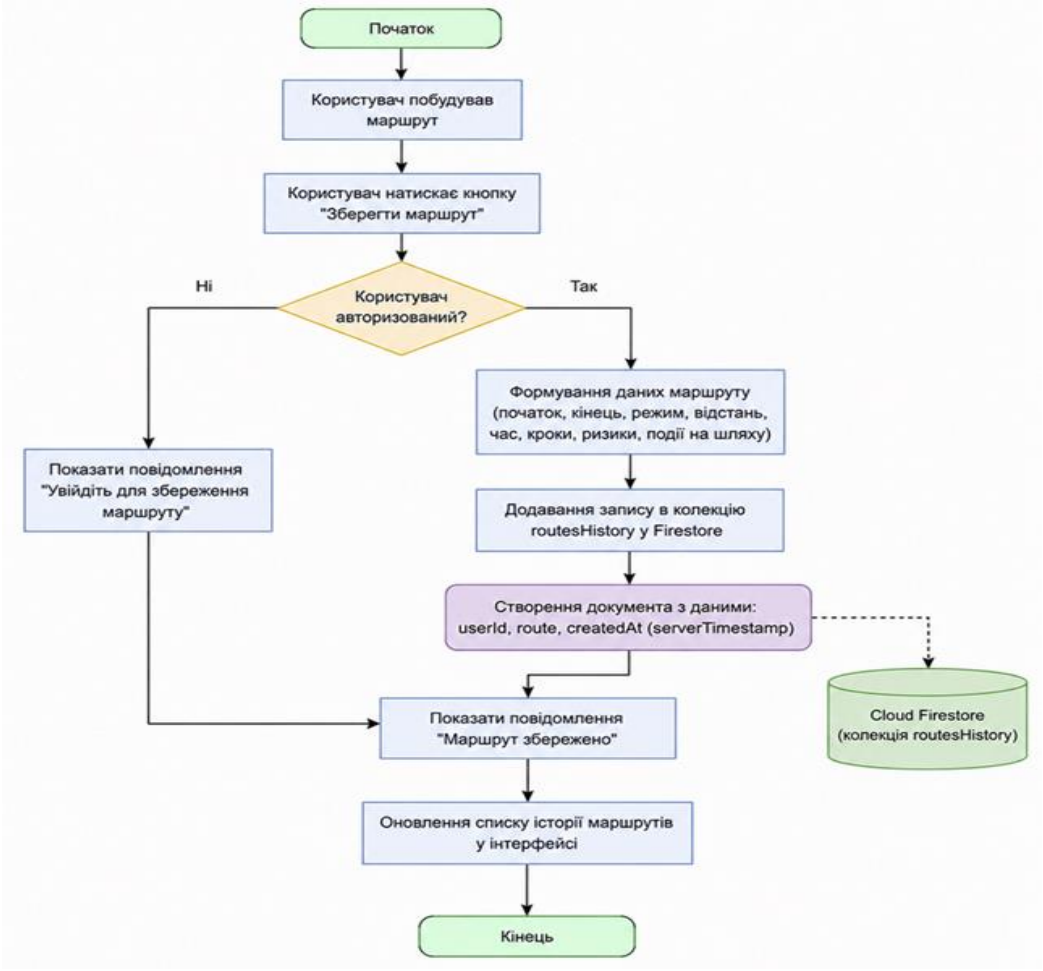

Рис. 23 Алгоритм збереження маршруту користувача у Cloud Firestore

Таким чином, інтеграція із зовнішніми сервісами та хмарною інфраструктурою забезпечує реалізацію основних функціональних можливостей RoadGuard City, включаючи інтерактивну навігацію, побудову маршрутів, realtime-оновлення дорожніх подій, хмарне зберігання інформації та захищену взаємодію між компонентами системи. Використання Google Maps Platform та Firebase дозволило реалізувати масштабовану web-oriented архітектуру із підтримкою сучасних механізмів синхронізації та обробки геоданих.

4 ЕКСПЛУАТАЦІЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Підготовка середовища виконання

Перед початком запуску та тестування системи RoadGuard City необхідно підготувати середовище виконання, яке забезпечує стабільну роботу клієнтської частини, інтеграцію із зовнішніми API та коректну взаємодію із сервісами Firebase і Google Maps Platform. Основу середовища складають Node.js, менеджер пакетів npm, середовище збірки Vite та хмарна інфраструктура Firebase.

Клієнтська частина системи реалізована на основі React 18 і TypeScript 5, тому для запуску проєкту використовується Node.js версії не нижче 18.x. Використання актуальної LTS-версії Node.js забезпечує сумісність із сучасними бібліотеками та підтримку можливостей ECMAScript. Node.js використовується для запуску локального Vite Dev Server, збірки SPA-застосунку та керування npm-залежностями.

У якості менеджера пакетів використовується npm, через який встановлюються всі бібліотеки та модулі проєкту, зазначені у файлі package.json. До основних залежностей належать React, React Router, Firebase SDK, Zustand, TypeScript та @react-google-maps/api. Використання Vite дозволяє пришвидшити запуск локального сервера та забезпечує підтримку hot reload під час розробки інтерфейсу.

На відміну від класичних багаторівневих клієнт-серверних систем, у RoadGuard City не використовується окремий backend-сервер або реляційна СУБД. Архітектура системи побудована за принципом Backend-as-a-Service із використанням Firebase. Для зберігання даних використовується Cloud Firestore, а для реалізації авторизації – Firebase Authentication. Такий підхід дозволяє спростити процес розгортання системи та зменшити складність серверної інфраструктури.

Для коректної роботи застосунку необхідно підключити сервіси Google Maps Platform, зокрема Maps JavaScript API, Directions API, Places API та Geometry Library. Дані сервіси забезпечують відображення інтерактивної карти, побудову маршрутів, автодоповнення адрес та аналіз дорожніх подій.

Запуск системи виконується локально через Vite Dev Server за допомогою команд npm install та npm run dev. Після запуску застосунок автоматично ініціалізує підключення до Firebase та Google Maps Platform і відкривається у браузері через localhost.

Для зберігання параметрів підключення використовується файл .env, у якому задаються ключі доступу до Firebase та Google Maps Platform. У файлі також можуть бути визначені змінні для інтеграції із зовнішніми джерелами live-даних про дорожні інциденти, зокрема VITE_LIVE_ACCIDENTS_FEED_URL та VITE_LIVE_ACCIDENTS_FEED_TOKEN.

Завдяки використанню serverless-архітектури та хмарних сервісів Firebase процес підготовки середовища виконання значно спрощується, оскільки система не потребує окремого налаштування серверного програмного забезпечення, бази даних або мережевої інфраструктури. Такий підхід забезпечує швидке розгортання застосунку, підтримку масштабованості та спрощує подальше супроводження програмного забезпечення.

4.2 Розгортання системи

Для наочного представлення фізичного розміщення компонентів системи RoadGuard City та принципів їх взаємодії було побудовано діаграму розгортання, наведену на рис. 24. Дана діаграма відображає структуру взаємодії між клієнтським SPA-застосунком, сервісами Firebase, платформою Google Maps Platform та зовнішніми джерелами дорожніх даних. Крім того, схема дозволяє продемонструвати канали передачі інформації, способи підключення до хмарної інфраструктури та організацію realtime-синхронізації у системі.

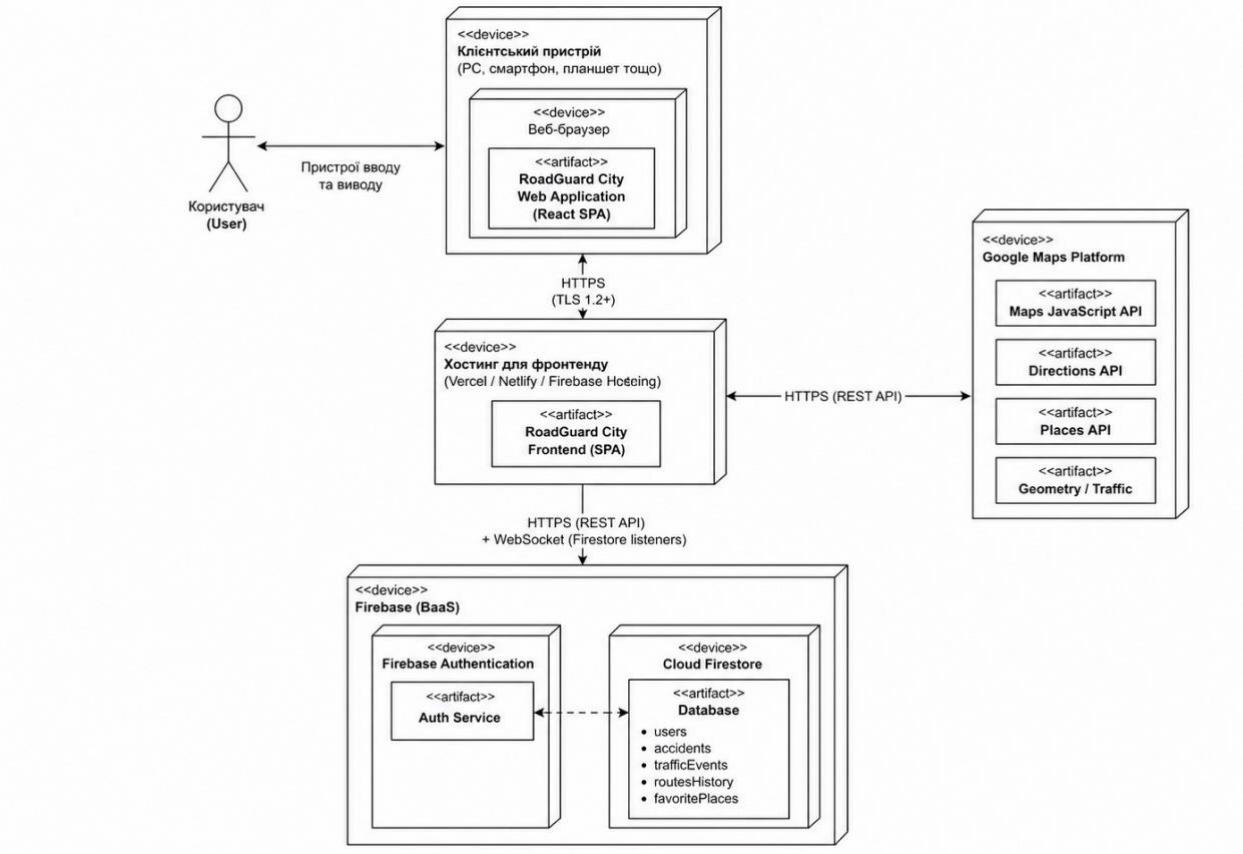

Рис. 24 Діаграма розгортання системи RoadGuard City

Як показано на рис. 24, основним компонентом системи є клієнтський веб-застосунок RoadGuard City Web Application, реалізований на основі React та TypeScript. Застосунок виконується у веб-браузері користувача та підтримує роботу на персональних комп’ютерах, ноутбуках, планшетах і смартфонах. Взаємодія із системою здійснюється через браузерний інтерфейс, у якому відображаються інтерактивна карта, маршрути, дорожні події та результати аналізу дорожньої ситуації.

