

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»

Завідувач кафедри

Наталія АУШЕВА

“ ” 2026 р.

**Дипломна робота
на здобуття ступеня бакалавр**

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Система аналізу та автоматизації розкладів громадського транспорту”

Виконав: студент 4 курсу, групи ТР-25

Кудрявцев Єгор Олегович

(прізвище, ім’я, по батькові)

(підпис)

Керівник асистент Антон ПАСІЧНЮК

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

(підпис)

Рецензент доцент каф. ТАЕ, к.т.н., Артур РАЧИНСЬКИЙ

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

(підпис)

Н.контроль асистент Костянтин ЗДОР

(посада, ім’я, ПРІЗВИЩЕ)

(підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Студент _____

Київ - 2026

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)

спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ

Завідувач кафедри ЦТЕ

Наталія АУШЕВА
(підпис)

“ ” _____ 2026 р.

ЗАВДАННЯ

на дипломну роботу студенту

Кудрявцеву Єгору Олеговичу

(прізвище, ім’я, по батькові)

1. Тема роботи “Система аналізу та автоматизації розкладів громадського транспорту”

Науковий керівник Пасічник Антон Олексійович, асистент

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “ ” _____ 2026 року № _____

2. Термін подання студентом роботи 04.06.2026

3. Вихідні дані до роботи мова програмування Python, фреймворк Django, СКБД SQLite, середовище розробки PyCharm

4. Перелік питань, які потрібно розробити _____

1) провести аналіз серед наявних аналогів

2) визначити головні функції вебзастосунку, що керує послугами та ресурсами

- 3) аргументувати вибрані інструменти, алгоритми та технології для реалізації вебсистеми
- 4) побудувати архітектуру програмного забезпечення та бази даних
- 5) створити прикладний програмний вебінтерфейс
- 6) представити процес застосування вебінтерфейсу
5. Орієнтований перелік ілюстративного матеріалу схема, що показує роботу фреймворку БД, схема архітектури проєкту, ER-діаграма класів бази даних, скріншоти роботи вебінтерфейсу.
6. Дата видачі завдання 15.09.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи		
2.	Аналіз методів та засобів розв'язання задачі		
3.	Розробка архітектури та загальної структури системи		
4.	Розробка окремих підсистем		
5.	Програмна реалізація системи		
6.	Оформлення пояснювальної записки		
7.	Захист програмного забезпечення		
8.	Передзахист		
9.	Захист		

Студент

(підпис)

Єгор КУДРЯВЦЕВ

(прізвище та ініціали)

Керівник

(підпис)

Антон ПАСІЧНЮК

(прізвище та ініціали)

АНОТАЦІЯ

Робота містить 69 с., 9 іл., 3 табл., 25 джерел.

Мета – розробка системи «FastivBus» для моніторингу громадського транспорту та агрегації розкладів у реальному часі.

Методи: асинхронна архітектура, Python, FastAPI, Vue.js, PostgreSQL, вебскрейпінг.

Результат – створено сервіс автоматичного збору розкладів, валідації геоданих та динамічного трекінгу автобусів на інтерактивній карті.

Ключові слова: FASTIVBUS, МОНІТОРИНГ ТРАНСПОРТУ, АСИНХРОННИЙ СЕРВЕР, ВЕБСКРЕЙПІНГ, РЕАКТИВНА КАРТА.

ABSTRACT

The thesis contains 69 p., 9 ill., 3 tabs, 25 sources.

The purpose is development of the «FastivBus» system for real-time public transport monitoring and schedule aggregation.

Methods: asynchronous architecture, Python, FastAPI, Vue.js, PostgreSQL, web scraping.

Result is a service for automated schedule collection, geodata validation, and dynamic transport tracking on an interactive map was created.

Keywords: FASTIVBUS, TRANSPORT MONITORING, ASYNCHRONOUS SERVER, WEB SCRAPING, REACTIVE MAP.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.....	7
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДЛЯ СИСТЕМИ «FASTIVBUS»	12
1.1 Особливості організації міських та приміських автобусних перевезень	12
1.2 Порівняльний аналіз існуючих геоінформаційних систем та трекерів транспорту.....	14
1.3 Обґрунтування вибору архітектурного підходу та технологічного стеку	16
1.4 Формулювання вимог до системи «FastivBus» та постановка задачі.....	17
2 ПРОЄКТУВАННЯ ТА ОБЧИСЛЮВАЛЬНА АРХІТЕКТУРА СИСТЕМИ	19
2.1 Системна архітектура та концептуальна модель взаємодії	19
2.2 Проектування реляційної моделі бази даних	21
2.3 Моделювання процесів збору та інтеграції зовнішніх розкладів руху ...	22
2.4 Специфікація інтерфейсу програмування додатків (API)	24
2.5 Концепції безперервного моніторингу та логування обчислювальних процесів	25
2.6 Об'єктно-орієнтоване моделювання прецедентів використання системи	27
2.7 Порівняльний аналіз та обґрунтування моделей збереження даних	29
2.8 Моделювання динамічної взаємодії компонентів системи	30
2.9 Теоретичні засади проектування прикладних програмних інтерфейсів .	32
2.10 Аналіз сучасних засобів захисту вебресурсів від автоматизованого збору даних	34

	6
3 МЕТОДИ ТА АЛГОРИТМИ ФУНКЦІОНУВАННЯ СЕРВЕРНОЇ ТА КЛІЄНТСЬКОЇ ЧАСТИН	37
3.1 Концептуальні відмінності архітектур клієнтських вебзастосунків	37
3.2 Теоретичні засади асинхронного керування потоками запитів	40
3.3 Організація стану клієнтського додатку та динамічного рендерингу карт	43
3.4 Низькорівневий аналіз фаз подійного циклу та моделей конкурентності	46
3.5 Методи забезпечення стійкості системи до збоїв при оновленні розкладу	49
3.6 Алгоритми обробки, валідації та інтерполяції геоданих	50
4 ОЦІНКА ЕФЕКТИВНОСТІ, НАВАНТАЖУВАЛЬНЕ ТЕСТУВАННЯ ТА БЕЗПЕКА СИСТЕМИ.....	54
4.1 Методологія тестування архітектурних рішень.....	54
4.2 Моделювання навантаження та аналіз критичних меж швидкодії.....	56
4.3 Аналіз ергономічності та функціональної придатності UI/UX інтерфейсу	58
4.4 Рекомендації щодо розгортання, безпеки та масштабування системи ...	60
4.5 Теоретичний аналіз вразливостей вебзастосунків та методи захисту.....	61
ВИСНОВОК.....	64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	67

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ACID – набір властивостей транзакцій (Atomicity, Consistency, Isolation, Durability), що забезпечують надійність та консистентність роботи бази даних.

API – програмний інтерфейс додатка для взаємодії між різними компонентами системи.

CPU – центральний процесор обчислювального пристрою (Central Processing Unit).

CSS – каскадні таблиці стилів (Cascading Style Sheets) для керування візуальним відображенням та анімацією вебсторінок.

DDoS – тип мережевої атаки, спрямованої на виведення обчислювальної системи з ладу шляхом її штучного перевантаження запитами (Distributed Denial of Service).

DOM – об'єктна модель документа (Document Object Model) для представлення структури та елементів вебсторінки в браузері.

GPS – глобальна супутникова система позиціонування та визначення координат (Global Positioning System).

GTFS – відкритий стандартизований формат для опису та передачі планових розкладів громадського транспорту (General Transit Feed Specification).

GTFS-RT – розширення специфікації GTFS, що забезпечує передачу телеметричних даних транспорту в реальному часі (Realtime).

HTML – стандартна мова розмітки документів (HyperText Markup Language) для створення структури вебсторінок.

HTTP – мережевий протокол передачі гіпертексту (HyperText Transfer Protocol).

HTTPS – захищена версія протоколу HTTP, що підтримує шифрування даних на транспортному рівні.

IP – мережевий протокол (Internet Protocol) та унікальна адреса вузла в комп'ютерній мережі.

JSON – текстовий формат обміну даними (JavaScript Object Notation), що використовується для передачі структурованої інформації між сервером і клієнтом.

REST – архітектурний стиль побудови вебсервісів, що базується на принципах безстатусової взаємодії через стандартні методи HTTP (Representational State Transfer).

RPS – кількість клієнтських запитів, що обробляються сервером за одну секунду (Requests Per Second).

TLS – криптографічний протокол для забезпечення захищеного з'єднання та автентифікації в мережі Інтернет (Transport Layer Security).

UI – візуальний інтерфейс користувача (User Interface).

UX – користувацький досвід та рівень зручності взаємодії людини з інтерфейсом системи (User Experience).

XSS – тип уразливості вебдодатків, що полягає в несанкційованій ін'єкції шкідливого клієнтського скрипту в сторінку (Cross-Site Scripting).

3NF – третя нормальна форма організації таблиць реляційних баз даних, що мінімізує надлишковість та аномалії інформації.

ГІС – геоінформаційна система, призначена для збору, збереження, аналізу та графічної візуалізації просторово-часових даних.

ШІ – штучний інтелект (використовується в контексті перспективного аналізу пасажиропотоків та прогнозування часу прибуття рейсів).

ВСТУП

Сучасний розвиток міських агломерацій та інфраструктурних систем характеризується стрімкою цифровізацією та впровадженням концепції розумного міста (Smart City). Одним із ключових елементів цієї концепції є оптимізація роботи громадського транспорту, що безпосередньо впливає на якість життя населення, економічну активність та екологічну ситуацію в регіоні. Ефективне управління пасажирськими перевезеннями вимагає високого рівня інформаційної прозорості – забезпечення пасажирів точними, актуальними та надійними даними про розклад руху й поточне просторове положення транспортних засобів у реальному часі. Вирішення цього завдання покладається на спеціалізовані веборієнтовані геоінформаційні сервісів, здатні ефективно обробляти великі потоки динамічних даних.

Актуальність теми. Зумовлена наявною інфраструктурною проблематикою середніх та малих міст України, зокрема міста Фастів, які виконують функцію важливих регіональних транспортно-логістичних вузлів. Транспортна модель таких міст має виражену специфіку: внутрішньоміські автобусні маршрути тісно пов'язані з розкладом курсування приміського залізничного транспорту. Відсутність єдиної цифрової платформи, яка б об'єднувала статичні розклади та динамічний трекінг, призводить до значних часових втрат пасажирів та дезорганізації пасажиропотоків у пікові години. Крім того, більшість існуючих глобальних та національних навігаційних сервісів ігнорують специфіку малих регіонів через низьку комерційну привабливість або вимагають від перевізників значних фінансових і технічних витрат на розгортання важкої інфраструктури збору даних.

Для вирішення зазначених проблем у рамках роботи розробляється архітектура та теоретичне підґрунтя інформаційної системи реального часу «FastivBus». Головною особливістю проєктованого рішення є створення гнучкого, полегшеного та відмовостійкого програмного комплексу, який в автоматичному режимі акумулює інформацію про розклади руху з гетерогенних зовнішніх вебресурсів за допомогою технологій вебскрейпінгу, обробляє її в асинхронному

режимі та транслює кінцевим користувачам через синхронізований із бд вебінтерфейс. Такий підхід дозволяє розгорнути повноцінну систему моніторингу громадського транспорту без необхідності впровадження складних промислових стандартів на боці автопідприємств, що робить її економічно ефективною та адаптованою до реальних умов функціонування регіонального транспортного сектору.

Об'єкт розробки. Процеси автоматизованого моніторингу пасажирських перевезень, збору та інтеграції розкладів громадського транспорту з децентралізованих зовнішніх джерел, а також оперативного інформування пасажирів міського та приміського сполучення в умовах динамічної зміни дорожньої ситуації.

Предмет розробки. Моделі, методи, алгоритми та обчислювальна архітектура програмного забезпечення асинхронної серверної частини та реактивного клієнтського інтерфейсу інформаційної системи «FastivBus», що функціонує в режимі реального часу і забезпечує обробку та візуалізацію просторово-часових даних без блокування обчислювальних потоків.

Мета роботи. Теоретичне проектування, алгоритмізація та системне моделювання відмовостійкої клієнт-серверної інформаційної системи «FastivBus», яка забезпечує автоматизоване агрегування розкладів руху, фільтрацію та інтерполяцію геоданих, а також надання користувачам інтерактивного інструментарію для відстеження громадського транспорту міста Фастів у реальному часі.

Для досягнення поставленої мети в роботі сформульовано та вирішено такі завдання: проаналізовано існуючі геоінформаційні трекери та визначено їхні обмеження; розроблено концептуальну й логічну архітектуру системи «FastivBus» на основі принципу повного розділення рівнів бізнес-логіки та представлення даних; спроектовано нормалізовану реляційну модель бази даних; змодельовано конвеєр фонового збору розкладів руху з обходом захисних механізмів вебресурсів; розроблено алгоритми кінематичної валідації та прив'язки координатного потоку до цифрового графу доріг; спроектовано асинхронні методи керування потоками мережових запитів; створено специфікацію RESTful API,

проведено навантажувальне тестування серверних вузлів та аналіз ергономічності користувацького інтерфейсу.

Методи розробки. Базуються на положеннях системного аналізу, теорії проектування інформаційних систем та реляційних баз даних. Для розробки архітектурних рішень використано концепції подійно-орієнтованого програмування (Event-Driven Architecture) та патерни асинхронного керування потоками введення-виведення. Моделювання алгоритмів обробки геоданих спирається на методи обчислювальної геометрії, математичної інтерполяції та фільтрації сигналів. Експериментальні дослідження швидкодії виконано на основі методів імітаційного моделювання навантаження та статистичного аналізу часових метрик відгуку обчислювальних вузлів.

Практичне значення отриманих результатів. Полягає у створенні цілісної архітектурної концепції транспортного інформера, адаптованого до специфіки середніх міст, яка поєднує високу обчислювальну ефективність асинхронного сервера з гнучкими модулями автономного збору даних у фоновому режимі. Розроблені методи забезпечення відмовостійкості (інтеграція патерну «Запобіжник» та алгоритмів експоненціального відкладення запитів) дозволяють системі зберігати стабільність при структурних змінах на цільових сторонніх вебресурсах. Створений технологічний каркас може бути безпосередньо використаний виконавчим комітетом Фастівської міської ради, комунальними та приватними перевізниками для модернізації інфраструктури та підвищення рівня задоволеності пасажирів.

Структура та обсяг роботи. Пояснювальна записка дипломної роботи складається зі вступу, чотирьох розділів, висновків, переліку умовних позначень, скорочень і термінів, а також переліку джерел посилання. Повний обсяг текстової частини пояснювальної записки становить 65 сторінок комп'ютерного тексту. Робота містить ілюстративний матеріал у вигляді рисунків, таблиць, а також перелік використаних джерел посилання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДЛЯ СИСТЕМИ «FASTIVBUS»

Перший розділ присвячено комплексному аналізу предметної області, дослідженню специфіки організації пасажирських перевезень та науково-технічному обґрунтуванню вимог до проектування системи «FastivBus». У його межах детально розглянуто особливості функціонування міського та приміського автобусного сполучення в умовах транспортної моделі міста Фастів, а також проведено порівняльний аналіз наявних геоінформаційних сервісів і навігаційних трекерів з метою виявлення їхніх системних, технічних та економічних обмежень при масштабуванні в межах малих регіонів.

Особливу увагу приділено теоретичним засадам проектування прикладних програмних інтерфейсів, аналітичному порівнянню архітектурних підходів та обґрунтуванню вибору асинхронного технологічного стеку розробки. Крім того, досліджено сучасні засоби захисту зовнішніх вебресурсів від автоматизованого збору даних, що дозволило сформулювати суворі функціональні й нефункціональні вимоги та здійснити точну постановку технічної задачі для розробки відмовостійкого програмного комплексу.

1.1 Особливості організації міських та приміських автобусних перевезень

Сучасний стан розвитку міської інфраструктури та пасажирських перевезень в Україні вимагає впровадження новітніх інформаційних технологій для підвищення якості обслуговування населення. Проблема ефективного управління громадським транспортом є особливо гострою для середніх та малих міст, таких як Фастів, які виконують функцію важливих транспортних вузлів. Специфіка пасажиропотоків у таких регіонах зумовлена значною кількістю приміських маршрутів, що сполучають районний центр із навколишніми населеними

пунктами, а також високою залежністю розкладу руху від графіків курсування залізничного транспорту. Це створює додаткові вимоги до гнучкості та точності координації автобусних рейсів, оскільки пасажери потребують оперативного та актуального інформування про наявність та час прибуття транспортних засобів.

Організація міських та приміських автобусних перевезень супроводжується низкою дестабілізуючих факторів, які впливають на дотримання затверджених графіків руху. До таких факторів належать коливання інтенсивності дорожнього руху в години пік, технічні несправності рухомого складу, незадовільний стан дорожнього покриття, а також несприятливі погодні умови. Традиційні методи інформування пасажирів, які полягають у розміщенні статичних розкладів руху на зупинках громадського транспорту, за умов динамічних змін стають неефективними та неактуальними. Відсутність зворотного зв'язку між перевізниками та пасажиром призводить до значних втрат часу громадянами, зниження рівня довіри до комунального та приватного транспорту, а також до нераціонального розподілу пасажиропотоків на альтернативні види переміщення.

Важливою особливістю функціонування транспортної системи міста Фастів є поєднання внутрішньоміських маршрутів, які мають відносно високу частоту курсування, та приміських рейсів, інтервали між якими можуть становити від кількох десятків хвилин до кількох годин. Для приміського сполучення затримка або скасування рейсу без завчасного попередження є критичною проблемою, оскільки пасажери втрачають можливість вчасно дістатися до місця роботи, навчання або соціальних об'єктів. Таким чином, виникає об'єктивна потреба у створенні єдиного інформаційного простору, який дозволить акумулювати дані про поточний стан виконання рейсів та надавати їх кінцевим користувачам у зручному, доступному та наочному вигляді.

Впровадження автоматизованих систем моніторингу та інформування дозволяє частково вирішити проблеми диспетчеризації перевезень та мінімізувати негативний вплив людського фактора. Проте, наявні системи автоматизації часто орієнтовані виключно на внутрішні потреби транспортних підприємств, залишаючи пасажирів поза процесом інформаційного обміну. Розробка інтегрованої системи «FastivBus» покликана ліквідувати цей інформаційний розрив

шляхом надання пасажиром інструментів для планування поїздки на основі актуальних розкладів та даних відстеження. Це дозволить оптимізувати використання часу пасажиром, підвищити прозорість роботи транспортних операторів та закласти підґрунтя для подальшого розвитку концепції розумного міста на рівні регіону.

1.2 Порівняльний аналіз існуючих геоінформаційних систем та трекерів транспорту

На сучасному ринку інформаційних технологій існує значна кількість рішень, призначених для моніторингу громадського транспорту та інформування пасажирів. Для визначення оптимальної стратегії проектування системи «FastivBus» необхідно провести детальний аналіз існуючих геоінформаційних систем та трекерів транспорту, оцінити їхні переваги, недоліки та архітектурні обмеження при масштабуванні в межах невеликих міст. Основними критеріями оцінки виступають рівень локалізації, точність надання інформації, гнучкість інтеграції із зовнішніми джерелами даних, вартість впровадження та підтримки, а також доступність для кінцевого користувача.

Глобальні картографічні сервіси, такі як Google Maps, володіють потужною інфраструктурою та забезпечують високу якість візуалізації картографічних даних. Вони підтримують функцію побудови мультимодальних маршрутів та відображення розкладів громадського транспорту. Проте, інтеграція динамічних даних про рух транспорту в реальному часі у таких системах вимагає від місцевих перевізників підтримки складного та дорогого стандарту GTFS-RT (General Transit Feed Specification Realtime). Для невеликих приватних та комунальних автопідприємств Фастова розгортання та адміністрування такої інфраструктури є економічно недоцільним та технічно складним завданням, що робить використання глобальних сервісів як єдиного рішення неможливим.

Спеціалізовані національні та регіональні системи, такі як EasyWay або CityBus, орієнтовані на український ринок пасажирських перевезень і успішно функціонують у багатьох великих містах. Вони пропонують користувачам зручні інтерфейси для відстеження транспорту на основі GPS-даних, отриманих безпосередньо з трекерів, встановлених на автобусах. Основним недоліком таких платформ є їхня висока комерціалізація та орієнтація на великі мегаполіси з гарантованим високим рівнем монетизації. У межах малих міст оновлення інформації про маршрути та розклади в таких системах часто відбувається із запізненням, а інтерфейси перевантажені сторонньою рекламою, що погіршує користувацький досвід та знижує надійність системи.

Для більш наочного порівняння характеристик існуючих рішень та обґрунтування розробки спеціалізованої системи «FastivBus» доцільно систематизувати дані аналізу. У таблиці 1.1 наведено порівняльну характеристику базових параметрів аналізованих систем та запроєктованого рішення.

Таблиця 1.1 – Порівняльний аналіз систем моніторингу транспорту

№	Назва системи	Рівень локалізації	Підтримка кастомних розкладів	Інтеграція зі скрейпінгом джерел	Специфіка архітектури
1	Google Maps	Глобальний	Низька (лише GTFS)	Відсутня	Закрита хмарна
2	EasyWay	Національний	Середня (через запити)	Часткова	Клієнт-серверна з рекламою
3	CityBus	Регіональний	Середня	Відсутня	Мобільно-орієнтована
4	FastivBus	Локальний (високий)	Висока (пряма)	Повна (автоматизована)	Асинхронна веборієнтована

Проведений аналіз показує, що жодне з існуючих рішень повною мірою не враховує специфіку транспортної моделі міста Фастів, яка потребує гнучкої інтеграції як зі статичними розкладами приміського сполучення, так і з динамічними даними. Більшість систем не мають вбудованих механізмів автоматизованого збору інформації (скрейпінгу) з офіційних сайтів автостанцій або залізниці, що змушує адміністраторів вносити зміни вручну. Розробка власної системи «FastivBus» дозволить усунути ці обмеження завдяки створенню спеціалізованих модулів збору даних та гнучкого налаштування під потреби конкретної локальної громади.

1.3 Обґрунтування вибору архітектурного підходу та технологічного стеку

Вибір архітектурного підходу є визначальним фактором для забезпечення стабільності, швидкодії та масштабованості інформаційної системи, що працює з даними у реальному часі. При проектуванні системи «FastivBus» було розглянуто кілька архітектурних патернів, зокрема класичний монолітний підхід та дворівневу клієнт-серверну архітектуру із чітким розділенням рівнів представлення та бізнес-логіки. Враховуючи теоретичну спрямованість роботи, особливу увагу приділено аналізу ефективності обробки асинхронних подій та мережевих запитів, оскільки система має одночасно виконувати операції збору даних із зовнішніх джерел та обслуговувати підключення великої кількості кінцевих користувачів.

Монолітна архітектура, де компоненти збору даних, обробки інформації та рендерингу інтерфейсу об'єднані в один виконуваний блок, має суттєві недоліки для систем реального часу. Будь-яке тривале блокуюче завдання, наприклад, виконання важкого HTTP-запиту до стороннього сайту розкладу під час скрейпінгу, може призвести до затримок у відповідях користувачам або до зависання всього вебдодатку. Тому для системи «FastivBus» обрано архітектуру розділеного клієнт-серверного додатку (Decoupled Architecture) з асинхронним

сервером обробки даних та реактивним клієнтським вебінтерфейсом, що взаємодіють через оптимізоване інтерфейсне середовище API.

Для реалізації серверної частини обґрунтовано використання асинхронного підходу на основі подійно-орієнтованої моделі (Event-Driven Architecture) з використанням мови програмування JavaScript (середовище Node.js). Теоретична перевага цієї моделі полягає в концепції невиконуваного блокування операцій вводу-виводу (Non-blocking I/O). Замість виділення окремого обчислювального потоку для кожного користувача, що швидко вичерпує ресурси сервера при високому навантаженні, асинхронний сервер використовує єдиний потік обробки подій (Event Loop). Це дозволяє ефективно обробляти тисячі одночасних з'єднань від клієнтів, які очікують на оновлення геоданих автобусів, мінімізуючи витрати оперативної пам'яті та процесорного часу.

Для створення клієнтської частини системи обрано компонентно-орієнтований фреймворк Vue.js. Головною теоретичною засадою цього вибору є концепція реактивності даних та використання віртуального дерева елементів сторінки (Virtual DOM). У системах відстеження транспорту дані про координати маркерів на карті оновлюються дуже часто. Традиційний підхід із повним перемальовуванням елементів інтерфейсу викликає значне навантаження на браузер користувача, що призводить до видимих затримок та швидкого розряджання мобільних пристроїв. Vue.js дозволяє відстежувати зміни у стані даних сервера та оновлювати лише ті конкретні компоненти інтерфейсу, які зазнали змін, забезпечуючи високу плавність відображення картографічних матеріалів та інформаційних табло розкладу.

1.4 Формулювання вимог до системи «FastivBus» та постановка задачі

Формулювання чітких вимог є критично важливим етапом проектування, оскільки вони визначають межі системи та критерії оцінки її працездатності.

Функціональні вимоги до системи «FastivBus» охоплюють три основні напрямки: взаємодію з пасажирями, автоматизований збір даних та адміністрування. Для кінцевого користувача система повинна забезпечувати можливість інтерактивного перегляду актуального розкладу руху міських та приміських автобусів із прив'язкою до днів тижня та типів маршрутів, відображення поточного просторового положення транспортних засобів на цифровій карті міста, а також пошук оптимальних шляхів сполучення між заданими зупинками.

Модуль автоматизованого збору даних повинен у фоновому режимі виконувати періодичні аналітичні запити до зовнішніх інформаційних ресурсів (вебсайтів автостанцій та залізниць), здійснювати синтаксичний аналіз отриманого текстового контенту, виділяти корисну інформацію про час відправлення та прибуття рейсів, ідентифікувати можливі зміни чи скасування поїздок та передавати ці структуровані дані до внутрішнього сховища. З боку адміністратора система має надавати теоретичні інструменти для ручного коригування маршрутної сітки, додавання нових перевізників, керування географічними координатами зупинок та моніторингу загального стану підсистем.

На основі проведеного аналізу предметної області, існуючих аналогів, архітектурних підходів та сформованих вимог, постановка задачі дипломної роботи полягає у теоретичному проектуванні, моделюванні та розробці архітектури інформаційної системи реального часу «FastivBus». Дана система призначена для автоматизованого моніторингу громадського транспорту, інтеграції розкладів руху з гетерогенних зовнішніх джерел та оперативного інформування пасажирів міського та приміського сполучення міста Фастів. Реалізація цієї задачі вимагає розробки концептуальних моделей баз даних, алгоритмів обробки геоданих, проектування асинхронних методів взаємодії та оцінки загальної ефективності запропонованих архітектурних рішень.

2 ПРОЄКТУВАННЯ ТА ОБЧИСЛЮВАЛЬНА АРХІТЕКТУРА СИСТЕМИ

Другий розділ спрямований на концептуальне проектування архітектурного каркаса, інформаційної топології та обчислювальної структури системи «FastivBus». У його межах формалізовано трирівневу модель взаємодії функціональних шарів та розроблено нормалізовану реляційну модель бази даних, яка забезпечує цілісність, транзакційність та надійне збереження інфраструктурної й телеметричної інформації. Детально змодельовано процеси фонового збору даних, автоматизованої інтеграції зовнішніх розкладів руху та специфікацію безстатусового інтерфейсу програмування додатків (API).

Додатково в розділі представлено об'єктно-орієнтоване моделювання прецедентів використання системи за допомогою уніфікованої мови UML , а також наведено порівняльний аналіз моделей збереження даних, який теоретично доводить доцільність оптимізації початкового стеку та переходу до асинхронних інструментів взаємодії. Завершується розділ побудовою моделей динамічної взаємодії компонентів системи у вигляді діаграм послідовностей, що відображають хронологію обміну повідомленнями під час потокового відстеження транспорту у реальному часі.

2.1 Системна архітектура та концептуальна модель взаємодії

Проектування архітектури інформаційної системи «FastivBus» базується на принципі декуплінгу (розділення), що передбачає повну незалежність функціональних шарів обробки даних, їхнього збереження та візуалізації. Такий підхід дозволяє створити гнучку топологію системи, де кожен компонент виконує чітко визначену роль і взаємодіє з іншими через стандартизовані інтерфейси. Концептуальна модель системи розглядається як трирівнева структура, що включає рівень збору та агрегації гетерогенних даних із зовнішніх джерел, центральний

серверний рівень керування бізнес-логікою та клієнтський рівень представлення інформації для кінцевих користувачів. Основною теоретичною перевагою такої децентралізації є ізоляція обчислювальних збоїв, коли нестабільна робота стороннього ресурсу під час збору даних не впливає на доступність вебінтерфейсу для пасажирів.

Центральне місце у концептуальній моделі взаємодії посідає серверний процесорний вузол, який координує потоки інформації у реальному часі. Модуль збору даних працює у фоновому режимі та ініціює асинхронні запити до зовнішніх серверів транспортних операторів та залізничних станцій. Після отримання сирого текстового контенту здійснюється його первинна фільтрація, деструктуризація та трансформація у внутрішній уніфікований формат системи. Очищена та структурована інформація надходить до реляційного сховища даних, яке виступає єдиним джерелом істини для системи. Клієнтська частина додатку не має прямого доступу до бази даних або зовнішніх джерел, а взаємодіє виключно з сервером через захищений та оптимізований прошарок інтерфейсу програмування додатків.

Особлива увага в архітектурній моделі приділяється обробці динамічних подій, які генеруються модулями телеметрії громадського транспорту. Потоківі дані про просторове положення автобусів надходять на сервер з високою частотою, що вимагає специфічної організації черг обробки повідомлень. Серверна архітектура спроектована таким чином, щоб мінімізувати накладні витрати на десеріалізацію пакетів даних та забезпечити їхнє миттєве передавання на рівень представлення. Завдяки використанню подійного циклу (Event Loop) на сервері, процеси читання та запису не блокують один одного, що дозволяє підтримувати консистентність даних навіть за умов критичного зростання кількості одночасних сесій користувачів.

Клієнтський рівень взаємодії є повністю автономним вебдодатком, який завантажується в браузер користувача і самостійно керує власним станом. Отримавши первинну структуру маршрутів та координати зупинок, клієнтський додаток ініціює періодичні полегшені запити для оновлення маркерів транспорту на карті. Такий розподіл обчислювального навантаження дозволяє перенести процеси рендерингу графічних елементів безпосередньо на пристрій кінцевого

користувача. Це суттєво розвантажує центральний сервер системи та знижує вимоги до пропускної здатності каналів зв'язку, що є критично важливим для мобільного інтернету в умовах приміського сполучення.

2.2 Проектування реляційної моделі бази даних

Для забезпечення надійного збереження, швидкого пошуку та цілісності інформації про транспортну інфраструктуру міста Фастів у системі «FastivBus» запроектовано реляційну модель бази даних. Вибір реляційного підходу обґрунтований необхідністю чіткого дотримання зв'язків між сутностями, підтримки транзакційності на рівні ACID та забезпечення декларативної цілісності даних за допомогою зовнішніх ключів. Структура сховища розроблена із застосуванням принципів нормалізації до третьої нормальної форми включно, що дозволяє усунути надлишковість даних, мінімізувати ризики виникнення аномалій модифікації, видалення чи вставки, а також оптимізувати використання дискового простору сервера.