Архітектура системи побудована за принципом serverless Backend-as-a-Service із використанням Firebase. У поточній реалізації не використовується окремий backend-сервер, а функції авторизації, зберігання даних та realtime-синхронізації реалізовані через Firebase Authentication і Cloud Firestore. Такий підхід дозволяє зменшити складність серверної інфраструктури та спростити процес розгортання системи.

Для роботи з геоданими та маршрутами система інтегрується із Google Maps Platform через HTTPS REST API. Maps JavaScript API використовується для відображення інтерактивної карти, Directions API – для побудови маршрутів, Places API – для автодоповнення адрес та геолокаційного пошуку, а Geometry Library – для просторового аналізу маршрутів і дорожніх подій.

Cloud Firestore використовується як основне хмарне сховище даних. У базі даних зберігаються колекції users, accidents, trafficEvents, routesHistory та favoritePlaces. Завдяки використанню realtime listeners через onSnapshot() система автоматично синхронізує зміни між усіма активними користувачами без необхідності ручного оновлення сторінки.

Для розміщення клієнтської частини застосунку можуть використовуватися Firebase Hosting, Vercel або Netlify. Дані сервіси забезпечують підтримку HTTPS, CDN-кешування та стабільну доставку ресурсів SPA-застосунку користувачам.

Для коректної роботи RoadGuard City необхідно дотримуватися мінімальних апаратних вимог, наведених у таблиці 4.1.

Таблиця 4.1

Мінімальні апаратні вимоги для експлуатації системи

Компонент	Мінімальні вимоги
Процесор	Intel Core i3 / AMD Ryzen 3 або вище
Оперативна пам’ять	4 ГБ
Вільне місце на диску	1 ГБ
Відеопідсистема	Підтримка WebGL та HTML5
Мережеве підключення	Інтернет-з’єднання від 10 Мбіт/с
Пристрій	ПК, ноутбук, планшет або смартфон

Наведені у табл. 4.1 характеристики є достатніми для стабільної роботи SPA-застосунку, відображення інтерактивної карти та виконання realtime-синхронізації дорожніх подій.

Для розгортання та запуску системи також необхідне відповідне програмне середовище. Основні програмні вимоги наведені у табл. 4.2.

Таблиця 4.2

Програмні вимоги середовища виконання

Програмний компонент	Версія / вимога
Операційна система	Windows 10/11, Linux або macOS
Node.js	18.x або вище
npm	9.x або вище
Веб-браузер	Google Chrome, Microsoft Edge, Mozilla Firefox
React	18+
TypeScript	5+
Vite	5+
Firebase SDK	10+

Дані, наведені у табл. 4.2, демонструють, що програмне середовище системи базується на сучасному web-oriented стеку технологій та підтримує роботу у більшості актуальних браузерів і операційних систем.

Для запуску системи необхідно встановити залежності проєкту через npm install та створити файл .env, у якому задаються параметри підключення до Firebase і Google Maps Platform. За потреби у файлі також можуть бути визначені параметри доступу до зовнішніх API дорожніх подій.

Після налаштування середовища запуск локального dev-сервера виконується командою npm run dev, після чого застосунок автоматично ініціалізує Firebase SDK, підключення до Firestore та Google Maps Platform. Після завершення розгортання необхідно перевірити роботу авторизації, побудови маршрутів, realtime-синхронізації та відображення дорожніх подій.

Таким чином, розгортання системи RoadGuard City реалізоване із використанням сучасної serverless-архітектури та хмарних сервісів Firebase, що забезпечує масштабованість, спрощене адміністрування та стабільну роботу застосунку без необхідності використання окремого backend-сервера.

4.3 Адміністрування та керування системою

Для забезпечення контролю дорожніх подій та підтримки актуальності інформації у системі RoadGuard City реалізовано механізм адміністративного

доступу. Адміністративний функціонал інтегрований безпосередньо у SPA-застосунок та активується залежно від ролі користувача після проходження авторизації. Такий підхід дозволяє централізовано керувати дорожніми інцидентами, виконувати тестування системи та підтримувати коректність даних у режимі реального часу.

У системі передбачено два рівні доступу – user та admin. Користувач із роллю user має доступ до функцій побудови маршрутів, перегляду дорожньої ситуації, роботи з історією маршрутів та обраними місцями. Роль admin надає розширені можливості керування дорожніми подіями, аваріями та механізмами live-синхронізації.

Після проходження авторизації система через Firebase Authentication та Cloud Firestore автоматично визначає роль користувача й активує відповідний функціонал інтерфейсу. Якщо користувач має роль адміністратора, у застосунку відкривається доступ до адміністративної панелі, реалізованої через компонент AdminSidebar та інтегрованої у сторінку MapPage.

Загальний вигляд адміністративної панелі наведено на рис. 25.

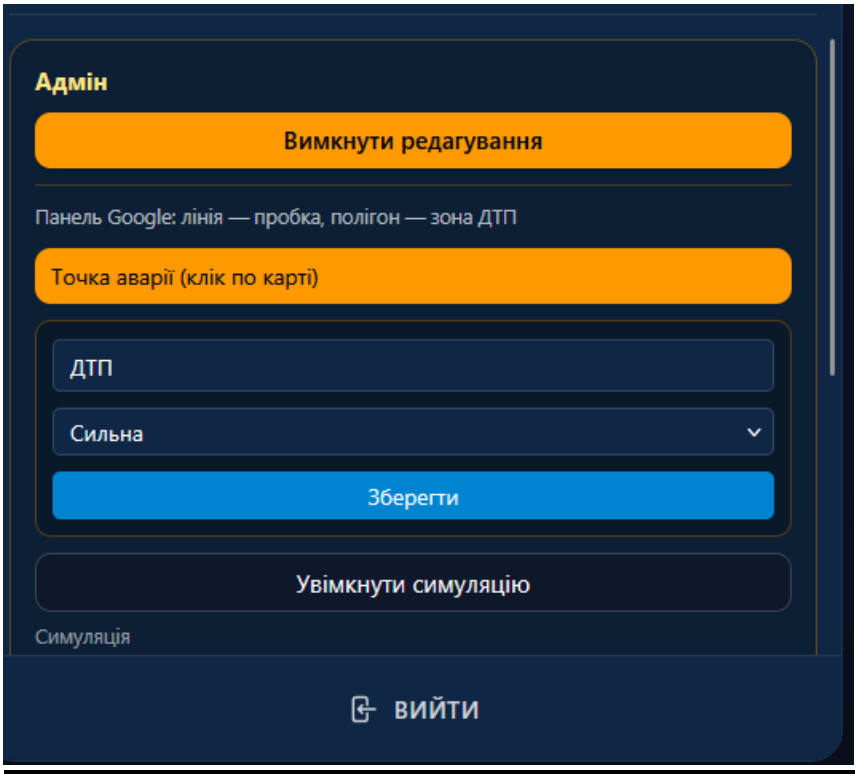

Рис.25 Панель адміністратора системи RoadGuard City

Як показано на рис. 25, адміністративна панель дозволяє керувати режимом редагування карти, створювати аварії, задавати рівень небезпечності дорожньої події та запускати режим симуляції. Інтерфейс адміністратора інтегрований безпосередньо у карту, що спрощує процес роботи із дорожніми інцидентами та дозволяє оперативно вносити зміни до системи.

Адміністратор може створювати нові аварії шляхом вибору точки на карті та введення параметрів події. Для кожного інциденту задаються координати, опис та рівень небезпечності. Після підтвердження введених даних інформація автоматично записується у Cloud Firestore та синхронізується між усіма активними користувачами системи через realtime listeners.

Окрему роль у процесі адміністрування відіграє механізм live-синхронізації дорожніх інцидентів. Через адміністративну панель адміністратор може активувати автоматичне отримання даних із зовнішніх JSON feed джерел. Після запуску синхронізації система через модуль liveAccidents.ts виконує HTTP-запити до зовнішнього API, обробляє отримані дані та автоматично зберігає їх у колекції liveAccidents бази Firestore. Завдяки цьому забезпечується підтримка актуальної інформації про дорожню ситуацію без необхідності ручного введення всіх подій.

Для забезпечення безпеки доступ до адміністративних функцій обмежується через Firebase Security Rules. Правила доступу дозволяють створення, редагування та видалення дорожніх подій виключно користувачам із роллю admin. Звичайні користувачі мають доступ лише до перегляду дорожньої ситуації та роботи із власними даними.

Особливістю реалізації є відсутність окремої серверної адміністративної панелі або backend-сервера. Увесь адміністративний функціонал працює через Firebase Backend-as-a-Service та інтегрований безпосередньо у клієнтський React-застосунок. Такий підхід дозволяє зменшити складність серверної інфраструктури та спростити процес супроводження системи.

Під час експлуатації адміністратор може контролювати актуальність дорожніх подій, перевіряти коректність відображення аварій на карті та

виконувати тестування алгоритмів маршрутизації у різних транспортних ситуаціях. Завдяки realtime-синхронізації всі внесені зміни миттєво відображаються у клієнтських застосунках користувачів.