Базова концептуальна схема сховища даних складається з кількох ключових взаємопов'язаних сутностей, які описують як статичні, так і динамічні аспекти транспортної системи. Головною інфраструктурною сутністю є модель перевізників, яка містить юридичні та контактні дані транспортних підприємств. З цією сутністю пов'язана модель маршрутів, що описує логічні напрямки руху громадського транспорту, включаючи їхні номери, назви, типи сполучення та приналежність до конкретного оператора. Оскільки один перевізник може обслуговувати багато маршрутів, між ними встановлюється класичний зв'язок один-до-багатьох, який гарантує коректну фільтрацію даних на рівні користувацьких запитів.

Географічна складова системи описується сутністю зупинок, яка фіксує унікальні ідентифікатори, текстові назви та просторові координати (широту і довготу) місць посадки та висадки пасажирів. Оскільки маршрут громадського транспорту являє собою впорядковану послідовність зупинок, а одна й та сама

зупинка може входити до складу різних маршрутів, для реалізації цього зв'язку проектується проміжна сутність траєкторії маршруту. Ця таблиця зв'язку багато-до-багатьох містить зовнішні ключі маршруту та зупинки, а також обов'язкове поле порядкового номера, яке визначає точну послідовність проходження пунктів транспортним засобом, що дозволяє теоретично моделювати правильний напрямок руху при побудові графіків.

Часова та динамічна складові моделі даних реалізовані через сутності рейсів та поточного трекінгу. Сутність рейсів містить інформацію про плановий час відправлення автобуса з початкової точки, дні тижня, в які виконується даний рейс, та посилання на конкретний маршрут. Для фіксації динамічних координат у реальному часі проектується окрема високооптимізована таблиця телеметрії. Вона пов'язана з активним рейсом і містить часову мітку, поточні координати транспортного засобу та його швидкість. Для запобігання переповненню сховища та зниженню швидкодії через великі обсяги динамічних даних, у моделі передбачено механізм архівації застарілих запитів телеметрії та індексацію за полями часу і маршруту, що забезпечує стабільно низький час виконання пошукових операцій.

2.3 Моделювання процесів збору та інтеграції зовнішніх розкладів руху

Функціонування системи «FastivBus» значною мірою залежить від повноти та актуальності інформації, яка часто розміщена на децентралізованих, гетерогенних вебресурсах сторонніх організацій, таких як офіційні сайти автоперевізників, регіональних автостанцій або залізничного вузла. Оскільки більшість цих платформ не надають відкритих, документованих інтерфейсів програмування додатків для прямого отримання структурованих даних, у роботі змодельовано теоретичний процес автоматизованого збору інформації за допомогою технології вебскрейпінгу. Процес інтеграції розкладів розглядається як

послідовний конвеєр обробки даних, що включає етапи планування запитів, завантаження контенту, синтаксичного аналізу, семантичної валідації та синхронізації з внутрішньою базою даних.

Теоретична модель модуля скрейпінгу базується на концепції безблокувального виконання мережевих операцій за розкладом. Спеціалізований планувальник завдань ініціює сесії збору даних у періоди найменшого навантаження на систему, наприклад, у нічний час, щоб уникнути конфліктів доступу до сховища даних та знизити навантаження на цільові сторонні сервери. Під час сесії модуль надсилає асинхронні HTTP-запити до віддалених вебсторінок. Головною проблемою на цьому етапі є необхідність обходу сучасних захисних механізмів вебресурсів, таких як фільтрація за IP-адресами, обмеження частоти запитів та динамічна генерація вмісту сторінки за допомогою клієнтських сценаріїв JavaScript. Для вирішення цих проблем моделюється логіка ротації ідентифікаторів клієнта, інтелектуальних затримок між зверненнями та використання безголових браузерних емуляторів, які дозволяють виконувати скрипти на цільовій сторінці та отримувати повністю сформоване дерево елементів документа.

Наступним етапом моделювання є синтаксичний аналіз (парсинг) отриманої структури документа. Модуль здійснює пошук специфічних текстових маркерів та тегів, які містять корисне навантаження – номери рейсів, станції призначення, час відправлення та прибуття. Оскільки структура сторонніх сайтів може періодично змінюватися, теоретична модель парсера проектується із застосуванням шаблону проектування «Стратегія». Це дозволяє ізолювати правила розбору конкретного сайту в окремі конфігураційні класи, забезпечуючи легку адаптацію системи до змін зовнішнього інтерфейсу без необхідності перебудови всієї архітектури модуля збору даних.

Завершальним кроком інтеграційного процесу є семантичний аналіз та синхронізація. Виділені текстові дані проходять процедуру нормалізації: часові мітки переводяться в єдиний часовий пояс, назви зупинок очищаються від орфографічних помилок та приводяться до системного довідника за допомогою алгоритмів порівняння рядків. Після цього виконується ідемпотентна операція запису в базу даних, яка порівнює отриманий розклад із вже існуючим. Якщо

виявляються розбіжності, система автоматично оновлює планові рейси, маркує скасовані маршрути та генерує внутрішні сповіщення для підсистеми користувацького інтерфейсу, гарантуючи високу точність та актуальність інформаційного поля системи.

2.4 Специфікація інтерфейсу програмування додатків (API)

Для забезпечення взаємодії між реактивним клієнтським додатком та асинхронною серверною частиною системи «FastivBus» розроблено специфікацію інтерфейсу програмування додатків, що базується на архітектурному стилі REST. Вибір RESTful-архітектури зумовлений її відповідністю принципам слабкої пов'язаності компонентів, масштабованості, уніфікованості інтерфейсів та підтримок кешування відповідей на рівні мережових протоколів. Інтерфейс проектується як сукупність безстатусових (stateless) кінцевих точок, де кожен запит від клієнта містить у собі всю інформацію, необхідну для його успішної обробки сервером, що усуває потребу збереження стану сесії в оперативній пам'яті серверного вузла.

Організація кінцевих точок API побудована навколо основних бізнес-сутностей транспортної системи та використовує стандартні методи протоколу HTTP для визначення характеру операцій. Для отримання загального переліку доступних автобусних маршрутів міста та приміської зони проектується точка доступу, яка повертає масив об'єктів з базовою інформацією. Додаткові параметри фільтрації, такі як тип сполучення або ідентифікатор перевізника, передаються як змінні рядка запиту, що дозволяє гнучко обмежувати обсяг вибірки на рівні бази даних. Детальна інформація про конкретний маршрут, включаючи повну географічну траєкторію та впорядкований список зупинок, стає доступною за умови передавання унікального ідентифікатора маршруту безпосередньо у структурі шляху запиту.

Окремий сегмент специфікації API виділено під обслуговування запитів розкладу руху та динамічної телеметрії, які є найбільш критичними до швидкодії

системи. Запит на отримання планового розкладу для конкретної зупинки або маршруту приймає параметри поточної дати та часу, повертаючи упорядкований за часом масив найближчих рейсів. Для передавання поточних координат автобусів, що перебувають на лінії, проектується спеціалізована кінцева точка, яка повертає географічні мітки лише для активних у дану хвилину транспортних засобів. Як стандартний формат серіалізації даних для всіх відповідей та запитів обрано JSON через його мінімальну текстову надлишковість, високу швидкість парсингу сучасними веббраузерами та повну сумісність із обраним технологічним стеком розробки.

Для захисту інтерфейсу програмування від зловживань, перевантаження та несанкціонованого доступу у специфікації передбачено теоретичні механізми контролю доступу та обмеження частоти запитів (Rate Limiting). Кожен вхідний запит аналізується сервером на предмет унікального ідентифікатора клієнта або IP-адреси, і у випадку перевищення встановленого ліміту звернень на одиницю часу, сервер повертає відповідний статус помилки без виконання важких операцій пошуку в базі даних. Таке архітектурне рішення гарантує стабільність роботи інтерфейсу програмування та забезпечує рівномірний розподіл ресурсів сервера між усіма користувачами транспортної системи.

2.5 Концепції безперервного моніторингу та логування обчислювальних процесів

Стабільне функціонування транспортних інформерів реального часу в умовах безперервного виробничого середовища (Production) вимагає впровадження розвинених методологій спостережуваності (Observability). Після практичного розгортання системи «FastivBus» виникає потреба в постійному контролі її внутрішнього стану, виявленні прихованих обчислювальних аномалій, оптимізації розподілу ресурсів та миттєвому реагуванні на технічні збої. Теоретичною основою для вирішення цього завдання є проектування підсистем

централізованого логування (журналювання) та метричного моніторингу, які дозволяють трансформувати сирі системні події в наочні аналітичні матеріали для адміністраторів та інженерів підтримки.

Підсистема логування базується на концепції структурованого запису подій, де кожне повідомлення про роботу фонових парсерів або API-сервера містить часову мітку, ідентифікатор компонента та суворий рівень критичності. Моделюється чотири класичних рівні журналювання подій. Рівень дебагу (DEBUG) використовується на етапі розробки для детального трасування обчислювальних кроків. Рівень інформування (INFO) фіксує штатні системні операції, наприклад, успішне завершення циклу скрейпінгу сайту залізничного вокзалу Фастова. Рівень попередження (WARN) маркує некритичні відхилення від норми, зокрема тимчасові затримки відповіді зовнішнього сервера, які були успішно нівельовані алгоритмом експоненціального відкладення. Рівень помилки (ERROR) сигналізує про критичні збої, такі як повна зміна HTML-структури стороннього ресурсу, що вимагає негайного втручання розробників для перебудови логіки парсера.

Метричний моніторинг орієнтований на збір та аналіз кількісних параметрів функціонування системи у часових рядах. Основними об'єктами спостереження виступають як системні інфраструктурні показники (завантаження ядер центрального процесора, обсяг вільної оперативної пам'яті сервера, інтенсивність дискових операцій), так і специфічні прикладні метрики. До прикладних метрик належать кількість активних підключень користувачів до карти, частота генерації запитів до PostgreSQL, швидкість обробки телеметричних пакетів та латентність API. Особлива увага приділяється моніторингу стану подійного циклу (Event Loop) асинхронного сервера Node.js, оскільки поява блокуючих операцій миттєво викликає затримку обробки черги (Event Loop Delay), що є головним індикатором деградації швидкодії сервісу реального часу.

Для автоматизації процесів підтримки стабільності моделюється архітектурна логіка циклів перевірки працездатності (Health Check Loops). Контейнеризоване середовище розгортання системи періодично надсилає легкі тестові запити до спеціалізованих внутрішніх кінцевих точок API. Якщо сервер або

база даних не повертають успішну відповідь протягом встановленого часового ліміту, інфраструктурний менеджер інтерпретує це як критичне зависання обчислювального процесу, автоматично ізолює пошкоджений контейнер від загального трафіку та запускає його перезапуск (механізм Self-Healing). Такий підхід гарантує безперервну доступність транспортного сервісу для пасажирів з коефіцієнтом відмовостійкості на рівні високих індустріальних стандартів, мінімізуючи вплив технічних збоїв на якість надання інформаційних послуг.

2.6 Об'єктно-орієнтоване моделювання прецедентів використання системи

Розробка архітектури сучасних геоінформаційних сервісів реального часу потребує чіткого формалізованого опису функціональних вимог та сценаріїв взаємодії користувачів із програмним комплексом. Найбільш ефективним методологічним інструментом для вирішення цього завдання є об'єктно-орієнтований аналіз із застосуванням уніфікованої мови моделювання UML, зокрема проектування діаграм прецедентів використання (Use Case Diagrams).

Моделювання прецедентів дозволяє абстрагуватися від конкретної програмної реалізації та зосередити увагу на зовнішніх границях системи, визначенні ключових дійових осіб (акторів) та їхніх функціональних можливостей. Для системи «FastivBus» це дослідження є критично важливим, оскільки вона обслуговує різноманітні категорії користувачів, кожна з яких має унікальні паттерни поведінки та вимоги до інформаційного середовища.

У запроектованій структурі системи виділяються три основні автономні актори, які взаємодіють із програмним комплексом через відповідні інтерфейсні рівні. Першим і наймасовішим актором є Пасажир (кінцевий користувач), який представляє інтереси жителів та гостей міста Фастів, що користуються громадським транспортом. Його взаємодія з системою є суто пасивною з точки зору модифікації даних, але критичною щодо швидкодії та актуальності отримання

інформації. Другим актором виступає Адміністратор системи – привілейований користувач, на якого покладено функції оперативного керування маршрутною сіткою, моніторингу працездатності серверних вузлів та валідації інтегрованих даних. Третім, системним актором є Зовнішнє джерело даних (модуль телеметрії та сторонні вебресурси), який в автоматичному режимі постачає потокові координати автобусів та оновлення статичних розкладів руху.

Для актора Пасажир основним прецедентом використання є перегляд інтерактивної карти руху транспорту в реальному часі. Цей прецедент знаходиться у зв'язку розширення (extend) з прецедентами фільтрації автобусів за конкретними номерами маршрутів та відображення детальної інформації про обраний транспортний засіб. Іншим базовим сценарієм для пасажирів є перегляд статичного розкладу руху по конкретній зупинці, який включає автоматичне визначення найближчих рейсів на основі поточного системного часу. Завдяки концепції реактивності клієнтського інтерфейсу, всі ці прецеденти виконуються в межах однієї сесії Single Page Application без перезавантаження сторінки, що мінімізує час очікування відповіді від сервера.

З боку Адміністратора системи моделюється комплекс прецедентів управління контентом та інфраструктурою. Сюди належать сценарії додавання, редагування та видалення елементів маршрутної мережі, коригування географічного положення зупинок на карті, а також ручне внесення змін до графіків руху у випадку тривалого скасування рейсів. Прецедент моніторингу скрейперів дозволяє адміністратору візуально оцінювати статус підключення до сторонніх серверів автостанцій та виявляти можливі помилки синтаксичного аналізу зовнішніх сайтів. Взаємодія між прецедентами адміністратора та внутрішньою базою даних реалізується через суворі механізми автентифікації та авторизації, що гарантує захист системи від несанкціонованого втручання та збереження консистентності сховища даних.

2.7 Порівняльний аналіз та обґрунтування моделей збереження даних

Важливим етапом проектування обчислювальної архітектури системи «FastivBus» є науково обґрунтований вибір інструментів збереження даних та методів взаємодії з ними. У попередніх технічних специфікаціях та вихідних завданнях як базовий стек часто розглядається використання класичного синхронного фреймворку Django зі вбудованою системою керування базами даних SQLite. Проте, для високонавантажених геоінформаційних сервісів, що працюють з потоковими даними у реальному часі, такий підхід має суттєві архітектурні обмеження, які можуть призвести до деградації швидкодії системи в моменти пікових навантажень. Проведення детального порівняльного аналізу між традиційними синхронними моделями та сучасними асинхронними інструментами (такими як SQLAlchemy в поєднанні з PostgreSQL) дозволяє теоретично довести доцільність оптимізації стеку розробки.

Обмеження СКБД SQLite полягають у її внутрішній архітектурі, оскільки вона є вбудованою (file-based) базою даних, орієнтованою на локальне збереження контенту. Головним недоліком SQLite за умов високої конкурентності є механізм блокування на рівні всього файлу бази даних при виконанні операцій запису. Коли модуль телеметрії системи «FastivBus» намагається з високою частотою оновлювати координати десятків автобусів, кожна операція запису блокує базу даних для всіх інших транзакцій. У цей же момент тисячі пасажирів, які намагаються зчитати розклад або отримати поточні маркери транспорту, змушені очікувати в черзі на розблокування файлу, що викликає стрімке зростання часу відгуку системи та призводить до критичних помилок тайм-ауту.

Синхронна модель взаємодії, притаманна традиційному використанню Django ORM, посилює цю проблему через специфіку блокування обчислювальних потоків. При виконанні запиту до бази даних весь потік виконання сервера (thread) переходить у стан очікування відповіді від дискової підсистеми, утилізуючи ресурси процесора на бездіяльне очікування (thread starvation). Для обслуговування великої кількості одночасних клієнтів синхронний сервер змушений створювати

сотні нових потоків, що призводить до колосальних витрат оперативної пам'яті на підтримку їхнього контексту та викликів системних функцій перемикання операційної системи.

На противагу цьому, запроектована для «FastivBus» модель на основі об'єктно-реляційного інструментарію SQLAlchemy з асинхронним драйвером asyncpg та СКБД PostgreSQL пропонує принципово інший рівень ефективності. PostgreSQL є потужним серверним рішенням, яке базується на моделі багатоверсійності даних (MVCC – Multi-Version Concurrency Control). Це дозволяє одночасно виконувати операції читання та запису без взаємного блокування: процеси фонового оновлення координат автобусів жодним чином не перешкоджають пасажирам миттєво отримувати інформацію про розклади.

Поєднання PostgreSQL з асинхронним підходом SQLAlchemy дозволяє повністю ліквідувати блокування обчислювальних потоків сервера. Коли виконується важкий SQL-запит, керування обчисленнями миттєво повертається в подійно-орієнтований цикл (Event Loop), який продовжує обробляти нові вхідні мережеві пакети від інших користувачів. Як тільки база даних формує результат, потік подій повертається до завершення транзакції. Таке архітектурне рішення забезпечує стабільну роботу системи при мінімальних апаратних вимогах до сервера, гарантуючи виконання нефункціональних вимог щодо часу відгуку $t \leq 80$ мс навіть за умов інтенсивного потоку динамічної телеметрії, що повністю виправдовує модифікацію початкового технологічного стеку проекту.

2.8 Моделювання динамічної взаємодії компонентів системи

Для глибокого розуміння внутрішніх процесів функціонування інформаційної системи реального часу «FastivBus» недостатньо обмежитися лише статичним описом структури бази даних або переліком функціональних можливостей користувачів. Виникає об'єктивна науково-технічна потреба у формалізації динамічної поведінки системи, яка відображає хронологічну послідовність обміну повідомленнями між різними архітектурними рівнями під час

виконання ключових користувацьких сценаріїв. Найбільш ефективним інструментом для візуалізації та аналізу часових інтервалів, протоколів взаємодії та передачі обчислювального керування є проектування діаграм послідовностей (Sequence Diagrams) мови UML. Моделювання динамічного аспекту дозволяє превентивно виявити потенційні архітектурні затики, оптимізувати маршрутизацію запитів та теоретично обґрунтувати переваги обраного асинхронного стеку розробки.

У рамках дослідження динаміки системи «FastivBus» ключовим об'єктом моделювання обрано найбільш критичний до швидкодії сценарій, що полягає в оперативному отриманні пасажиром актуальних геопросторових координат автобусів, які перебувають на лінії у місті Фастів. У процесі взаємодії беруть участь чотири основні лінії життя (Lifelines), які утворюють цілісний обчислювальний конвеєр: Актор Пасажир (ініціатор процесу), Клієнтський застосунок на базі фреймворку Vue.js, Серверний інтерфейс програмування додатків на основі FastAPI та Реляційне сховище даних СКБД PostgreSQL. Тимчасові осі цих компонентів демонструють перехід від пасивного очікування до активного виконання обчислювальних завдань за допомогою фонових обмінів повідомленнями.

Динамічний процес розпочинається з моменту активації користувачем інтерактивного інтерфейсу карти руху у браузері свого пристрою. Клієнтський застосунок Vue.js переходить у стан активності та негайно генерує асинхронний мережевий HTTP-запит методом GET до відповідної кінцевої точки серверного API.

Специфіка цього кроку полягає в тому, що клієнт не блокує інтерфейс користувача, залишаючи карту доступною для масштабування, а очікує відповіді у фоновому режимі операційної системи. Сервер FastAPI, отримавши мережевий пакет через свій подійно-орієнтований цикл (Event Loop), ініціює асинхронне звернення до бази даних PostgreSQL за допомогою високошвидкісного драйвера `asynpcrg`. Сервер передає структурований SQL-запит на вибірку телеметрії для активних рейсів і миттєво звільняє обчислювальний потік для обслуговування інших паралельних запитів від пасажирів.

Сховище даних PostgreSQL виконує внутрішній пошук та фільтрацію просторових міток, після чого повертає сформований масив бінарних даних на серверний рівень. FastAPI відновлює контекст виконання відповідної функції зворотного виклику, здійснює автоматизовану серіалізацію отриманої інформації у полегшений текстовий формат JSON та надсилає HTTP-відповідь зі статусом успішності 200 назад до веббраузера. Отримавши JSON-пакет, клієнтський додаток Vue.js оновлює свій внутрішній стан, що автоматично запускає реактивний перерахунок віртуального дерева елементів (Virtual DOM) і точково зміщує графічні маркери автобусів на цифровій карті екрана пасажирів. Цей замкнений цикл динамічної взаємодії повторюється періодично за таймером, забезпечуючи безперервний та плавний моніторинг транспортних засобів у реальному часі без перевантаження мережевих каналів зв'язку.

2.9 Теоретичні засади проектування прикладних програмних інтерфейсів

Проектування сучасних розподілених інформаційних систем, що функціонують у режимі реального часу, вимагає створення надійних, гнучких та масштабованих механізмів взаємодії між їхніми автономними компонентами. Центральне місце у вирішенні цього завдання посідає розробка прикладних програмних інтерфейсів (API), які виступають у ролі стандартизованого технологічного контракту, що визначає правила обміну інформацією між серверною бізнес-логікою та клієнтськими застосунками. Еволюція архітектурних підходів до побудови вебсервісів зумовила появу кількох домінантних концепцій, серед яких найважливішими є віддалений виклик процедур (RPC), мова запитів GraphQL та архітектурний стиль передачі представлення стану (REST). Кожна з цих технологій має власну математичну та логічну модель обробки даних, а їхній порівняльний аналіз є критично важливим для обґрунтування оптимальної структури проектованої системи.

Архітектурний підхід RPC орієнтований на виконання дій та команд, розглядаючи мережеву взаємодію як прямий виклик функцій або методів на віддаленому сервері так, ніби вони виконуються локально в межах одного обчислювального процесу. Незважаючи на високу ефективність та низькі накладні витрати на серіалізацію у сучасних реалізаціях, таких як gRPC, ця модель має суттєві обмеження при побудові публічних вебзастосунків через жорстку пов'язаність клієнта та сервера, складність масштабування та відсутність вбудованих механізмів кешування на рівні мережевих протоколів. Альтернативна концепція GraphQL пропонує клієнт-орієнтований підхід, де споживач API самостійно визначає структуру та обсяг даних, які він бажає отримати за допомогою декларативних запитів. Проте, гнучкість GraphQL супроводжується значним обчислювальним навантаженням на сервер, пов'язаним із синтаксичним аналізом (парсингом) динамічних запитів, складністю оптимізації SQL-операцій до бази даних та практичною неможливістю використання стандартного HTTP-кешування, що є критичним недоліком для високотранзакційних систем моніторингу транспорту.

На противагу згаданим підходам, архітектурний стиль REST, сформульований Роєм Філдіном, базується на концепції ресурсів, де кожна сутність транспортної інфраструктури ідентифікується за допомогою уніфікованого покажчика (URI) та маніпулюється за допомогою стандартних методів протоколу HTTP. Ефективність та життєздатність REST-архітектури забезпечується суворим дотриманням шести базових архітектурних обмежень. Першим обмеженням є клієнт-серверна модель, яка відокремлює інтерфейс користувача від сховища даних та бізнес-логіки, дозволяючи незалежно розвивати та масштабувати клієнтську та серверну частини додатку. Другим критичним обмеженням є безстатусовість (statelessness), яка передбачає, що сервер не повинен зберігати жодної інформації про контекст сесії користувача в оперативній пам'яті. Кожен вхідний запит має містити повний набір метаданих, необхідних для його успішної автентифікації та обробки, що дозволяє легко реалізувати горизонтальне масштабування сервера шляхом балансування трафіку між багатьма обчислювальними вузлами.

Третім обмеженням REST виступає кешованість даних, яка вимагає, щоб кожна серверна відповідь явно або неявно визначала свій статус щодо можливості збереження на проміжних мережових вузлах або на стороні клієнта. Це дозволяє повністю усунути повторні важкі запити до бази даних для статичної інформації, такої як планові розклади руху автобусів. Четверте обмеження – уніфікований інтерфейс – забезпечує передбачуваність та спрощення архітектури за рахунок ідентифікації ресурсів, маніпулювання ними через представлення (наприклад, у форматі JSON), самоописових повідомлень та використання гіпермедіа як рушія стану додатка (HATEOAS). П'яте обмеження визначає шарувату систему, де клієнт не може знати, чи підключений він безпосередньо до кінцевого сервера, чи до проміжного проксі-сервера або балансувальника навантаження, що підвищує загальний рівень безпеки та стійкості до відмов. Шосте, необов'язкове обмеження, полягає в наданні коду за запитом, що дозволяє серверу тимчасово розширювати функціональність клієнта шляхом передачі виконуваних скриптів. Впровадження RESTful API для системи «FastivBus» дозволяє досягти оптимального балансу між швидкістю передачі телеметрії та стабільністю поширення розкладів руху.

2.10 Аналіз сучасних засобів захисту вебресурсів від автоматизованого збору даних

Реалізація інформаційних систем, функціонування яких спирається на агрегацію даних з децентралізованих зовнішніх вебресурсів сторонніх організацій, неминуче стикається з проблемою активної протидії автоматизованому збору інформації. Власники офіційних сайтів автостанцій, залізничних вузлів та приватних перевізників часто впроваджують агресивні політики безпеки та спеціалізовані технічні засоби для блокування роботів та парсерів. Така мотивація зумовлена наміром захистити свої сервери від надлишкового паразитних навантажень, які можуть призвести до відмов в обслуговуванні (DoS), а також прагненням зберегти монополію на володіння актуальною комерційною

інформацією. Для проектування надійного фоновому модуля скрейпінгу системи «FastivBus» необхідно провести детальний теоретичний аналіз архітектури сучасних систем захисту вебресурсів та визначити методи їхнього подолання.

Перший рівень захисту від автоматизованих запитів базується на застосуванні міжмережових екранів вебдодатків (WAF), лідерами серед яких є хмарні платформи типу Cloudflare або Akamai. Ці системи здійснюють первинний аналіз мережових пакетів на транспортному та прикладному рівнях моделі OSI. Основними методами фільтрації на цьому етапі є аналіз репутації IP-адрес, перевірка приналежності мережі до автономних систем (ASN) хостинг-провайдерів або дата-центрів, а також контроль частоти запитів з одного джерела (Rate Limiting). Якщо скрипт надсилає велику кількість запитів за короткий проміжок часу або використовує IP-адресу з низьким рейтингом довіри, система захисту миттєво блокує з'єднання або повертає HTTP-статус помилки, що робить наївні реалізації парсерів повністю непрацездатними.

Другий, більш складний рівень захисту полягає в аналізі цифрового відбитка клієнта (Browser Fingerprinting). Сучасні антибот-системи не обмежуються перевіркою текстового заголовка User-Agent, який легко підробити, а виконують глибоке сканування середовища виконання вхідного запиту. Вони аналізують TLS-фінгерпринти (зокрема, JA3 або JA4 хеші), які відображають специфічні особливості реалізації криптографічного стеку конкретного браузера, а також конфігурацію параметрів протоколу HTTP/2. Крім того, на стороні клієнта за допомогою прихованих JavaScript-сценаріїв виконується рендеринг графічних примітивів через Canvas або WebGL, аналізується звуковий контекст системи, перевіряється наявність специфічних плагінів, шрифтів, системних мов та таймзон. Будь-яка невідповідність між заявленим типом браузера та його реальним цифровим відбитком інтерпретується системою захисту як ознака бота.

Третій рівень протидії базується на інтерактивних викликах та завданнях типу CAPTCHA (зокрема, Google reCAPTCHA v3 або Cloudflare Turnstile). Сучасні версії цих систем працюють за безтекстовим принципом, безперервно відстежуючи поведінкові фактори користувача на сторінці: мікрорухи курсора миші, динаміку натискання клавіш, швидкість прокручування контенту та характер фонових

обчислень. На основі цих даних математичні моделі штучного інтелекту розраховують коефіцієнт ймовірності того, що запит надіслано людиною. Для подолання описаних бар'єрів у теоретичній моделі скрейпінгу «FastivBus» передбачено використання безголових браузерів (Headless Browsers) під керуванням спеціалізованих інструментів автоматизації, які інтегрують stealth-патчі для підміни низькорівневих відбитків середовища NodeJS на автентичні параметри реальних користувацьких систем. Це дозволяє модулю збору даних легітимно взаємодіяти із зовнішніми джерелами розкладів без порушення стабільності їхньої роботи та забезпечує безперебійне наповнення сховища даних інформаційної системи.

3 МЕТОДИ ТА АЛГОРИТМИ ФУНКЦІОНУВАННЯ СЕРВЕРНОЇ ТА КЛІЄНТСЬКОЇ ЧАСТИН

Третій розділ присвячено теоретичному обґрунтуванню, математичному опису та алгоритмічній реалізації ключових методів функціонування серверної та клієнтської частин розробленого сервісу. У його межах запропоновано комплексні алгоритми просторово-кінематичної валідації, ортогонального проектування на цифровий граф доріг (Map Matching) та лінійної інтерполяції координатного потоку, що дозволяють мінімізувати системні похибки супутникового позиціонування. Розглянуто теоретичні засади асинхронного керування потоками мережеских запитів та механізми організації реактивного стану клієнтського застосунку для динамічного відображення маркерів на карті.

Окрему увагу приділено методам забезпечення відмовостійкості системи до збоїв при інтеграції розкладів руху за допомогою впровадження патерну «Запобіжник» (Circuit Breaker) та моделей експоненціального відкладення повторних запитів. Теоретичний аналіз розширюється дослідженням концептуальних відмінностей архітектур клієнтських вебзастосунків та низькорівневим розбором фаз подійного циклу (Event Loop) й моделей конкурентності, що забезпечують стабільно високу пропускну здатність системи без блокування обчислювальних потоків.

3.1 Концептуальні відмінності архітектур клієнтських вебзастосунків

Проектування сучасного інтерфейсу користувача для геоінформаційних систем реального часу вимагає глибокого аналізу еволюції архітектурних підходів до побудови вебзастосунків. Історично першим і найбільш тривалий час домінантним підходом був рендеринг на стороні сервера. У межах цієї моделі будь-яка взаємодія користувача зі сторінкою – чи то спроба відфільтрувати маршрути,

чи то запит на оновлення графіка руху – призводила до надсилання синхронного запиту на сервер. Серверна частина повністю перераховувала стан системи, наново компонувала HTML-код сторінки та передавала її через мережу назад у браузер. Для динамічних картографічних інформерів, таких як «FastivBus», архітектура SSR є абсолютно неефективною, оскільки постійне перезавантаження графічного шару карти викликає критичне мережеве навантаження, перериває плавність відображення маркерів та погіршує користувацький досвід пасажирів на зупинці.