Таким чином, механізм адміністрування RoadGuard City забезпечує централізоване керування дорожніми подіями, підтримку актуальності даних у режимі реального часу та ефективний контроль роботи системи моніторингу дорожнього руху.

4.4 Демонстрація роботи програми

Для оцінювання працездатності та функціональних можливостей системи RoadGuard City було проведено демонстрацію роботи веб-застосунку у режимі реального часу. У процесі тестування перевірялися основні сценарії взаємодії користувача із системою, зокрема авторизація, побудова маршрутів, відображення дорожніх подій, робота із історією маршрутів та функціонування адміністративного модуля. Інтерфейс системи реалізований у вигляді SPA-застосунку із адаптивним дизайном та підтримкою інтерактивної взаємодії з картою.

Після відкриття веб-застосунку користувач потрапляє на лендінгову сторінку системи, загальний вигляд якої наведено на рис. 26.

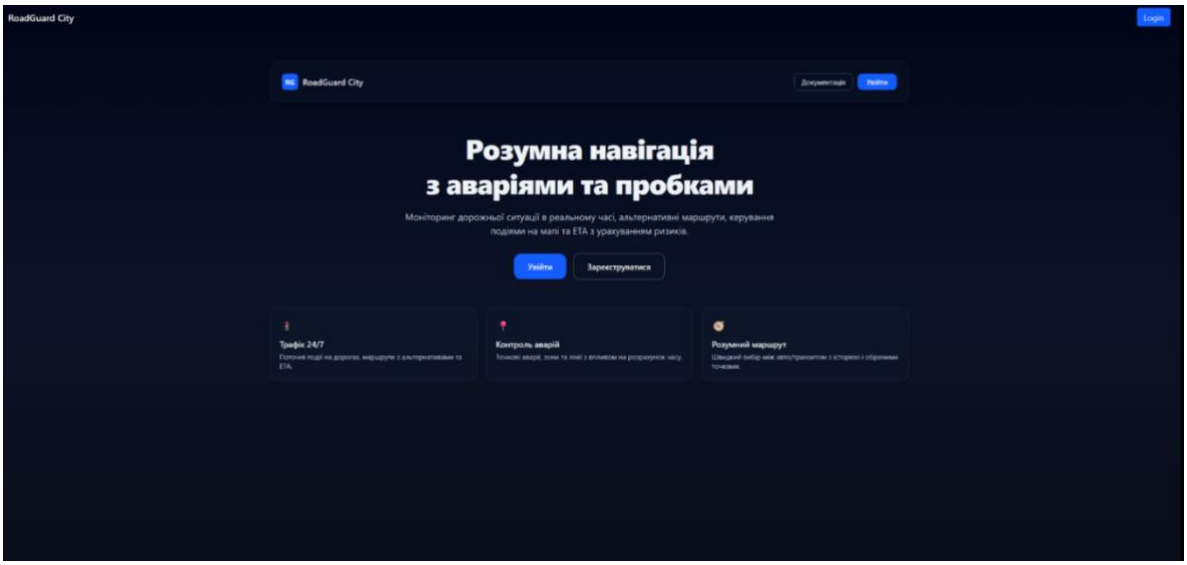

Рис. 26 Лендінгова сторінка системи RoadGuard City

Основним призначенням даної сторінки є ознайомлення користувача із функціональними можливостями RoadGuard City та швидкий перехід до роботи із системою.

Як показано на рис. 26, центральна частина сторінки містить інформаційний блок із коротким описом системи та кнопками переходу до авторизації або реєстрації. У нижній частині розташовані функціональні картки, що демонструють основні можливості застосунку, зокрема моніторинг дорожньої ситуації, контроль аварій та побудову альтернативних маршрутів.

Для доступу до основного функціоналу користувач проходить авторизацію через Firebase Authentication. Сторінка входу до системи наведена на рис. 27.

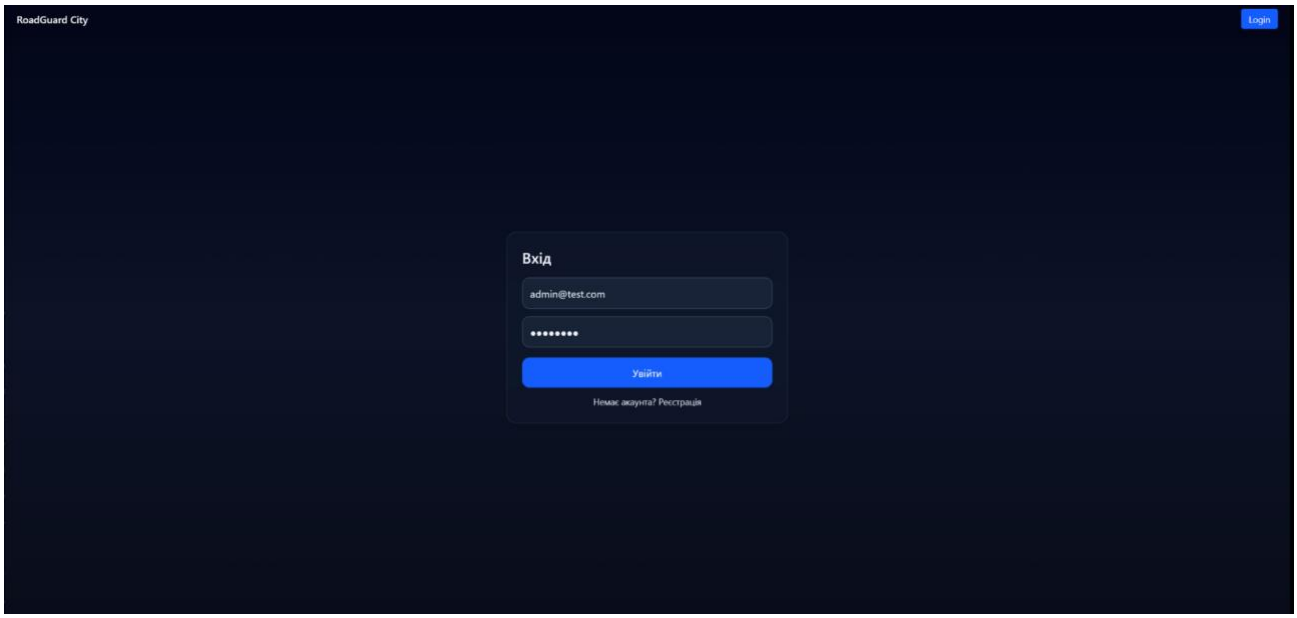

Рис.27 Сторінка авторизації користувача

Інтерфейс авторизації реалізований у спрощеному вигляді та містить поля введення електронної пошти і пароля. Після введення коректних облікових даних система автоматично створює сесію користувача та перенаправляє його до головного робочого інтерфейсу без перезавантаження сторінки.

Основне робоче середовище системи наведено на рис. 28. Даний інтерфейс поєднує інтерактивну карту, модуль побудови маршрутів та панель навігації між функціональними компонентами застосунку.

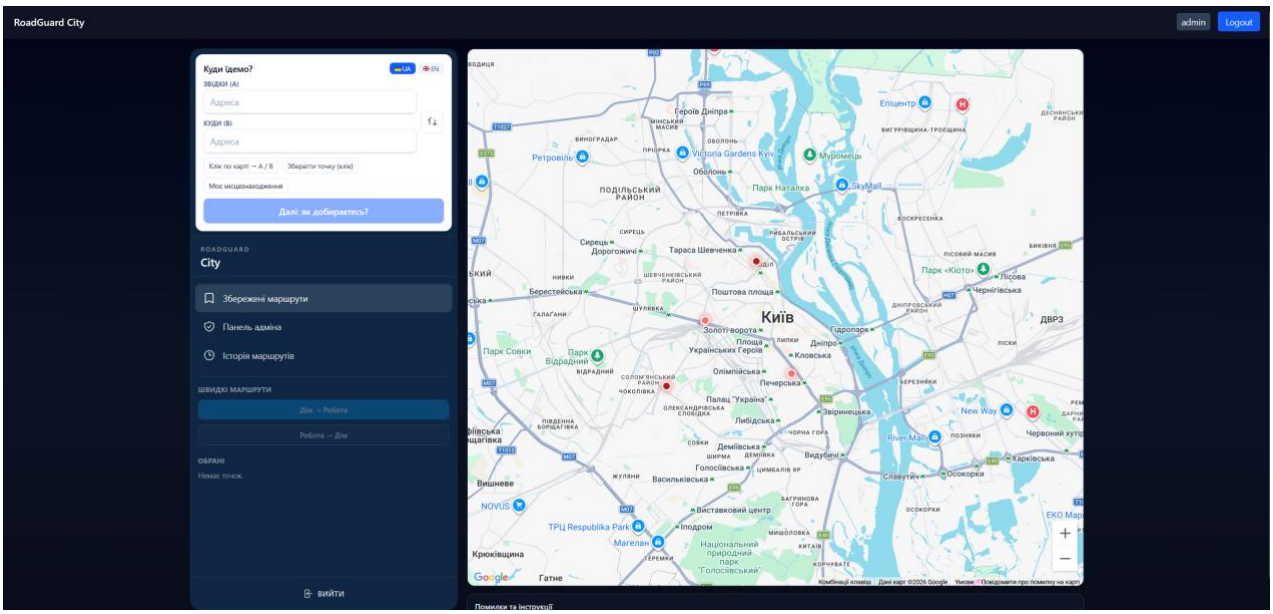

Рис. 28 Робочий інтерфейс системи

Центральну частину інтерфейсу займає карта Google Maps, на якій у режимі реального часу відображаються дорожні події, аварії та побудовані маршрути. У лівій частині екрана розташована функціональна бокова панель, через яку користувач задає початкову та кінцеву точки маршруту, використовує поточне місцезнаходження або працює із збереженими маршрутами.

Після введення адрес система виконує запит до Google Directions API та формує декілька альтернативних маршрутів. Результат роботи алгоритму наведено на рис. 29.

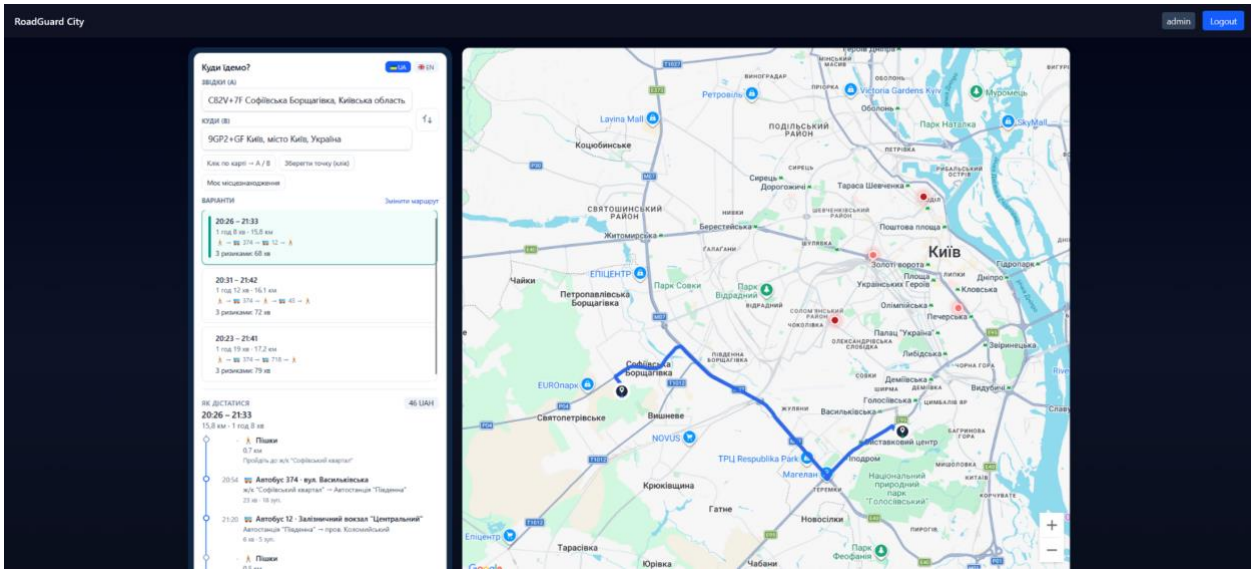

Рис.29 Побудова та аналіз маршрутів

Як показано на рис. 29, система відображає список доступних маршрутів із зазначенням орієнтовного часу прибуття, довжини маршруту та рівня ризику. Після вибору маршруту його траєкторія відображається на карті, а користувач отримує покрокову інформацію про пересування.

У системі також реалізовано механізм збереження та повторного використання маршрутів. Для цього передбачено окремий розділ історії маршрутів, наведений на рис. 30.

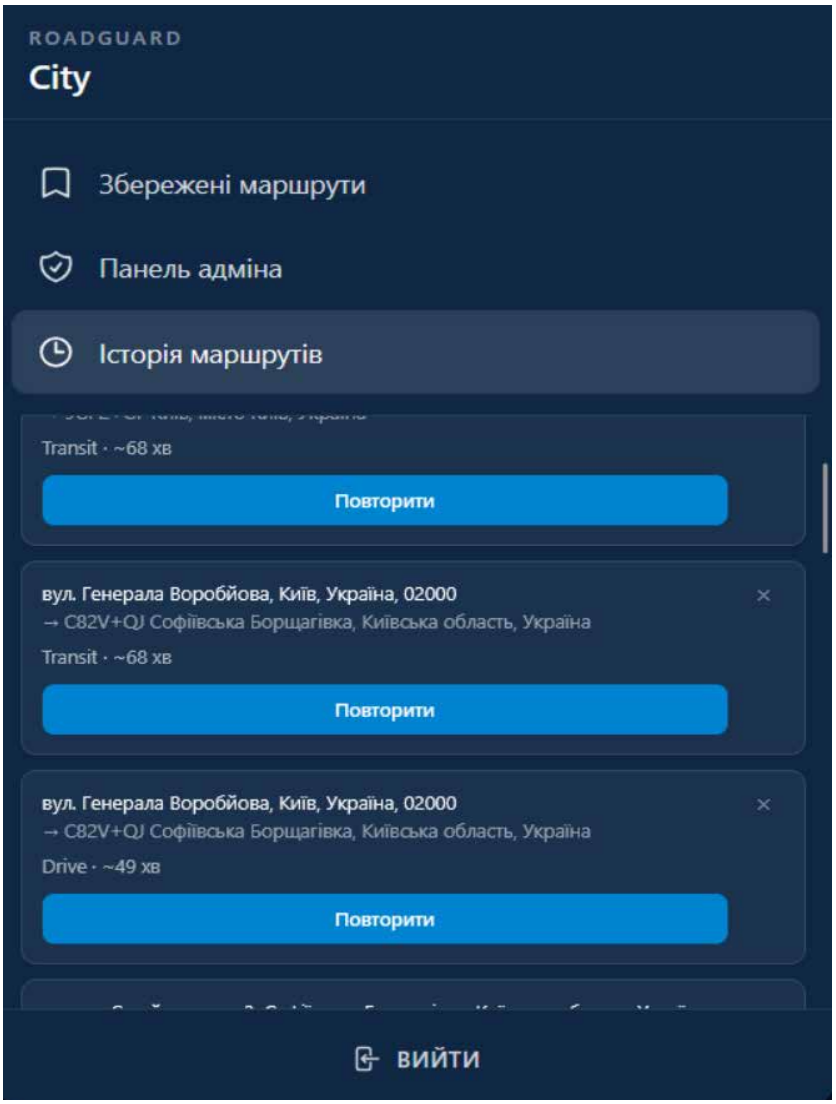

Рис. 30 Історія маршрутів користувача

Розділ історії маршрутів дозволяє переглядати попередні маршрути, повторно запускати побудову шляху та швидко отримувати доступ до часто використовуваних напрямків. Для кожного запису відображаються початкова та кінцева точки, тип маршруту та орієнтовний час пересування.

Окрему роль у системі виконує адміністративний модуль, доступний лише користувачам із роллю admin. Загальний вигляд адміністративної панелі наведено на рис. 31.

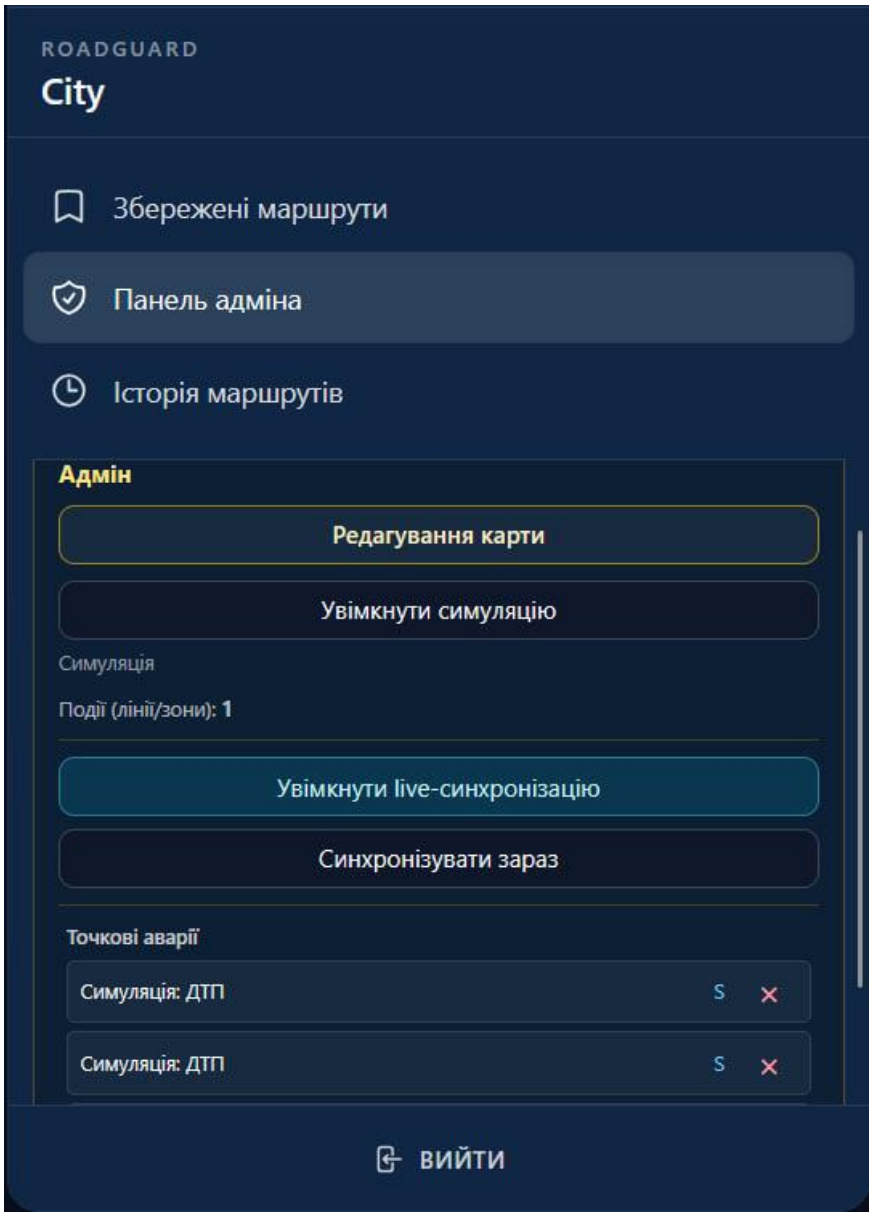

Рис 31. Адміністративна панель системи RoadGuard City

Адміністративна панель інтегрована безпосередньо у головний інтерфейс системи та дозволяє виконувати керування дорожніми подіями у режимі реального часу. Адміністратор може активувати режим редагування карти, створювати аварії, запускати симуляцію дорожніх подій та керувати механізмом live-синхронізації.

Під час створення аварії адміністратор задає точку події на карті, тип інциденту та рівень небезпечності. Після збереження інформація автоматично записується у Cloud Firestore та миттєво відображається у всіх активних користувачів через realtime listeners Firebase.