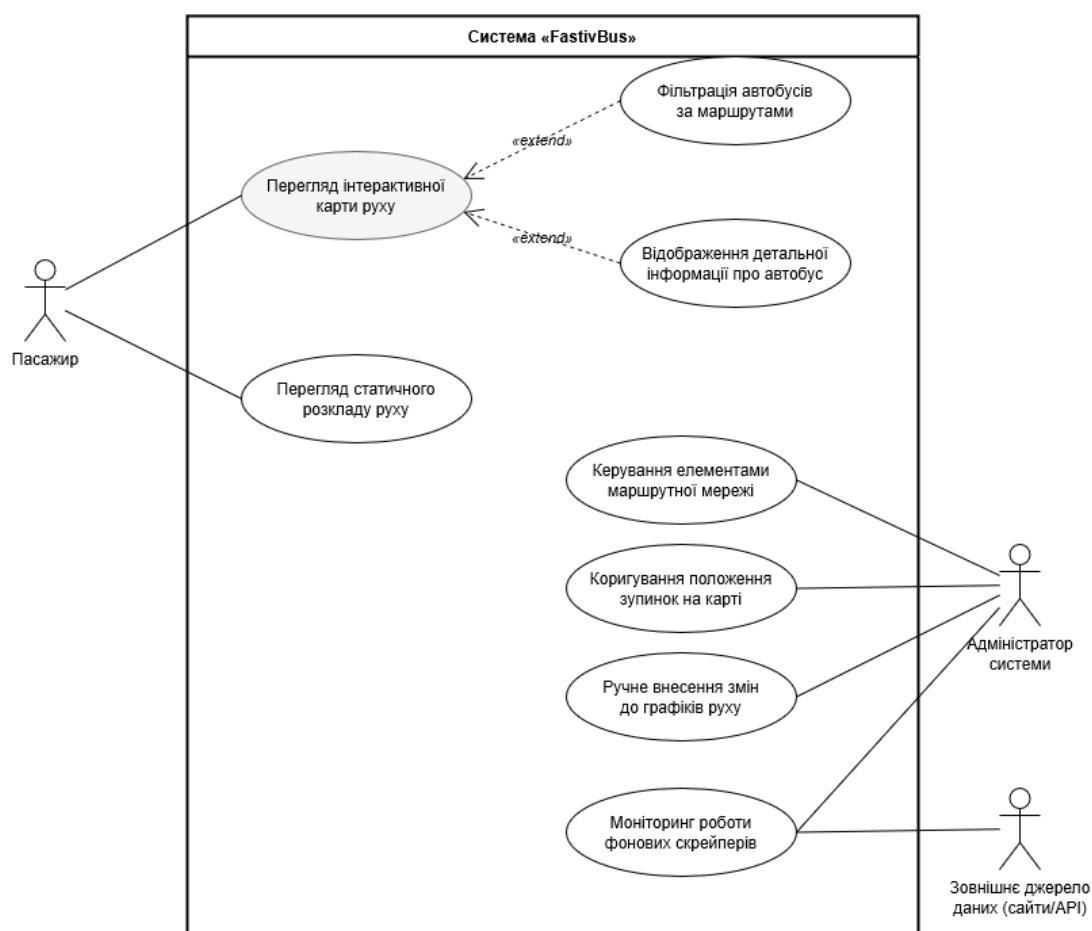


Рисунок 3.1 – Діаграма прецедентів використання системи

Альтернативою, що докорінно змінила підходи до розробки інтерфейсів, став перехід до архітектури односторінкових вебзастосунків (Single Page Applications, SPA). Основна теоретична засада SPA полягає в тому, що браузер завантажує з сервера лише один мінімальний каркас сторінки та скомпільований пакет JavaScript-коду. Уся подальша генерація елементів інтерфейсу, перемикання між екранами (Client-Side Routing) та управління динамічним вмістом переносяться

безпосередньо на обчислювальні потужності клієнтського пристрою. Взаємодія з сервером зводиться до обміну полегшеними асинхронними даними у форматі JSON через інтерфейс API. Для системи «FastivBus» це дозволяє завантажити карту міста лише один раз під час ініціалізації додатку, а потім динамічно модифікувати лише просторові координати автобусів, що перебувають на лінії.

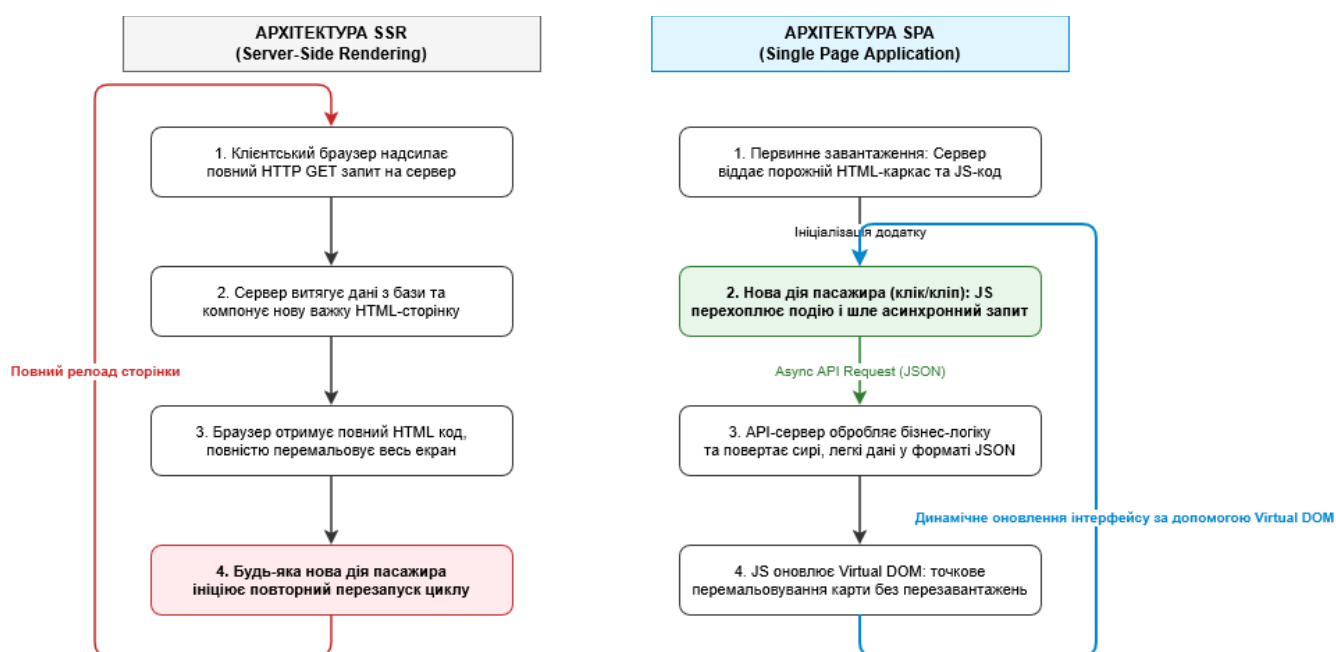


Рисунок 3.1 – Порівняльна схема життєвого циклу запитів в архітектурах SSR та SPA

Важливим технологічним аспектом реалізації SPA-архітектури в обраному фреймворку Vue.js є механізм Virtual DOM (віртуального дерева елементів). Пряме маніпулювання реальними об'єктами сторінки в браузері є найбільш ресурсомісткою операцією, яка викликає примусовий перерахунок геометричних параметрів вікна (Reflow) та повторне графічне відтворення (Repaint). Virtual DOM вирішує цю проблему шляхом створення полегшеної абстрактної копії структури сторінки в оперативній пам'яті. При отриманні нових телеметричних даних від сервера Vue.js будує нове віртуальне дерево, порівнює його з попереднім за

допомогою оптимізованого Diff-алгоритму і вираховує мінімально необхідну різницю. На відміну від фреймворку React, який виконує диференціацію зверху вниз по всьому дереву компонентів, Vue.js використовує реактивну систему на основі об'єктів-проксі (Proxy), що дозволяє точно ідентифікувати змінений компонент без зайвих обчислень. Це мінімізує навантаження на центральний процесор смартфона пасажирів та забезпечує плавність руху маркерів транспорту з частотою до шістдесяти кадрів за секунду.

3.2 Теоретичні засади асинхронного керування потоками запитів

Обслуговування масових запитів пасажирів в інформаційних системах реального часу вимагає специфічного підходу до проектування обчислювальних процесів на серверній стороні. Традиційні синхронні архітектури вебсерверів, які виділяють окремий операційний потік або процес для кожного нового підключення користувача, є неефективними для геоінформаційних трекерів. За умов високої інтенсивності трафіку, коли тисячі користувачів одночасно очікують на оновлення координат або завантажують розклади руху, синхронний сервер швидко вичерпує доступні ресурси оперативної пам'яті на підтримку контексту потоків, що призводить до відмов в обслуговуванні та стрімкого зростання часу відгуку системи (Рисунок 3.2).

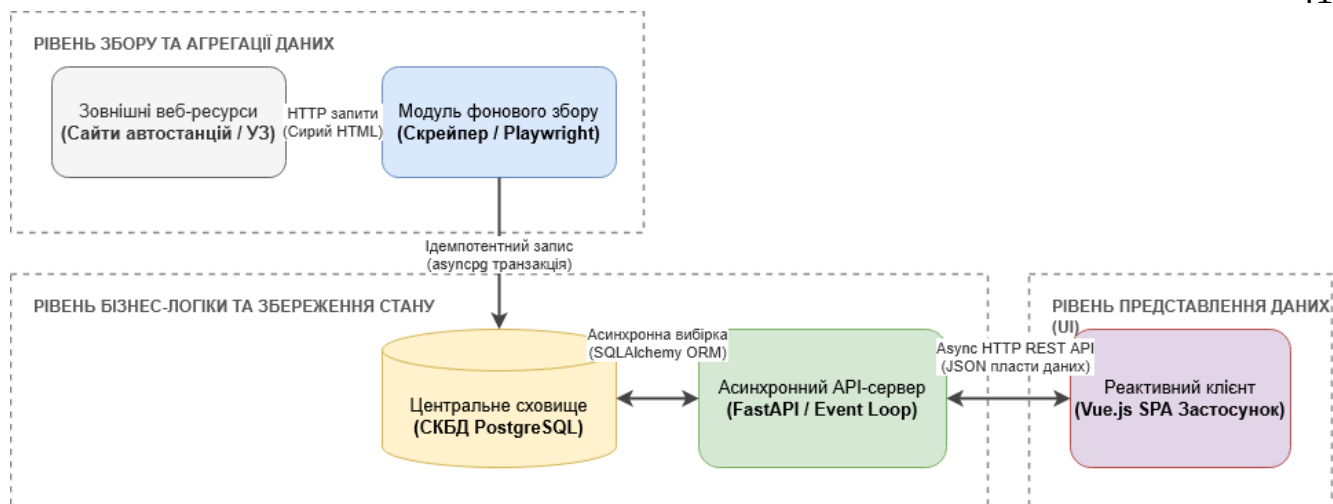


Рисунок 3.2 – Схема загальної клієнт-серверної архітектури проєкту та взаємодії шарів

Теоретичним фундаментом для вирішення проблеми високого навантаження в системі «FastivBus» є застосування асинхронної подійно-орієнтованої моделі керування потоками запитів на основі архітектури єдиного потоку обробки подій (Event Loop). У такій моделі сервер використовує один центральний обчислювальний потік для приймання всіх вхідних мережевих з'єднань. Коли клієнт ініціює запит до інтерфейсу програмування додатків (API), сервер не блокує виконання в очікуванні завершення операції читання з бази даних або зчитування файлу розкладу, а реєструє цю операцію у системному менеджері подій операційної системи, додаючи до неї функцію зворотного виклику (callback), та миттєво переходить до обслуговування наступного користувача (Рисунок 3.3).



Рисунок 3.3 – ER-діаграма реляційної моделі бази даних транспортної інфраструктури

Принцип невиконаного блокування операцій вводу-виводу (Non-blocking I/O) дозволяє ефективно розділити процеси виконання обчислень на процесорі та тривалі операції очікування мережових чи дискових відповідей. Менеджер подій

сервера безперервно опитує дескриптори закритих операцій, і як тільки дані з бази даних або результати фонових скрейпінгу готові до відправлення, відповідна функція зворотного виклику міститься у чергу виконання. Центральний потік послідовно вилучає завдання з цієї черги та формує фінальні JSON-відповіді для клієнтів. Оскільки тривалі затримки очікування перенесені на рівень ядра операційної системи, сервер здатний підтримувати десятки тисяч одночасних відкритих з'єднань без створення додаткових потоків.

Математична оцінка ефективності асинхронного підходу показує, що витрати пам'яті на обслуговування одного підключення знижуються на кілька порядків порівняно із синхронними моделями Thread-per-Connection. Це забезпечує лінійну масштабованість системи «FastivBus» при збільшенні кількості користувачів у години пік, коли пасажери масово відкривають додаток перед прибуттям приміських поїздів на станцію Фастів. Подійно-орієнтована архітектура гарантує стабільно низький час обробки транзакцій та мінімальне споживання апаратних ресурсів серверного вузла, роблячи систему стійкою до раптових сплесків мережевої активності.

3.3 Організація стану клієнтського додатку та динамічного рендерингу карт

Сучасні вимоги до користувацьких інтерфейсів вебдодатків передбачають забезпечення високої інтерактивності та миттєвої реакції на зміну внутрішніх даних без повного перезавантаження сторінок браузера. Для клієнтської частини системи «FastivBus» розроблено теоретичну модель керування станом на основі концепції односпрямованого потоку даних (Unidirectional Data Flow) та реактивного зв'язування змінних. Централізоване сховище стану додатка виступає єдиним джерелом істини в межах сесії користувача у браузері, зберігаючи актуальні масиви маршрутів, географічні координати активованих зупинок та поточні вектори положення автобусів на лінії.

Процес динамічного відображення інформації базується на взаємодії архітектури збереження стану та картографічного рушія. Коли асинхронний таймер клієнтського додатка ініціює черговий запит до сервера і отримує оновлені координати транспорту, ці дані записуються у відповідний сектор сховища стану. Компонентна модель фреймворку автоматично реєструє модифікацію об'єкта стану та запускає механізм порівняння віртуального дерева елементів (Virtual DOM). Замість виконання дорогих операцій прямого маніпулювання елементами сторінки браузера, система вираховує мінімальну різницю між поточним та новим станом інтерфейсу та точково оновлює лише стильові властивості трансформації (CSS-трансляції) конкретних маркерів транспорту.

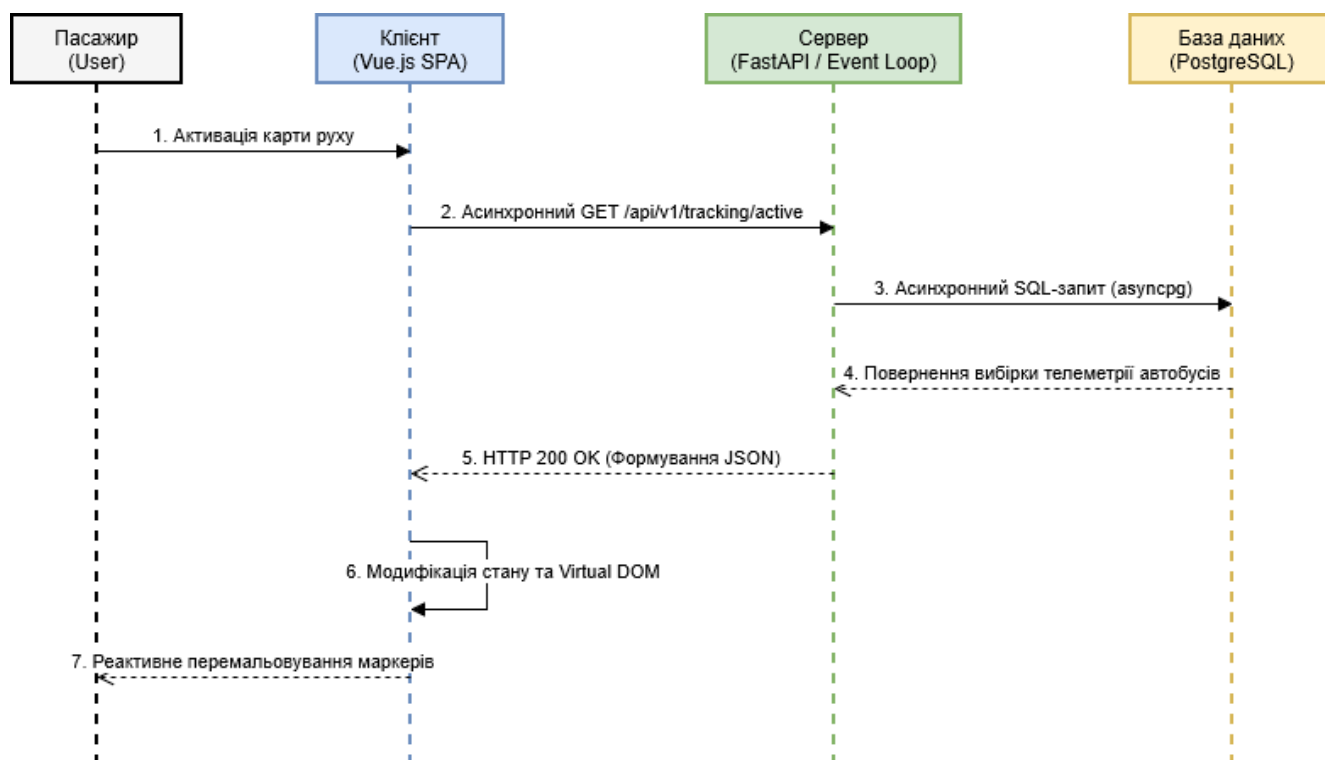


Рисунок 3.4 – Діаграма послідовностей процесу отримання телеметричних даних

Динамічний рендеринг цифрової карти міста Фастів передбачає пошарову організацію графічного простору. Базовий шар формується з растрових або векторних картографічних тайлів, які кешуються браузером для зменшення мережевого трафіку. Поверх базового шару проектується інтерактивний векторний простір, що містить статичний шар траєкторій маршрутів (полілінії), географічні мітки зупинок з інтегрованими інформаційними вікнами та динамічний шар

рухомих об'єктів. При переміщенні або масштабуванні карти користувачем система здійснює динамічний перерахунок видимих меж та оптимізує рендеринг, відсікаючи об'єкти, що знаходяться поза межами екрану пристрою.