Для підтримки актуальності інформації у системі реалізовано модуль live-синхронізації дорожніх подій. Після його активації застосунок виконує отримання даних із зовнішнього JSON feed та автоматично додає нові інциденти до бази даних Firestore. Адміністратор також може вручну виконувати синхронізацію через відповідну кнопку інтерфейсу.

Завдяки реалізації SPA-архітектури всі дії користувача виконуються без повного перезавантаження сторінки, що позитивно впливає на швидкодію системи та зручність роботи із картою. Використання Firebase забезпечує realtime-синхронізацію подій, а інтеграція із Google Maps Platform дозволяє відображати актуальну дорожню ситуацію та виконувати аналіз маршрутів у режимі реального часу.

Таким чином, реалізований користувацький інтерфейс RoadGuard City забезпечує зручну взаємодію із системою моніторингу дорожнього руху, підтримує побудову інтелектуальних маршрутів та надає інструменти адміністрування дорожніх подій у межах єдиного web-oriented середовища.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи розроблено веб-застосунок RoadGuard City, призначений для моніторингу дорожньої ситуації та побудови маршрутів з урахуванням аварій, заторів і транспортних подій у режимі реального часу. Розроблена система забезпечує відображення дорожньої інформації на інтерактивній карті, підтримує аналіз маршрутів та дозволяє оперативно оновлювати дані про транспортну ситуацію у міському середовищі.

У процесі виконання роботи проведено аналіз сучасних систем навігації та моніторингу дорожнього руху, визначено основні проблеми, пов’язані з відображенням дорожніх подій, оперативністю оновлення інформації та побудовою маршрутів в умовах динамічної зміни транспортної ситуації. На основі проведеного аналізу сформовано функціональні та нефункціональні вимоги до програмної системи, визначено основні сценарії взаємодії користувачів із застосунком та обґрунтовано вибір технологічного стеку.

Для реалізації системи використано React, TypeScript та Vite, що дозволило створити SPA-застосунок із динамічним оновленням даних без перезавантаження сторінки. Для роботи з картографічними сервісами інтегровано Google Maps Platform, зокрема Maps JavaScript API, Directions API та Places API. Це забезпечило підтримку інтерактивної карти, побудову маршрутів, роботу з геоданими та відображення дорожніх подій у режимі реального часу.

У ролі хмарної платформи використано Firebase. За допомогою Firebase Authentication реалізовано механізми автентифікації користувачів та підтримку користувацьких сесій. Cloud Firestore використовується для зберігання інформації про дорожні події, маршрути та користувацькі дані, а також для реалізації realtime-синхронізації між клієнтськими застосунками. Реалізований підхід дозволив відмовитися від окремого backend-сервера та спростити серверну інфраструктуру системи.

У програмному забезпеченні реалізовано функціонал створення, редагування та відображення дорожніх подій, механізми аналізу маршрутів і модуль адміністративного керування. Система підтримує побудову альтернативних маршрутів з урахуванням поточної дорожньої ситуації, а також забезпечує автоматичне оновлення інформації без необхідності ручного перезавантаження сторінки.

Під час тестування перевірено працездатність основних функціональних можливостей системи, коректність взаємодії із Firebase та Google Maps Platform, стабільність realtime-синхронізації та правильність роботи механізмів маршрутизації. Результати тестування підтвердили коректність роботи реалізованого програмного забезпечення та відповідність системи поставленим вимогам.

Практичним результатом роботи є створення працездатного веб-застосунку для моніторингу дорожньої ситуації у міському середовищі. Розроблена система може бути використана як основа для подальшого розвитку інформаційних транспортних сервісів, інтеграції з додатковими джерелами дорожніх даних та розширення аналітичних можливостей системи моніторингу дорожнього руху.

.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google Maps Platform Documentation. URL: https://developers.google.com/maps.
2. Google Maps Platform. Routes API Documentation. URL: https://developers.google.com/maps/documentation/routes.
3. Google Maps Platform. Maps JavaScript API Documentation. URL: https://developers.google.com/maps/documentation/javascript.
4. Google Maps Platform. Places API Documentation. URL: https://developers.google.com/maps/documentation/places/web-service.
5. Firebase Documentation. URL: https://firebase.google.com/docs.
6. Cloud Firestore Documentation. Firebase. URL: https://firebase.google.com/docs/firestore.
7. Firebase Authentication Documentation. URL: https://firebase.google.com/docs/auth.
8. Firebase Security Rules Documentation. URL: https://firebase.google.com/docs/rules.
9. React Documentation. URL: https://react.dev/.
10. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/.
11. Vite Documentation. URL: https://vite.dev/.
12. React Router Documentation. URL: https://reactrouter.com/.
13. Zustand Documentation. URL: https://zustand.docs.pmnd.rs/.
14. npm Documentation. URL: https://docs.npmjs.com/.
15. Node.js Documentation. URL: https://nodejs.org/docs/latest/api/.
16. MDN Web Docs. Geolocation API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API.
17. MDN Web Docs. WebSocket API. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.
18. MDN Web Docs. JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
19. MDN Web Docs. CSS. URL: https://developer.mozilla.org/en-US/docs/Web/CSS.

20. WHATWG. HTML Living Standard. URL: https://html.spec.whatwg.org/.
21. UNECE. Intelligent Transport Systems (ITS) for Sustainable Mobility. 2nd ed. Geneva : United Nations Economic Commission for Europe, 2024. URL: https://unece.org/sites/default/files/2024-06/ITS%20for%20sustainable%20Mobility_E_pdf_web.pdf
22. CINEA. Intelligent Transport Systems in the EU. European Commission. URL: https://cinea.ec.europa.eu/programmes/connecting-europe-facility/transport-infrastructure/intelligent-transport-systems-eu_en.
23. ElSahly O., Abdelfatah A., Khamis A. A Systematic Review of Traffic Incident Detection Algorithms. Sustainability. 2022. Vol. 14, № 22. Article 14859. DOI: https://doi.org/10.3390/su142214859.
24. Zohir H. M. et al. Advancements in accident-aware traffic management. Scientific Reports. 2025. URL: https://www.nature.com/articles/s41598-025-20878-x.
25. Lodhia A. S. An Investigation into the Recent Developments in Intelligent Transport Systems. EASTS Proceedings. 2023. URL: https://easts.info/online/proceedings/vol.14/pdf/PP3808.pdf.
26. Freeman A. Pro React 18. Berkeley : Apress, 2022. 752 p.
27. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 310 p.
28. Cherny B. Programming TypeScript: Making Your JavaScript Applications Scale. Sebastopol : O'Reilly Media, 2019. 324 p.
29. Гужва В. М. *Інформаційні системи і технології на підприємствах : навчальний посібник*. Київ : КНЕУ, 2021. 400 с.
30. Вакалюк Т. А., Спірін О. М. *Web-технології та Web-дизайн : навчальний посібник*. Житомир : Вид-во ЖДУ ім. І. Франка, 2023. 300 с.
31. Соммервілл І. Інженерія програмного забезпечення / пер. з англ. Київ : Вільямс, 2021. 912 с.
32. Буч Г., Рамбо Дж., Джекобсон А. UML. Класика проєктування інформаційних систем / пер. з англ. Харків : Ранок, 2020. 672 с.
33. Плескач В. Л., Затонацька Т. Г. *Інформаційні системи і технології на*

підприємствах : підручник. Київ : Знання, 2021. 718 с.

34. Кужель В. П., Данчук В. Д. Інтелектуальні транспортні системи в умовах цифровізації міської інфраструктури // *Вісник Національного транспортного університету*. 2022. № 3(53). С. 45–53.
35. Дубина М. В., Тарасенко А. В. Використання геоінформаційних систем у транспортному моніторингу міст // *Системи управління, навігації та зв'язку*. 2023. № 2. С. 87–94.
36. Спірін О. М., Носенко Ю. Г. Хмарні технології у побудові сучасних інформаційних систем // *Фізико-математична освіта*. 2022. № 1(31). С. 78–86.

ДОДАТОК А

Лістинг А.1 Код підключення БД і Auth

```
aimport { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
import { getFirestore } from 'firebase/firestore';

const firebaseConfig = {
  apiKey: import.meta.env.VITE_FIREBASE_API_KEY,
  authDomain: import.meta.env.VITE_FIREBASE_AUTH_DOMAIN,
  projectId: import.meta.env.VITE_FIREBASE_PROJECT_ID,
  storageBucket: import.meta.env.VITE_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: import.meta.env.VITE_FIREBASE_MESSAGING_SENDER_ID,
  appId: import.meta.env.VITE_FIREBASE_APP_ID,
};

const requiredKeys = ['apiKey', 'authDomain', 'projectId', 'appId'] as const;
for (const key of requiredKeys) {
  if (!firebaseConfig[key]) {
    // Throw early to make missing env setup obvious.
    throw new Error(`Missing Firebase config: ${key}`);
  }
}

export const firebaseApp = initializeApp(firebaseConfig);
export const auth = getAuth(firebaseApp);
export const db = getFirestore(firebaseApp);
```

Лістинг А.2 Код формування профілей users

```
import { doc, getDoc, setDoc } from 'firebase/firestore';
import { db } from './firebase';

export type UserRole = 'user' | 'admin';

export const resolveRoleByEmail = (email: string | null | undefined): UserRole =>
  email === 'admin@test.com' ? 'admin' : 'user';

export const ensureUserProfile = async (uid: string, email: string) => {
  const ref = doc(db, 'users', uid);
  const snap = await getDoc(ref);
  const role = resolveRoleByEmail(email);
  if (!snap.exists()) {
    await setDoc(ref, { email, role });
    return role;
  }
  return (snap.data().role as UserRole) ?? role;
};
```