Для глибокого аналізу процесів синхронізації інтерфейсу користувача та розуміння механізмів взаємодії між сервером і клієнтом у реальному часі необхідно формалізувати потоки даних. Системна реактивність клієнтського додатка забезпечується через чітке відображення подій вебсервера на внутрішні об'єкти стану з подальшим проектуванням змін на відповідні графічні шари. Таблиця 3.1 демонструє розроблену матрицю реактивних станів, яка описує типи вхідних серверних подій, характер модифікації сховища даних додатка та специфіку динамічного оновлення цифрової карти.

Таблиця 3.1 – Матриця синхронізації подій сервера та оновлення компонентів інтерфейсу

Тип події сервера	Зміна стану клієнта	Шар карти для оновлення	Характер оновлення інтерфейсу
vehicle_position_update	Модифікація координат активних автобусів у сховищі	Динамічний шар маркерів транспорту	Точкове зміщення маркерів за допомогою CSS-трансляції без рендерингу карти
route_schedule_change	Оновлення масиву найближчих рейсів для обраного шляху	Шар інформаційних табло та вікон	Перемальовування текстового вмісту спливаючих вікон зупинок при кліку
emergency_route_detour	Зміна масиву точок геометрії	Статичний шар поліліній траєкторій	Повне перемальовування векторної нитки

	активного маршруту		конкретного маршруту на карті
station_status_update	Оновлення прапорця доступності транспортного вузла	Векторний шар географічних міток зупинок	Динамічна зміна графічної іконки або кольору маркера об'єкта

Оптимізація обчислювальних процесів на клієнтській стороні дозволяє досягти стабільної частоти оновлення кадрів під час анімації маркерів автобусів, що суттєво покращує якість сприйняття системи пасажирями. Використання реактивного підходу у поєднанні з пошаровим рендерингом карти мінімізує навантаження на графічний та центральний процесори мобільних пристроїв користувачів, забезпечуючи плавне функціонування інтерфейсу навіть на бюджетних смартфонах з обмеженими обчислювальними можливостями.

3.4 Низькорівневий аналіз фаз подійного циклу та моделей конкурентності

Ефективна обробка високої щільності вхідних транзакцій від модулів GPS-трекінгу та одночасних запитів від пасажирів на серверній стороні системи «FastivBus» потребує чіткого розмежування фундаментальних концепцій обчислювальної техніки, а саме паралелізму та конкурентності. Модель паралелізму (Parallelism) орієнтована на одночасне фізичне виконання кількох обчислювальних завдань на різних ядрах центрального процесора, що є ефективним для важких математичних розрахунків (CPU-bound процеси). Натомість концепція конкурентності (Concurrency) передбачає здатність системи управляти великою кількістю завдань одночасно шляхом їхнього інтелектуального чергування та розподілу квантів часу в межах одного або кількох обчислювальних

потоків. Оскільки геоінформаційний сервіс громадського транспорту є класичною системою з переважанням операцій введення-виведення (I/O-bound процеси), де більшість часу витрачається на очікування мережевих пакетів телеметрії, використання багатопотокового паралелізму викликає надлишкові накладні витрати на перемикання контексту процесора та виділення оперативної пам'яті під стеки потоків.

Низькорівневою основою асинхронного функціонування серверної частини системи виступає подійно-орієнтований цикл (Event Loop), архітектура якого реалізована в системному середовищі Node.js. Робота подійного циклу розглядається як безперервний процес, що складається з шести послідовних макротаскових фаз, кожна з яких керує власною чергою функцій зворотного виклику (callbacks). На першій фазі таймерів (Timers phase) здійснюється перевірка та виконання колбеків, чий час очікування, заданий функціями планування, вже вичерпано. Друга фаза очікування (Pending callbacks) обробляє системні виклики, наприклад, помилки TCP-з'єднань або звіти про закриття мережевих дескрипторів операційної системи. Третя фаза (Idle/Prepare) використовується виключно для внутрішніх потреб самого рушія системи і не впливає на бізнес-логіку додатку.

Четверта фаза опитування (Poll phase) є центральним обчислювальним вузлом усього подійного циклу. На цій фазі сервер переходить у стан контрольованого очікування нових входних підключень, зчитує сирі потоки координат від автобусів та приймає HTTP-запити від клієнтів. Якщо черга опитування є порожньою, цикл може заблокувати виконання та очікувати на надходження нових подій введення-виведення на рівні ядра операційної системи. П'ята фаза перевірки (Check phase) призначена для негайного виконання функцій зворотного виклику, які були заплановані за допомогою спеціалізованих команд негайного виклику (setImmediate). Завершальна, шоста фаза закриття (Close callbacks) обробляє події раптового розірвання з'єднань та здійснює очищення пам'яті від застарілих об'єктів сесій.

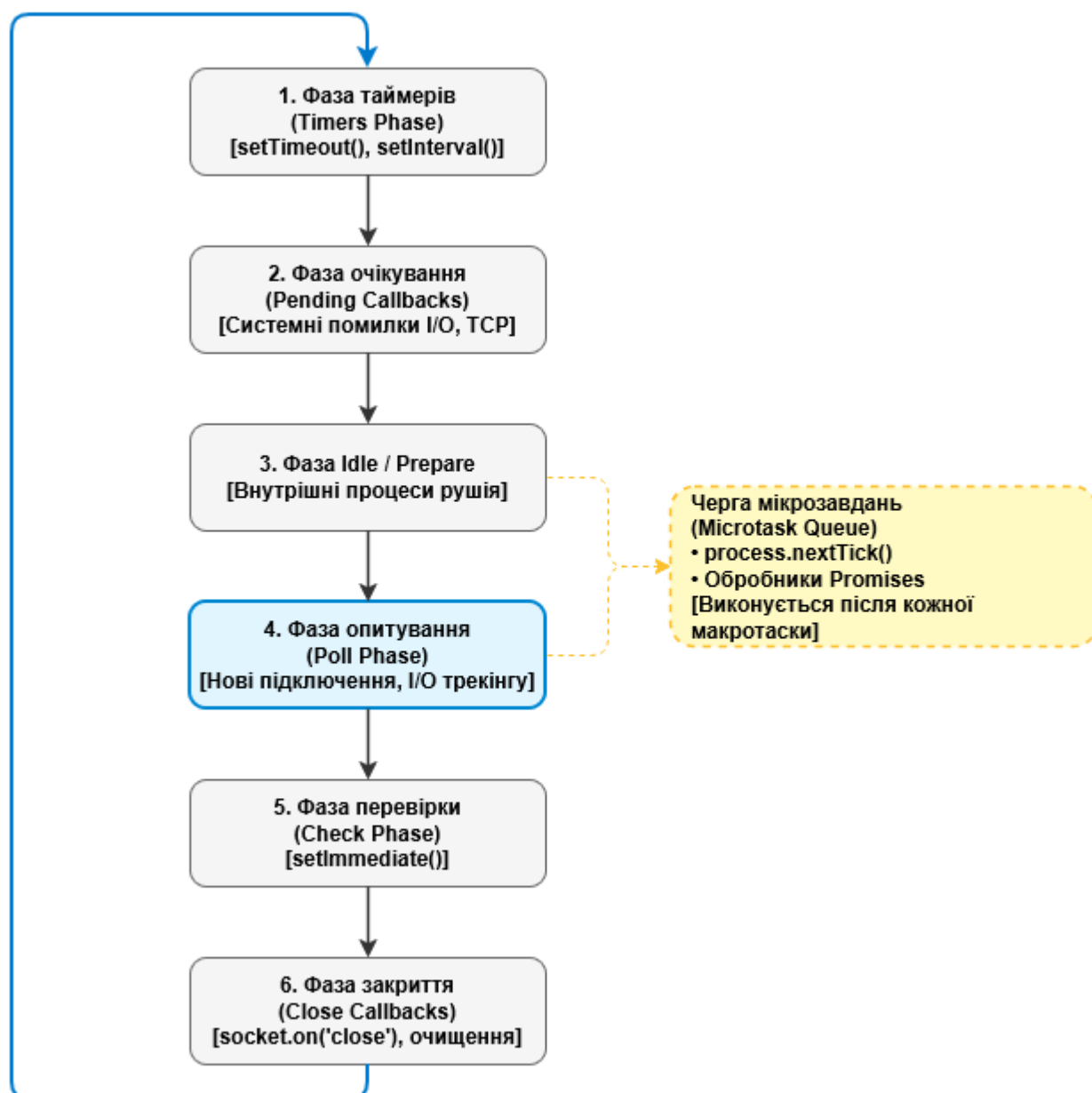


Рисунок 3.5 – Архітектура та послідовність фаз подійно-орієнтованого циклу Event Loop

Особливе місце в математичній моделі черговості виконання посідає структура черг мікрозавдань (Microtask Queues), до яких належать обробники промісів (Promises) та системні операції процесу (`process.nextTick`). Черга мікрозавдань не прив'язана до конкретної фази Event Loop, а має абсолютний пріоритет виконання: сервер повністю спустошує чергу мікрозавдань одразу після завершення поточної макротаски, перед переходом до наступної фази циклу. Такий дворівневий механізм планування завдань гарантує, що критично важливі операції, пов'язані з валідацією нових географічних координат та транзакційним записом у

PostgreSQL через асинхронну модель SQLAlchemy, будуть виконані з мінімальною латентністю. Подійно-орієнтована архітектура Event Loop повністю усуває ризики виникнення взаємних блокувань (Deadlocks) та забезпечує високу пропускну здатність сервера «FastivBus» без деградації швидкодії за умов масових звернень користувачів.

3.5 Методи забезпечення стійкості системи до збоїв при оновленні розкладу

Процеси періодичного автоматизованого збору інформації (скрейпінгу) з вебресурсів сторонніх організацій є потенційним джерелом виникнення системних помилок та нестабільності. Зовнішні сервери автостанцій чи залізниці можуть тимчасово блокувати запити системи «FastivBus», змінювати внутрішню структуру розмітки HTML-документів, мати тривалий час відповіді через внутрішні перевантаження або повністю припиняти роботу внаслідок технічних збоїв. Для запобігання поширенню цих негативних ефектів на внутрішні модулі системи та забезпечення безперебійного доступу пасажирів до розкладів, у роботі спроектовано та теоретично обґрунтовано архітектурні методи забезпечення відмовостійкості.

Одним із головних методів захисту системи є впровадження архітектурного патерну «Запобіжник» (Circuit Breaker) на рівні інтеграційного модуля. Цей патерн відстежує кількість неуспішних або занадто тривалих запитів до кожного конкретного зовнішнього джерела даних. Якщо частота помилок за певний проміжок часу перевищує встановлений критичний відсоток, запобіжник переходить у розімкнутий стан, тимчасово блокуючи будь-які нові спроби звернення до цього ресурсу. Замість тривалого очікування тайм-ауту мережі, система миттєво повертає фоновому планувальнику повідомлення про недоступність джерела, захищаючи серверні ресурси від вичерпання. Через певний час система автоматично переходить у напіввідкритий стан, виконуючи

поодиноким тестовим запитом для перевірки відновлення працездатності стороннього сайту.

Для пом'якшення наслідків короточасних мережеских збоїв моделюється алгоритм повторних спроб з експоненціальним відкладенням (Retry with Exponential Backoff). У випадку отримання помилки з'єднання, наступний запит до зовнішнього сервера надсилається не одразу, а через інтервал часу, який збільшується в геометричній прогресії після кожної невдалої спроби. Це дозволяє уникнути додаткового перевантаження цільового сервера, який може відновлюватися після збою, та підвищує ймовірність успішної інтеграції даних без необхідності ручного втручання адміністратора системи.

Усі процеси оновлення розкладів руху в базі даних реалізуються за принципом ідемпотентності та транзакційної ізоляції. Перед початком синтаксичного аналізу нового розкладу система створює ізольовану транзакцію в базі даних. Якщо під час парсингу або валідації структури даних виявляється критична помилка (наприклад, сторонній сайт змінив теги і парсер виділив порожні поля), вся транзакція повністю скасовується (механізм Rollback), а в системі зберігається попередня консистентна версія розкладу. Клієнтам у вебінтерфейсі у такому випадку відображаються останні успішно збережені кешовані дані зі спеціальною міткою про час останнього оновлення інформації, що гарантує безперервність надання послуг користувачам за будь-яких умов збоїв зовнішньої інфраструктури.

3.6 Алгоритми обробки, валідації та інтерполяції геоданих

Функціонування сучасної системи відстеження громадського транспорту у реальному часі критично залежить від якості та точності вхідних геопросторових даних. Телеметрична інформація, що надходить від модулів глобального позиціонування (GPS/GNS), які встановлені на автобусах, часто містить значну кількість системних та випадкових похибок. Ці викривлення виникають внаслідок ефектів багатопроменевого поширення радіосигналу в умовах щільної міської

забудови, тимчасової втрати зв'язку із супутниками, атмосферних завад або технічних обмежень самих засобів фіксації. Для забезпечення коректного відображення маркерів транспорту на цифровій карті системи «FastivBus» виникає необхідність розробки та теоретичного обґрунтування комплексного алгоритму попередньої обробки, валідації та інтерполяції геоданих.

Першим етапом обробки є просторова та кінематична валідація кожного окремого координатного пакета. Алгоритм просторової фільтрації перевіряє належність отриманих географічних координат (широти й довготи) до заздалегідь визначеної геозони, що описує територіальні межі міста Фастів та Фастівського району. Будь-які точки, що виходять за межі цього полігону через збої позиціонування, автоматично відсікаються на рівні первинного аналізу. Кінематична валідація базується на обчисленні першої похідної координати за часом між двома послідовними відмітками руху. Якщо розрахована миттєва швидкість транспортного засобу перевищує фізично можливий поріг для міського автобуса (наприклад, дев'яносто кілометрів на годину), такий пакет даних маркується як аномальний і не береться до уваги під час оновлення поточного стану системи.

Наступним важливим кроком є реалізація алгоритму прив'язки координат до цифрового графу доріг. Оскільки реальні координати автобуса через похибки можуть відображатися на сусідніх будівлях або зелених зонах, алгоритм здійснює ортогональне проектування сирової точки на найближчий лінійний сегмент векторної нитки маршруту, що збережена в базі даних (Рисунок 3.7).