Лістинг А.3 Код формування «Колекція аварій»

```
import {
  addDoc,
  collection,
```

```
  deleteDoc,
  doc,
  onSnapshot,
  orderBy,
  query,
  serverTimestamp,
  updateDoc,
} from 'firebase/firestore';
import { db } from './firebase';

export type Severity = 'low' | 'medium' | 'high';

export interface AccidentRecord {
  id: string;
  lat: number;
  lng: number;
  description: string;
  severity: Severity;
  createdAt?: Date;
}

const accidentsCollection = collection(db, 'accidents');

export const subscribeAccidents = (callback: (items: AccidentRecord[]) => void) =>
  onSnapshot(query(accidentsCollection, orderBy('createdAt', 'desc')), (snapshot)
=> {
    callback(
      snapshot.docs.map((d) => ({
        id: d.id,
        lat: d.data().lat,
        lng: d.data().lng,
        description: d.data().description,
        severity: d.data().severity,
        createdAt: d.data().createdAt?.toDate?.(),
      })),
    );
  });

export const addAccident = async (payload: Omit<AccidentRecord, 'id' |
'createdAt'>) =>
  addDoc(accidentsCollection, {
    ...payload,
    createdAt: serverTimestamp(),
  });

export const updateAccident = async (
  id: string,
  payload: Partial<Pick<AccidentRecord, 'description' | 'severity'>>,
) => updateDoc(doc(db, 'accidents', id), payload);

export const deleteAccident = async (id: string) => deleteDoc(doc(db, 'accidents',
id));
```

Лістинг А.4 Код формування «Колекція дорожніх подій»

```
/**
 * Події на карті: лінії та полігони (Firestore: trafficEvents).
```

```
 * geometryType: polyline | polygon; path – масив координат.
 */
import { addDoc, collection, deleteDoc, doc, onSnapshot, query, serverTimestamp }
from 'firebase/firestore';
import { db } from './firebase';

export type TrafficEventType = 'accident' | 'traffic';
export type Severity = 'low' | 'medium' | 'high';
export type GeometryKind = 'polyline' | 'polygon';

export interface TrafficEventRecord {
  id: string;
  type: TrafficEventType;
  geometryType: GeometryKind;
  path: { lat: number; lng: number }[];
  severity: Severity;
  createdAt?: Date;
}

const col = collection(db, 'trafficEvents');

export const subscribeTrafficEvents = (cb: (rows: TrafficEventRecord[]) => void) =>
  onSnapshot(query(col), (snap) => {
    const rows: TrafficEventRecord[] = snap.docs.map((d) => {
      const data = d.data();
      const rawPath = Array.isArray(data.path) ? data.path : [];
      return {
        id: d.id,
        type: data.type,
        geometryType: (data.geometryType as GeometryKind) ?? 'polyline',
        path: rawPath,
        severity: data.severity ?? 'medium',
        createdAt: data.createdAt?.toDate?.(),
      };
    });
    rows.sort((a, b) => (b.createdAt?.getTime() ?? 0) - (a.createdAt?.getTime() ??
0));
    cb(rows);
  });

export const addTrafficEvent = async (payload: Omit<TrafficEventRecord, 'id' |
'createdAt'>) =>
  addDoc(col, {
    type: payload.type,
    geometryType: payload.geometryType,
    path: payload.path,
    severity: payload.severity,
    createdAt: serverTimestamp(),
  });

export const deleteTrafficEvent = async (id: string) => deleteDoc(doc(db,
'trafficEvents', id));
```

Лістинг А.5 Код формування «Історія маршрутів»

```
/**
 * Історія маршрутів користувача (Firestore: routesHistory).
 */
```

```
import { addDoc, collection, deleteDoc, doc, onSnapshot, query, serverTimestamp,
where } from 'firebase/firestore';
import { db } from './firebase';

export type TransportType = 'driving' | 'transit';

export interface RouteHistoryRecord {
  id: string;
  userId: string;
  from: string;
  to: string;
  transportType: TransportType;
  /** Тривалість у хвилинах (з урахуванням штрафів, якщо збереглі) */
  durationMinutes: number;
  createdAt?: Date;
}

const col = collection(db, 'routesHistory');

export const subscribeRouteHistory = (userId: string, cb: (rows:
RouteHistoryRecord[]) => void) =>
  onSnapshot(query(col, where('userId', '==', userId)), (snap) => {
    const rows: RouteHistoryRecord[] = snap.docs.map((d) => {
      const data = d.data();
      return {
        id: d.id,
        userId: data.userId,
        from: data.from,
        to: data.to,
        transportType: data.transportType ?? 'driving',
        durationMinutes: typeof data.durationMinutes === 'number' ?
data.durationMinutes : 0,
        createdAt: data.createdAt?.toDate?.(),
      };
    });
    rows.sort((a, b) => (b.createdAt?.getTime() ?? 0) - (a.createdAt?.getTime() ??
0));
    cb(rows);
  });

export const addRouteHistory = async (payload: Omit<RouteHistoryRecord, 'id' |
'createdAt'>) =>
  addDoc(col, {
    ...payload,
    createdAt: serverTimestamp(),
  });

export const deleteRouteHistory = async (id: string) => deleteDoc(doc(db,
'routesHistory', id));
```

Лістинг А.6 Код формування «Улюблені місця»

```
/**
 * Обрані точки: Дім / Робота / Інше (Firestore: favoritePlaces).
 */
import {
  addDoc,
```

```
  collection,
  deleteDoc,
  doc,
  getDocs,
  onSnapshot,
  query,
  serverTimestamp,
  updateDoc,
  where,
} from 'firebase/firestore';
import { db } from './firebase';

export type FavoriteLabel = 'home' | 'work' | 'other';

export interface FavoritePlaceRecord {
  id: string;
  userId: string;
  label: FavoriteLabel;
  name: string;
  lat: number;
  lng: number;
  createdAt?: Date;
}

const col = collection(db, 'favoritePlaces');

export const subscribeFavorites = (userId: string, cb: (rows:
FavoritePlaceRecord[]) => void) =>
  onSnapshot(query(col, where('userId', '==', userId)), (snap) => {
    cb(
      snap.docs.map((d) => {
        const data = d.data();
        return {
          id: d.id,
          userId: data.userId,
          label: data.label,
          name: data.name ?? '',
          lat: data.lat,
          lng: data.lng,
          createdAt: data.createdAt?.toDate?.(),
        };
      }),
    );
  });

export const addFavoritePlace = async (
  payload: Omit<FavoritePlaceRecord, 'id' | 'createdAt'>,
) =>
  addDoc(col, {
    ...payload,
    createdAt: serverTimestamp(),
  });

/**
 * Одна точка на label – оновлення існуючого документа замість дубліката.
 * Тільки `where('userId')`, далі фільтр по label у клієнті (без складного
composite index у Firestore).
 */
```

```
export const upsertFavoritePlace = async (payload: Omit<FavoritePlaceRecord, 'id' |
'createdAt'>) => {
  const q = query(col, where('userId', '==', payload.userId));
  const snap = await getDocs(q);
  const existing = snap.docs.find((d) => d.data().label === payload.label);
  if (!existing) {
    await addDoc(col, {
      ...payload,
      createdAt: serverTimestamp(),
    });
    return;
  }
  await updateDoc(existing.ref, {
    name: payload.name,
    lat: payload.lat,
    lng: payload.lng,
  });
};

export const deleteFavoritePlace = async (id: string) => deleteDoc(doc(db,
'favoritePlaces', id));
```

ДОДАТОК Б

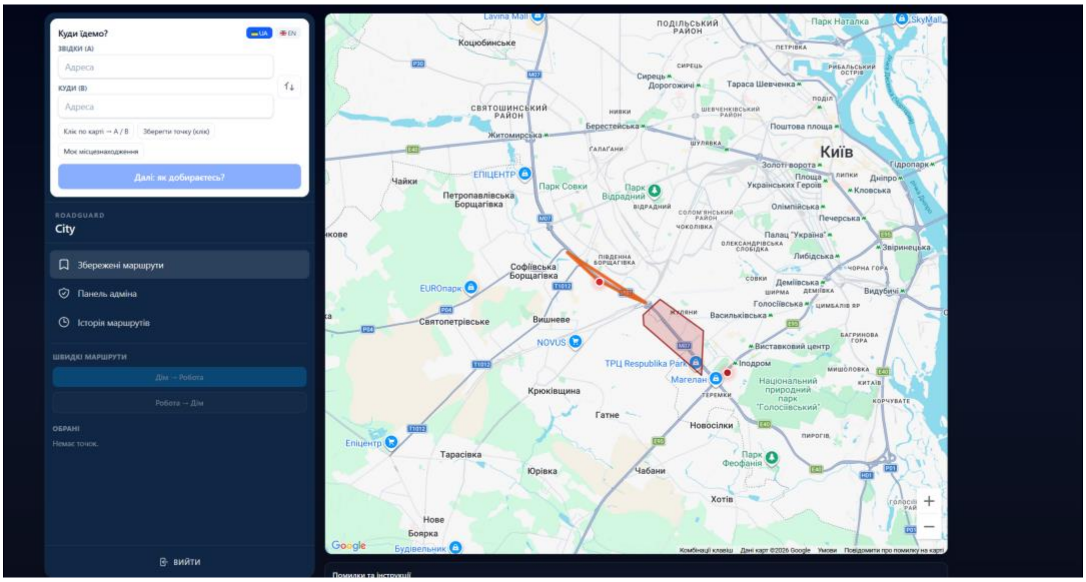

Рис. Б.1 Види аварій

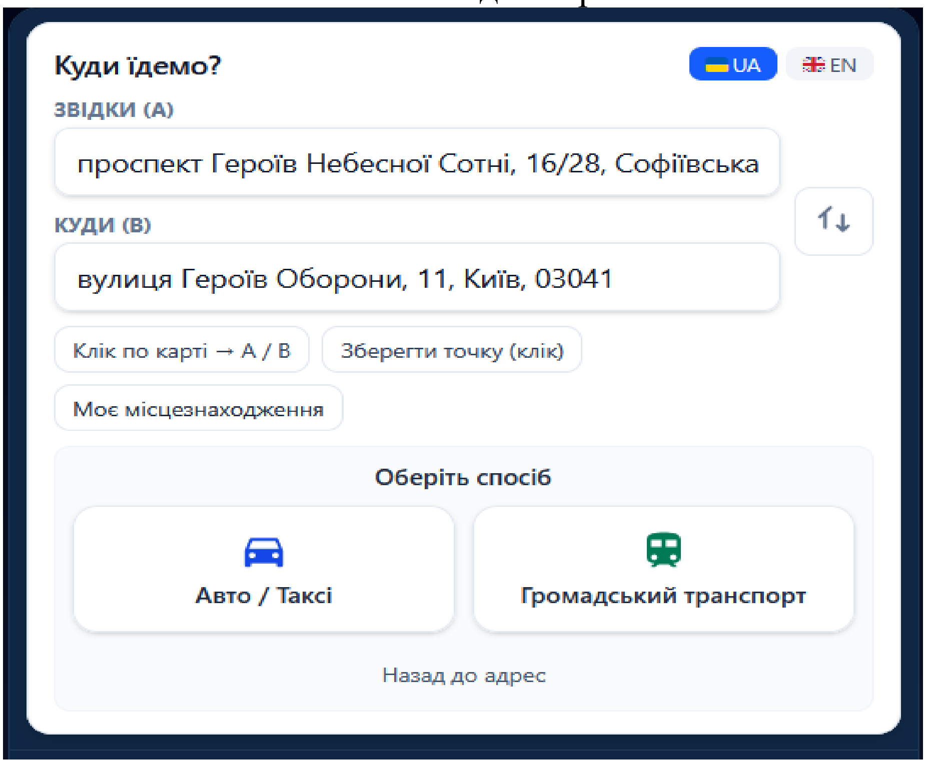

Рис Б.2 Вибір виду транспорту

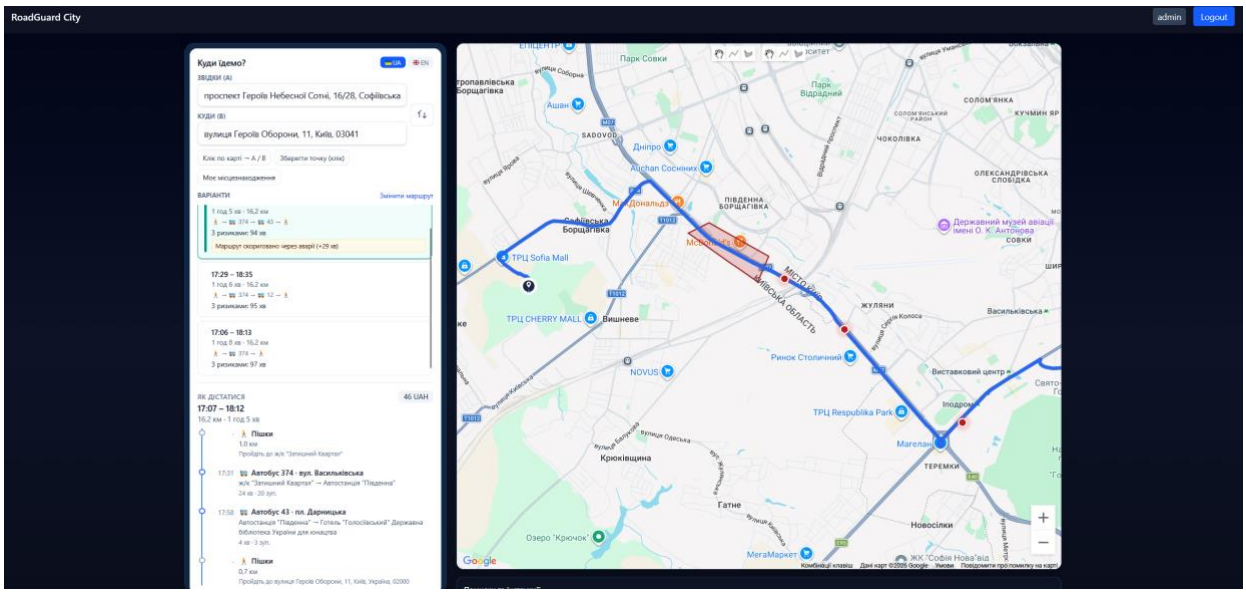

Рис Б.3 Формування маршруту за обраним видом транспорту

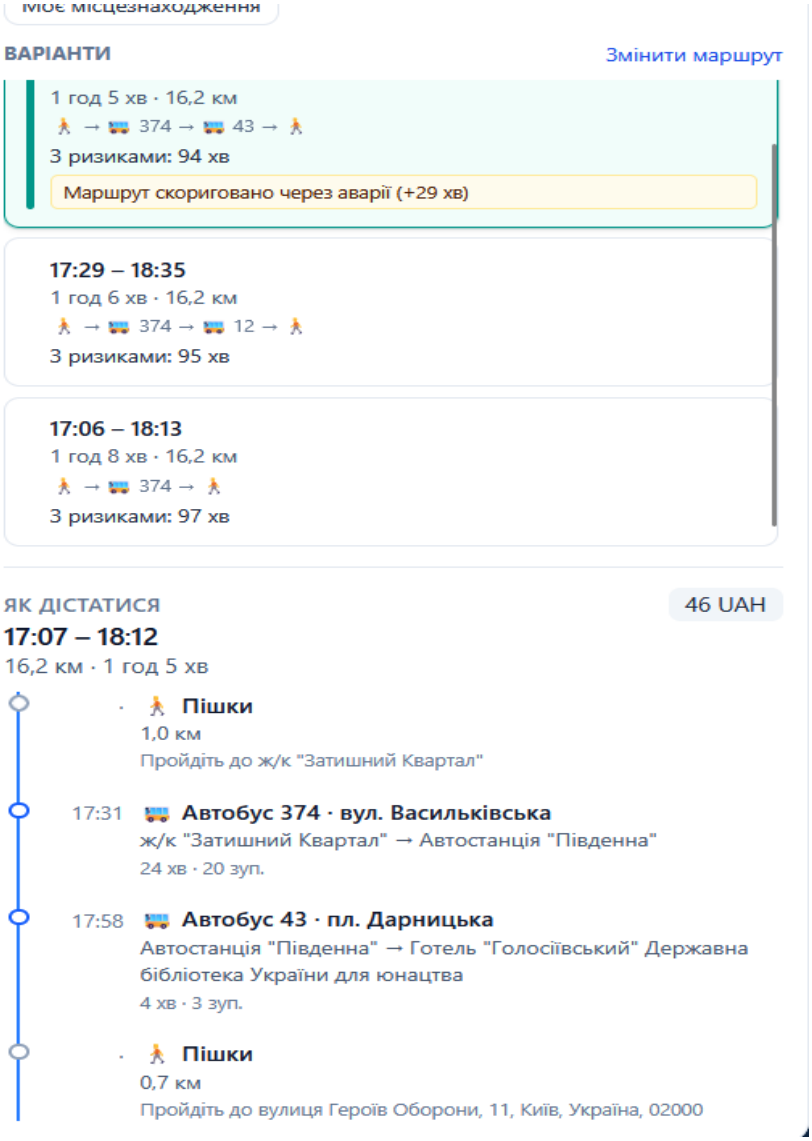

Рис Б.4 Побудова маршруту , коригування часу прибуття через аварії, та інструкція як добиратися до точки