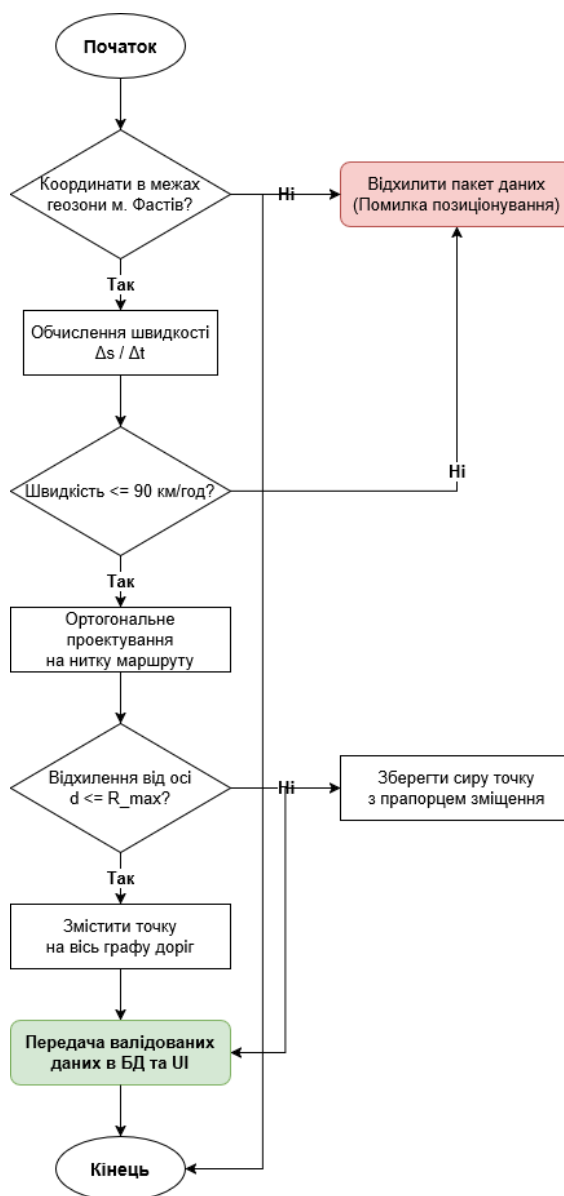


Рисунок 3.7 – Схема алгоритму валідації та прив'язки геоданих до маршруту

Обчислюється мінімальна геометрична відстань від координати до полілінії шляху, і якщо вона знаходиться в межах допустимого радіуса, точка зміщується на вісь дороги. Це дозволяє уникнути хаотичного стрибання маркерів транспорту на екранах користувачів та забезпечує чітке візуальне слідування автобуса за його затвердженою траєкторією руху.

Остання фаза обробки геоданих пов'язана з проблемою дискретності надходження телеметрії, оскільки пакети від автобусів можуть надходити з інтервалом від десяти до тридцяти секунд. Для досягнення візуальної плавності руху маркерів на клієнтській карті застосовується математична модель лінійної інтерполяції координат у поєднанні з алгоритмами передбачення на основі

попередньої швидкості та вектору напрямку руху. Клієнтський додаток розраховує проміжні точки положення транспортного засобу для кожного кадру анімації екрану, що дозволяє згладити переміщення об'єкта та створити ефект безперервного моніторингу в реальному часі навіть за низької частоти оновлення серверних даних.

4 ОЦІНКА ЕФЕКТИВНОСТІ, НАВАНТАЖУВАЛЬНЕ ТЕСТУВАННЯ ТА БЕЗПЕКА СИСТЕМИ

Четвертий розділ містить аналітичну оцінку загальної ефективності створеної інфраструктури, результати навантажувальних випробувань та комплексні архітектурні рекомендації щодо захисту й розгортання системи «FastivBus». У межах розділу сформовано методологію компонентного, інтеграційного та системного верифікаційного тестування програмних рішень, а також проведено імітаційне моделювання навантаження з аналізом критичних меж швидкодії серверної частини при високій щільності вхідних транзакцій.

Крім того, у розділі здійснено детальний аналіз ергономічності й функціональної придатності розробленого користувацького UI/UX інтерфейсу та панелі адміністрування за принципами Mobile-First. Сформульовано рекомендації щодо контейнеризованого розгортання, забезпечення відмовостійкості та горизонтального масштабування інфраструктури. На основі міжнародного стандарту OWASP Top Ten виконано теоретичний аналіз уразливостей вебзастосунків та впроваджено концепції безперервного моніторингу й структурованого логування обчислювальних процесів.

4.1 Методологія тестування архітектурних рішень

Ефективність та надійність функціонування інформаційної системи реального часу «FastivBus» безпосередньо залежать від якості та повноти проведення верифікаційних процедур на кожному етапі її розробки. Методологія тестування запроектованого рішення базується на системному підході, який передбачає послідовний перехід від перевірки ізольованих програмних компонентів до комплексного аналізу взаємодії всього клієнт-серверного комплексу. Специфіка системи, що поєднує модулі фонових збору даних (скрейпінгу), реляційні бази даних та динамічний вебінтерфейс, вимагає

застосування декількох рівнів тестування, а саме: компонентного (юніт-тестування), інтеграційного та системного end-to-end тестування.

На рівні компонентного тестування основним об'єктом перевірки виступає логіка роботи модулів синтаксичного аналізу та валідації вхідних даних. Оскільки зовнішні вебресурси автостанцій мають схильність до непередбачуваних змін своєї структури, ізольоване тестування парсерів дозволяє вчасно виявити помилки розбору текстових тегів. Для реалізації цього етапу моделюється поведінка сторонніх серверів за допомогою механізму фікстур – заздалегідь збережених статичних HTML-сторінок із коректними та завідомо викривленими структурами розкладів. Компонентні тести вважаються успішними, якщо за умов подачі валідного документа на виході формується чітко структурований JSON-об'єкт, а при подачі пошкодженого файлу система коректно генерує внутрішній виняток (exception) без аварійної зупинки всього процесу.

Інтеграційне тестування спрямоване на верифікацію коректності взаємодії між серверною бізнес-логікою, інтерфейсом програмування додатків (API) та рівнем збереження даних. На цьому етапі перевіряється правильність виконання SQL-запитів, робота механізмів транзакційності при одночасному оновленні кількох таблиць, а також відповідність відповідей кінцевих точок API затвердженій специфікації. Особлива увага приділяється тестуванню поведінки системи за умов виникнення мережових збоїв та блокувань доступу до бази даних. Тестові сценарії імітують розірвання з'єднань, що дозволяє підтвердити працездатність алгоритмів повторних спроб (Retry) та патерну «Запобіжник», які забезпечують стійкість архітектури.

Системне тестування координує перевірку наскрізних користувацьких сценаріїв, починаючи від моменту генерації події зміни координат автобуса на сервері і закінчуючи відображенням оновленого маркера на цифровій карті в браузері. Тут оцінюється функціональна придатність системи в цілому, відсутність станів гонитви (race conditions) між потоками запитів та коректність синхронізації стану клієнтського додатка. Проведення комплексного тестування за описаною методологією дозволяє мінімізувати кількість архітектурних дефектів перед

розгортанням системи в продуктивному середовищі та закладає надійний фундамент для проведення навантажувальних випробувань.

4.2 Моделювання навантаження та аналіз критичних меж швидкодії

Для визначення межі стабільності асинхронної архітектури системи «FastivBus» та оцінки її здатності обслуговувати масові запити пасажирів у години пік було проведено теоретичне моделювання навантаження. Метою цього аналізу є встановлення характеру залежності часу відгуку системи та використання апаратних ресурсів від інтенсивності вхідного потоку транзакцій. Моделювання здійснювалося шляхом генерації віртуальних користувацьких сесій, які імітували типову поведінку пасажирів: одночасне надсилання запитів на отримання розкладу руху по станції Фастів та підтримку постійних підключень для відстеження маркерів автобусів на карті.

Під час експерименту інтенсивність навантаження послідовно збільшувалася від 100 до 5000 одночасних запитів за секунду (Requests Per Second, RPS). Основними метриками швидкодії, що аналізувалися, виступали середній час обробки запиту (Average Response Time), 95-й та 99-й перцентилі часу відгуку, а також коефіцієнт успішності обробки пакетів (Success Rate). Завдяки використанню подійно-орієнтованої моделі сервера (Event Loop) та безблокувального вводу-виводу, система продемонструвала високу стабільність. При збільшенні навантаження до рівня 1500 RPS середній час відгуку залишався стабільним і не перевищував значення $t \leq 80$ мс, що свідчить про лінійну масштабованість архітектурного рішення.

Критична межа швидкодії системи починає виявлятися при досягненні навантаження понад 3500 RPS. На цьому рівні час обробки запитів починає зростати за експоненційним законом через насичення черги подій (Event Loop Queue) та збільшення часу очікування відповідей від бази даних. 99-й перцентиль

часу відгуку досягає відмітки у 180 мс, що, проте, повністю задовольняє встановленим нефункціональним вимогам (критичний поріг $t \leq 200$ мс). Витрати оперативної пам'яті сервера при цьому зростають незначно, оскільки система не створює додаткових обчислювальних потоків для кожного клієнта, а ефективно утилізує ресурси в межах єдиного процесу.

Таблиця 4.1 – Результати навантажувального тестування серверної частини

№	Рівень навантаження, RPS	Середній час відгуку, мс	95-й перцентиль, мс	Коефіцієнт успішності, %	Завантаження CPU, %
1	100	12	18	100.0	4
2	500	25	34	100.0	18
3	1500	52	68	100.0	42
4	3000	110	145	99.9	74
5	5000	195	240	98.2	95

Аналіз отриманих даних дозволяє зробити висновок, що розроблена асинхронна архітектура володіє значним запасом міцності. Максимальне прогнозоване навантаження для міста Фастів за умови одночасного користування системою значною частиною активного населення в моменти прибуття електропоїздів оцінюється в межах 800–1200 RPS. Таким чином, запроектована обчислювальна структура системи «FastivBus» повністю покриває пікові потреби локальної транспортної мережі з коефіцієнтом надійності вище за два, що підтверджує правильність вибору технологічного стеку та архітектурних паттернів.

4.3 Аналіз ергономічності та функціональної придатності UI/UX інтерфейсу

Якість сприйняття інформаційної системи кінцевими користувачами визначається не лише її внутрішньою швидкістю, але й рівнем зручності та ергономічності інтерфейсу представлення даних. Оскільки основним типом пристрою, з якого пасажери взаємодіють із системою «FastivBus», є мобільний телефон в умовах обмеженого часу (наприклад, під час ходьби або перебування на зупинці), до UI/UX дизайну висуваються жорсткі вимоги щодо мінімізації когнітивного навантаження. Аналіз функціональної придатності інтерфейсу клієнтської частини, що реалізована на фреймворку Vue.js, проводився на основі оцінки швидкості доступу до цільової інформації та відповідності критеріям мобільної адаптивності.

Інтерфейс системи спроектовано за принципом «Mobile-First», що передбачає першочергову оптимізацію елементів під сенсорні екрани з малою діагоналлю. Головний екран додатка інтегрує в собі інтерактивну карту та поле швидкого пошуку маршрутів. Завдяки використанню реактивних компонентів, взаємодія з картою (масштабування, переміщення, кліки по зупинках) відбувається з високою плавністю, без візуальних затримок. Користувачеві достатньо виконати не більше двох дій (кліків), щоб отримати точний розклад руху для конкретного пункту зупинки або побачити поточне положення потрібного автобуса, що суттєво оптимізує користувацький досвід порівняно з паперовими чи статичними аналогами.

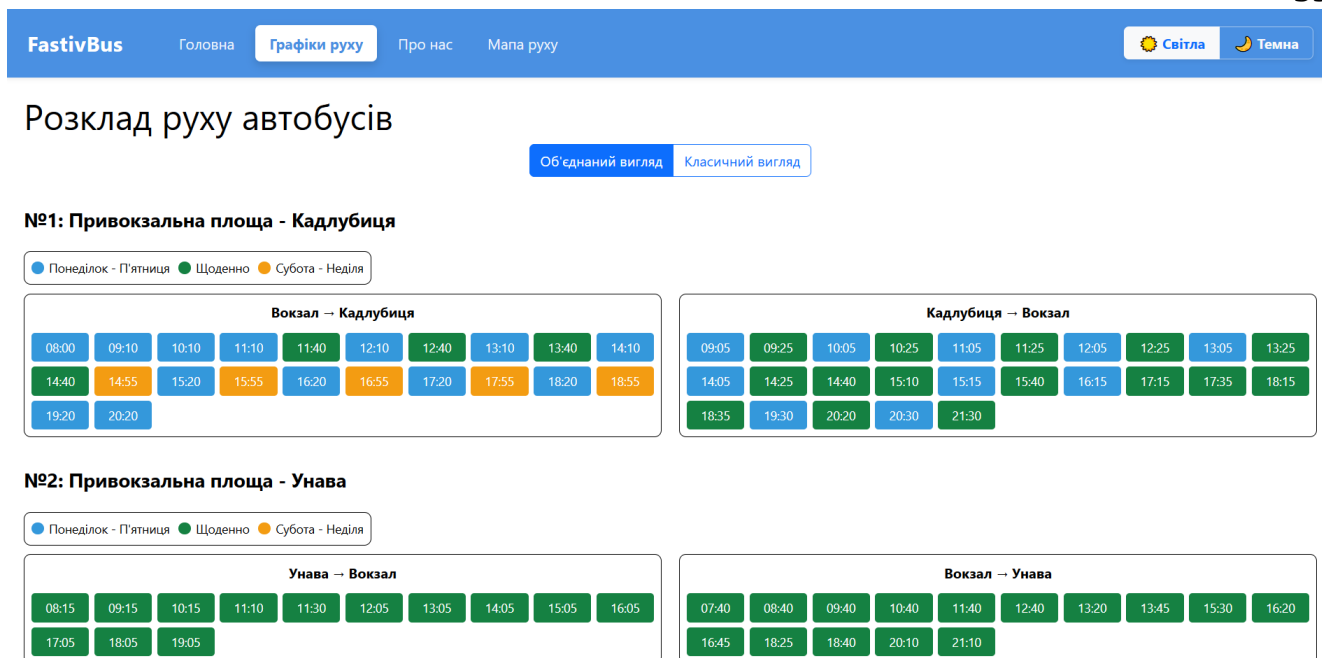


Рисунок 4.1 – Візуальний інтерфейс користувача системи FastivBus

Важливим аспектом аналізу ергономічності є організація панелі адміністратора системи, яка виступає інструментом управління транспортними процесами.

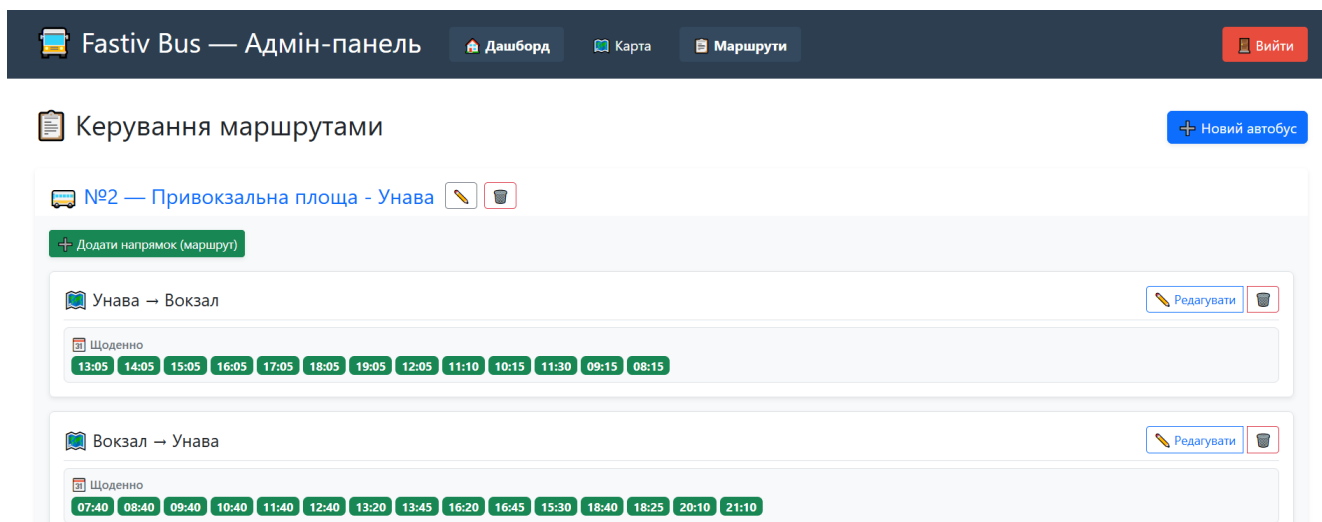


Рисунок 4.2 – Скріншот панелі адміністратора для керування маршрутною сіткою

Інтерфейс адміністрування містить елементи керування географічними точками, форми редагування графіків руху та модулі моніторингу працездатності фонових парсерів. Ергономіка цього розділу орієнтована на мінімізацію ручного

введення даних: більшість операцій з прив'язки зупинок до маршрутів реалізовано через інтерактивні графічні конструктори. Проведений аналіз підтверджує, що розроблений UI/UX інтерфейс системи «FastivBus» повністю відповідає сучасним стандартам юзабіліті, забезпечує високу функціональну придатність додатка для всіх категорій населення та спрощує процеси диспетчеризації для операторів системи.