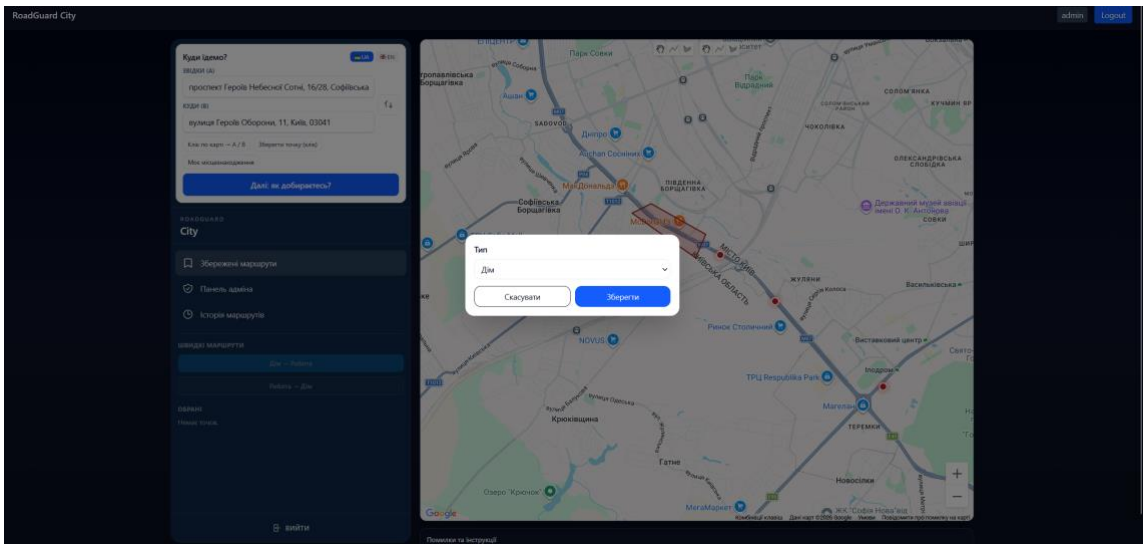

Рис Б.5 Збереження точки, для побудови швидких маршрутів на кожен день

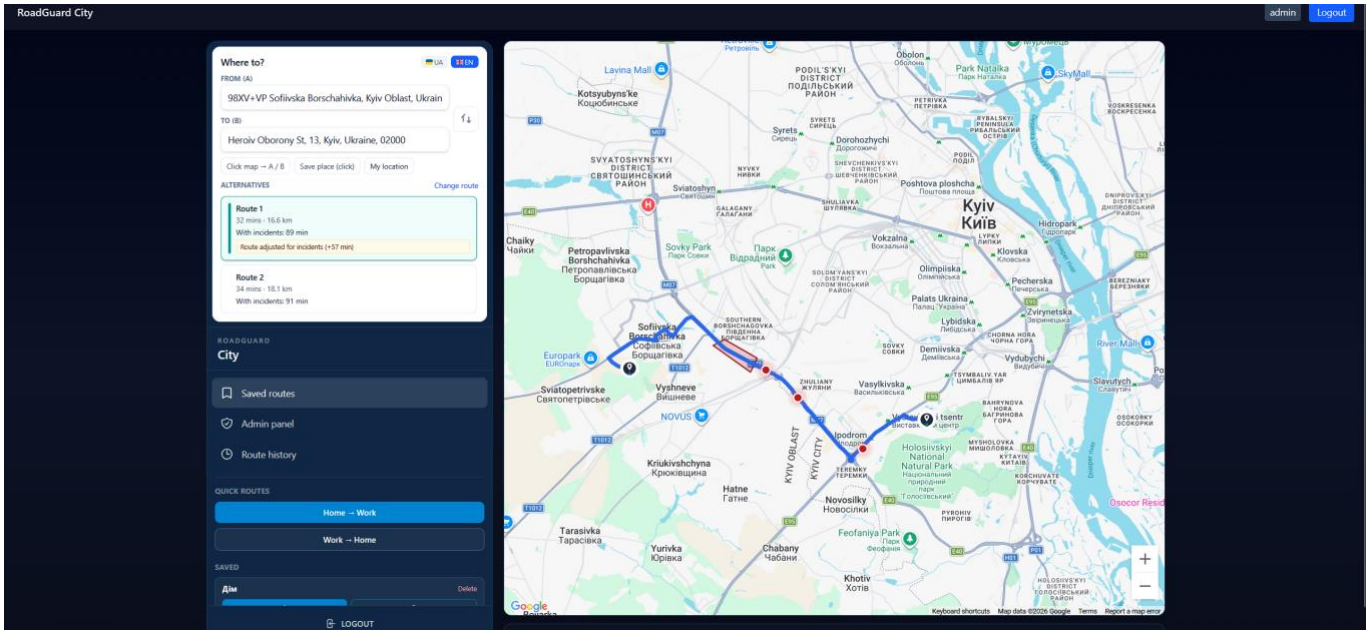

Рис. Б.6 Зміна мови інтерфейсу

ДОДАТОК В

Фрагменти програмної реалізації основних модулів системи RoadGuard City

Лістинг В.1 – Реалізація авторизації користувача

```
// AuthProvider.tsx

const register = async (email: string, password: string) => {
  await createUserWithEmailAndPassword(auth, email, password);
};

const login = async (email: string, password: string) => {
  await signInWithEmailAndPassword(auth, email, password);
};

const logout = async () => {
  await signOut(auth);
};
```

Наведені функції реалізують базову логіку автентифікації користувачів у вебзастосунку RoadGuard City. Для роботи із системою автентифікації використовується Firebase Authentication SDK, який забезпечує реєстрацію користувачів, авторизацію та підтримку активної сесії.

Функція register() використовується для створення нового облікового запису користувача через електронну пошту та пароль. Функція login() забезпечує вхід користувача до системи, а logout() виконує завершення поточної сесії та вихід із застосунку.

Лістинг В.2 – Відстеження стану авторизації користувача

```
// AuthProvider.tsx

useEffect(() => {
  const unsub = onAuthStateChanged(auth, async (nextUser) => {
    setUser(nextUser);

    if (nextUser?.email) {
      const userRole = await ensureUserProfile(
        nextUser.uid,
        nextUser.email
      );

      setRole(userRole);
    } else {
      setRole(null);
    }

    setLoading(false);
  });

  return unsub;
}, []);
```

Для підтримки актуального стану авторизації використовується функція onAuthStateChanged(), яка автоматично відстежує зміну стану користувача у Firebase Authentication. Після авторизації або оновлення сторінки система перевіряє наявність користувача та виконує завантаження його ролі із Cloud Firestore.

Такий підхід дозволяє забезпечити централізоване керування сесією користувача та підтримку механізму розмежування ролей у системі.

Лістинг В.3 – Формування профілю користувача у Firestore

```
// users.ts

export const ensureUserProfile = async (
  uid: string,
  email: string
) => {
  const ref = doc(db, 'users', uid);

  const snap = await getDoc(ref);

  const role = resolveRoleByEmail(email);

  if (!snap.exists()) {
    await setDoc(ref, {
      email,
      role,
    });

    return role;
  }

  return snap.data().role ?? role;
};
```

Функція ensureUserProfile() відповідає за створення профілю користувача у колекції users бази даних Cloud Firestore. Після успішної авторизації система перевіряє існування документа користувача та у разі його відсутності автоматично створює новий запис.

У документі зберігаються електронна пошта користувача та його роль у системі. Реалізація такого механізму дозволяє забезпечити централізоване керування користувачами та підтримку адміністративного функціоналу.

Лістинг В.4 – Ініціалізація Firebase та підключення модулів системи

```
// firebase.ts

export const firebaseApp = initializeApp(firebaseConfig);

export const auth = getAuth(firebaseApp);

export const db = getFirestore(firebaseApp);
```

Ініціалізація Firebase виконується у файлі firebase.ts. Після створення екземпляра Firebase-застосунку система окремо ініціалізує модуль автентифікації та модуль Cloud Firestore.

Об’єкт auth використовується для роботи з авторизацією користувачів, а db забезпечує взаємодію із хмарною базою даних Firestore.

Лістинг В.5 – Реалізація побудови маршруту

```
// RoutePanel.tsx

const handleBuildRoute = async () => {
  if (!origin || !destination) return;

  setLoading(true);

  try {
    const result = await buildRoute({
      origin,
      destination,
      mode,
    });

    setRoutes(result.routes);
  } catch (error) {
    showToast('Помилка побудови маршруту');
  } finally {
    setLoading(false);
  }
};
```

Функція handleBuildRoute() виконує запуск процесу побудови маршруту між двома точками. Перед формуванням маршруту система перевіряє наявність початкової та кінцевої точки маршруту, після чого виконується звернення до Google Directions API.

Отримані маршрути зберігаються у стані застосунку та відображаються на інтерактивній карті. У разі виникнення помилки система формує повідомлення для користувача.

Лістинг В.6 – Взаємодія з Google Directions API

```
// directions.ts

export async function buildRoute({
  origin,
  destination,
  mode,
}: BuildRouteParams) {

  const directionsService =
    new google.maps.DirectionsService();

  const response = await directionsService.route({
    origin,
    destination,
    travelMode: mode,
    provideRouteAlternatives: true,
  });

  return response;
}
```

Функція buildRoute() реалізує взаємодію із Google Directions API через JavaScript SDK платформи Google Maps. Для побудови маршруту використовується сервіс DirectionsService.

Параметр travelMode визначає тип пересування користувача, а параметр provideRouteAlternatives забезпечує отримання альтернативних маршрутів для подальшого аналізу дорожньої ситуації.

Лістинг В.7 – Аналіз дорожніх подій уздовж маршруту

```
// routeAnalysis.ts

export function analyzeRouteIncidents(
  route,
  accidents,
  trafficEvents
) {

  const risks = [];

  accidents.forEach((accident) => {
    if (isPointNearRoute(accident.position, route)) {
      risks.push({
        type: 'accident',
        data: accident,
      });
    }
  });

  trafficEvents.forEach((event) => {
    if (isPointNearRoute(event.position, route)) {
      risks.push({
        type: 'traffic',
        data: event,
      });
    }
  });

  return risks;
}
```

Функція analyzeRouteIncidents() виконує аналіз дорожніх подій уздовж маршруту користувача. Система перевіряє близькість аварій та транспортних подій до маршруту та формує список потенційних ризиків.

Отримані результати використовуються для оцінювання безпечності маршруту та формування рекомендацій щодо альтернативних варіантів пересування.

Лістинг В.8 – Реалізація realtime-оновлення дорожніх подій

```
// accidents.ts

export const subscribeAccidents = (callback) => {

  return onSnapshot(
    collection(db, 'accidents'),
    (snapshot) => {

      const items = snapshot.docs.map((doc) => ({
        id: doc.id,
```

```
      ...doc.data(),
    }));

    callback(items);
  }
 );
};
```

Для підтримки режиму реального часу система використовує Firestore realtime listeners через функцію onSnapshot().

Після будь-якої зміни у колекції accidents Firebase автоматично надсилає оновлені дані клієнтському застосунку без необхідності повторного HTTP-запиту. Це забезпечує оперативне оновлення інформації про дорожні події та підтримку realtime-моніторингу транспортної ситуації.

Лістинг В.9 – Збереження маршруту до історії користувача

```
// routesHistory.ts

export const saveRouteHistory = async (
  userId,
  route
) => {

  await addDoc(
    collection(db, 'routesHistory'),
    {
      userId,
      route,
      createdAt: serverTimestamp(),
    }
  );
};
```

Функція saveRouteHistory() використовується для збереження інформації про побудовані маршрути у Cloud Firestore. Після успішної побудови маршруту система створює новий документ у колекції routesHistory, де зберігаються параметри маршруту, ідентифікатор користувача та дата створення запису.

Лістинг В.10 – Отримання історії маршрутів користувача

```
// routesHistory.ts

export const subscribeRouteHistory = (
  userId,
  callback,
) => {

  const q = query(
    collection(db, 'routesHistory'),
    where('userId', '==', userId),
  );

  return onSnapshot(q, (snapshot) => {

    const routes = snapshot.docs.map((doc) => ({
```

```
      id: doc.id,
      ...doc.data(),
    }));

    callback(routes);
  });
};
```

Функція subscribeRouteHistory() забезпечує отримання історії маршрутів користувача у режимі реального часу. Для цього використовується Firestore listener та фільтрація документів за ідентифікатором користувача.

Завдяки використанню serverless-архітектури Firebase система забезпечує автоматичну синхронізацію даних між базою даних та клієнтським застосунком без використання окремого backend-сервера.

ДОДАТОК Д

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Мельниченко Тимур Русланович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення моніторингу завантаженості та аварійності на дорогах міст» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [ ] інструменти генеративного ШІ не використовувалися.
- [ ] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
  - [ ] перекладу іншомовних джерел;
  - [ ] стилістичного редагування та перевірки граматики;
  - [ ] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

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

« » 2026 р. _____ **Тимур МЕЛЬНИЧЕНКО**