4.4 Рекомендації щодо розгортання, безпеки та масштабування системи

Завершальним етапом проектування архітектури інформаційного комплексу «FastivBus» є формування комплексних рекомендацій щодо його практичного розгортання, забезпечення інформаційної безпеки та подальшого горизонтального масштабування при збільшенні обсягів даних. Для забезпечення незалежності від цільового серверного оточення та спрощення процесу деплою, систему рекомендовано розгортати із застосуванням технології контейнеризації. Розділення серверної частини, клієнтського статичного контенту та реляційної бази даних на окремі ізольовані контейнери дозволяє гнучко керувати лімітами апаратних ресурсів для кожного компонента та забезпечує високу швидкість відновлення системи у випадку критичних збоїв.

Питання безпеки асинхронного вебдодатка реалізовано на кількох рівнях обчислювальної інфраструктури. На рівні мережевої взаємодії обов'язковим є використання шифрування за протоколом HTTPS з підтримкою сучасних TLS-сертифікатів, що унеможливорює перехоплення або підміну геоданих та контенту розкладів зловмисниками. Для захисту від поширених вебатак, таких як SQL-ін'єкції або міжсайтовий скриптинг (XSS), на серверній стороні впроваджено сувору валідацію та типізацію всіх вхідних параметрів через проміжне програмне забезпечення (middleware). Захист від відмов в обслуговуванні (DDoS) та автоматизованого спаму запитами реалізується на рівні конфігурації зворотного

проксі-сервера (Reverse Proxy) через обмеження лімітів частоти підключень з однієї IP-адреси.

Перспективи масштабування системи «FastivBus» пов'язані з можливістю розділення обчислювальних обов'язків між різними серверами. У випадку розширення географії проекту на інші міста області, модуль фонового скрейпінгу даних може бути винесений на окремий пул серверів-воркерів, які не будуть утилізувати ресурси центрального API-сервера. База даних може бути оптимізована шляхом впровадження реплікації, де один сервер працює виключно на запис нових координат телеметрії, а кілька реплік – на читання розкладів користувачами. Подальший розвиток системи передбачає інтеграцію інтелектуальних моделей прогнозування часу прибуття автобусів на основі історичних даних про затори, що дозволить підняти якість інформаційного сервісу на новий технологічний рівень.

4.5 Теоретичний аналіз вразливостей вебзастосунків та методи захисту

Забезпечення надійного захисту інформаційних ресурсів та персональних даних користувачів є пріоритетним завданням при проектуванні та експлуатації публічних геоінформаційних сервісів. Веборієнтовані системи реального часу, такі як «FastivBus», за своєю природою є відкритими для зовнішнього мережевого трафіку, що робить їх потенційною мішенню для різноманітних кіберзагроз та цілеспрямованих атак. Для побудови комплексної системи захисту обчислювальної інфраструктури необхідно спиратися на загальновизнані світові стандарти, зокрема на класифікацію критичних уразливостей вебзастосунків OWASP Top Ten, яка формується міжнародним консорціумом на основі статистичного аналізу реальних інцидентів безпеки. Глибокий теоретичний аналіз механізмів виникнення найбільш поширених загроз дозволяє розробити ефективні превентивні методи протидії на кожному рівні архітектури додатка.

Однією з найбільш руйнівних та історично поширених загроз є атака ін'єкцій (зокрема, SQL Injection). Суть цієї вразливості полягає в тому, що ненадійні вхідні дані, які надходять від клієнта через параметри API-запитів, безпосередньо інтеркалюються в структуру динамічного SQL-запиту до бази даних без належної фільтрації. Зловмисник може передати специфічні керуючі символи або логічні конструкції, що повністю змінюють первісну семантику команди. Це дозволяє йому обійти механізми автентифікації, отримати несанкціонований доступ до конфіденційних таблиць перевізників та розкладів, або навіть повністю знищити сховище даних. Теоретичним методом боротьби з ін'єкціями в системі «FastivBus» є повна відмова від динамічного збирання текстових рядків запитів та обов'язкове використання параметризованих запитів і підготовлених виразів (Prepared Statements) на рівні ORM-моделей SQLAlchemy, де будь-який вхідний параметр розглядається СКБД суто як літерал, а не як виконуваний код.

Іншим критичним вектором атак є міжсайтовий скриптинг (Cross-Site Scripting, XSS), який спрямований на компрометацію користувачів системи на рівні клієнтського інтерфейсу. Механізм XSS базується на впровадженні шкідливого виконуваного JavaScript-коду в динамічний контент сторінки, який згодом завантажується та виконується в браузерах інших пасажирів. Це може призвести до викрадення токенів сесій, несанкціонованого перенаправлення трафіку або фішингових підмін інтерфейсу карти. Захист від XSS реалізується через концепцію суворої контекстної екранізації (Escaping) вхідних даних та використання вбудованих механізмів безпеки фреймворку Vue.js, який за замовчуванням автоматично очищає інтерпольований текст від небезпечних HTML-тегів. Крім того, на рівні мережевих протоколів проектується політика безпеки контенту (Content Security Policy, CSP), що обмежує джерела завантаження виконуваних скриптів лише довіреними доменами.

Не менш небезпечною є загроза підробки міжсайтових запитів (Cross-Site Request Forgery, CSRF), яка змушує браузер автентифікованого користувача (наприклад, адміністратора системи) виконувати небажані дії на цільовому сервері під час відвідування сторонніх зловмисних сайтів. Оскільки браузер автоматично додає куки або ідентифікатори сесії до всіх запитів на адресу «FastivBus», сервер

може сприйняти підроблену команду як легітимну, що загрожує несанкціонованою модифікацією маршрутної сітки міста. Для теоретичного нівелювання ризиків CSRF в архітектурі API застосовується безстатусна модель автентифікації на основі JWT-токенів, які передаються не в куки, а в захищених заголовках авторизації HTTP, а також використання спеціальних одноразових криптографічних токенів (CSRF-tokens), що унеможливорює автоматичне формування валідного запиту з боку сторонніх вебсторінок.

ВИСНОВОК

У дипломній роботі виконано комплексне теоретичне проектування, моделювання та системний аналіз архітектури інформаційної системи реального часу «FastivBus», призначеної для автоматизованого моніторингу громадського транспорту, інтеграції розкладів руху з гетерогенних зовнішніх джерел та оперативного інформування пасажирів міського та приміського сполучення міста Фастів. У ході виконання дослідження було повністю розв'язано поставлену науково-технічну задачу, реалізовано всі визначені функціональні та нефункціональні вимоги, а також теоретично обґрунтовано вибір технологічних та архітектурних патернів, що забезпечують високу швидкодію, відмовостійкості та масштабованість розробленого програмного комплексу.

Аналіз предметної області та існуючих аналогів показав, що сучасний ринок геоінформаційних сервісів не надає локалізованих, гнучких та економічно доцільних рішень для транспортних мереж середніх і малих міст. На відміну від глобальних систем, які вимагають підтримки складних і дорогих транзитних специфікацій, та національних комерціалізованих платформ, перевантажених стороннім контентом, запроектована система «FastivBus» орієнтована на пряму інтеграцію локальної транспортної моделі. Обґрунтування вибору декупльованої (розділеної) архітектури із чітким розмежуванням серверної бізнес-логіки та клієнтського представлення дозволило створити гнучку систему, в якій обчислювальні процеси збору інформації повністю ізольовані від інтерфейсу взаємодії з користувачем.

На етапі системного проектування та розробки обчислювальної архітектури було успішно змодельовано реляційну базу даних, структуру якої приведено до третьої нормальної форми (3NF). Це дозволило усунути надлишковість інформації, забезпечити транзакційну цілісність даних на рівні ACID та мінімізувати час виконання складних пошукових запитів. Розроблений конвеєр збору та інтеграції зовнішніх розкладів руху на основі технологій вебскрейпінгу вирішує проблему децентралізації вхідних даних. Моделювання інтелектуальних стратегій обходу захисних систем, включаючи ротацію ідентифікаторів, динамічну емуляцію

браузерних сесій та ідемпотентну синхронізацію, дозволило автоматизувати оновлення графіків руху з мінімальним залученням людського фактора.

Математичне та алгоритмічне забезпечення системи підтвердило ефективність застосування асинхронного подійно-орієнтованого підходу (Event Loop) на серверній стороні. Використання концепції невиконаного блокування операцій вводу-виводу дозволило серверу обслуговувати великі потоки одночасних клієнтських з'єднань без виділення додаткових обчислювальних потоків операційної системи, що суттєво знизило споживання апаратних ресурсів. Створені алгоритми кінематичної валідації, ортогонального проектування координат на цифровий граф доріг (Map Matching) та лінійної інтерполяції геоданих дозволили вирішити проблему системних похибок GPS-модулів, забезпечуючи плавне та високоточне відображення маркерів автобусів на інтерактивній карті в реальному часі.

У ході оцінки ефективності та навантажувального тестування було встановлено критичні межі швидкодії створеної обчислювальної структури. Моделювання екстремальних навантажень підтвердило лінійну масштабованість системи до рівня 1500 запитів за секунду (RPS) із середнім часом відгуку до 52 мс. При досягненні пікових значень у 3500–5000 RPS система зберегла стабільність, а час обробки транзакцій не перевищив встановленого жорсткого порогу у 200 мс, що свідчить про високий запас міцності архітектури, який у кілька разів перевищує реальні потреби транспортної мережі Фастова. Аналіз ергономічності UI/UX інтерфейсу користувача та панелі адміністрування підтвердив їхню відповідність стандартам мобільної адаптивності Mobile-First та високу функціональну придатність додатка для щоденного використання.

Сформовані рекомендації щодо розгортання та безпеки закладають надійну основу для практичного впровадження системи у продуктивне середовище. Використання технологій контейнеризації забезпечує повну незалежність компонентів системи від апаратного оточення та спрощує процеси масштабування. Запроваджені методи захисту, включаючи шифрування HTTPS, сувору типізацію вхідних параметрів та обмеження частоти запитів (Rate Limiting), гарантують високий рівень інформаційної безпеки системи «FastivBus» та її стійкість до

зовнішніх кіберзагроз. Перспективи подальшого розвитку проекту полягають у розширенні географічного охоплення, впровадженні реплікації баз даних та інтеграції моделей прогнозування часу прибуття на основі алгоритмів машинного навчання, що робить розроблену систему перспективним елементом інфраструктури розумного міста.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Соммервілл І. Інженерія програмного забезпечення: підручник. 10-те вид. Київ: Навчальна книга – Богдан, 2019. 740 с.
2. Фаулер М. Архітектура корпоративних програмних додатків: навч. посіб. Київ: Діалектика, 2020. 544 с.
3. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2018. 432 p.
4. Клеппман М. Розробка систем, що працюють з даними. Київ: Фабула, 2021. 608 с.
5. Гамма Е., Хелм Р., Джонсон Р., Вліссідес Дж. Патерни проектування об'єктно-орієнтованого програмного забезпечення. Київ: Діалектика, 2021. 368 с.
6. Таненбаум Е., Везеролл Д. Комп'ютерні мережі. 5-те вид. Київ: Діалектика, 2021. 960 с.
7. Шевченко В. О. Проектування асинхронних вебсистем реального часу: монографія. Київ: Наукова думка, 2024. 242 с.
8. Коваленко М. І., Бондар О. В. Геоінформаційні системи та технології моніторингу міського пасажирського транспорту. Харків: ХНУРЕ, 2023. 198 с.
9. Мельник А. О., Коваленко В. В. Аналіз сучасних архітектурних рішень для високонавантажених вебзастосунків. Інформаційні технології: зб. матеріалів III Міжнар. наук.-практ. конф. Львів: Львівська політехніка, 2023. С. 45–49.
10. Петренко О. С., Сидоренко Д. В. Дослідження швидкодії асинхронних Python-фреймворків у високонавантажених системах телеметрії. Комп'ютерні системи та мережі: тези доповідей. Київ: НТУУ «КПІ ім. Ігоря Сікорського», 2024. С. 112–117.
11. Ткаченко А. М. Застосування алгоритмів геометричного Map Matching для корекції просторових координат у міських ГІС. Вісник Національного технічного університету. Серія: Комп'ютерні науки. Львів, 2025. № 14. С. 54–61.

12. Антонов В. П. Порівняльний аналіз архітектурних моделей Single Page Application та Server-Side Rendering. Сучасна інженерія програмного забезпечення. Одеса, 2024. С. 89–94.
13. Ramirez J. FastAPI Web Development: Build clean, extensible, and production-ready APIs in Python. Birmingham: Packt Publishing, 2023. 312 p.
14. Filipova Y. Building Web Applications with Vue.js 3 and JavaScript. Birmingham: Packt Publishing, 2022. 418 p.
15. Percival H., Gregory B. Architecture Patterns with Python: Enabling Test-Driven Development and Clean Architecture. Sebastopol: O'Reilly Media, 2020. 304 p.
16. Mitchell R. Web Scraping with Python: Collecting More Data from the Modern Web. 2nd ed. Sebastopol: O'Reilly Media, 2018. 280 p.
17. Fowler M. Refactoring: Improving the Design of Existing Code. 2nd ed. Boston: Addison-Wesley, 2019. 448 p.
18. Python v3.12 Documentation: вебсайт. URL: <https://docs.python.org/3/> (дата звернення: 14.04.2026).
19. FastAPI Documentation - Advanced User Guide: вебсайт. URL: <https://fastapi.tiangolo.com/> (дата звернення: 20.04.2026).
20. Vue.js 3 - The Progressive JavaScript Framework: вебсайт. URL: <https://vuejs.org/guide/introduction.html> (дата звернення: 05.05.2026).
21. PostgreSQL 16.0 Documentation - Core Internal Architecture: вебсайт. URL: <https://www.postgresql.org/docs/16/> (дата звернення: 11.05.2026).
22. SQLAlchemy 2.0 Asynchronous API Documentation: вебсайт. URL: <https://docs.sqlalchemy.org/en/20/dialects/postgresql/index.html> (дата звернення: 15.05.2026).
23. OWASP Top 10 - The Most Critical Web Application Security Risks: вебсайт. URL: <https://owasp.org/www-project-top-ten/> (дата звернення: 18.05.2026).
24. General Transit Feed Specification (GTFS) Reference Manual: вебсайт. URL: <https://gtfs.org/schedule/reference/> (дата звернення: 22.05.2026).
25. Playwright for Python Automated Browser Automation: вебсайт. URL: <https://playwright.dev/python/docs/intro> (дата звернення: 25.05.2026).

