МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБПЛАТФОРМА ОРЕНДИ МІСЬКОГО ЕЛЕКТРОТРАНСПОРТУ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Сергій ГАВРИЛЕНКО

«___»_____ 2026 р.

Керівник професор кафедри ІІС _____Олександр МЄЩАНІНОВ

«___»_____ 2026 р.

Миколаїв – 2026

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

_____ Гавриленко Сергій Миколайович _____
((прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Вебплатформа оренди міського електротранспорту».

Керівник роботи: Мєщанінов Олександр Павлович, професор кафедри ІІС.

Затверджена наказом ЧНУ ім. Петра Могили від «25» грудня 2025 р. № 353.

2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: Вебплатформа оренди міського електротранспорту, Data-set з координатами електротранспорту.

4. Перелік питань, що підлягають розробці: аналіз існуючих вебплатформ, аналіз існуючих методів пошуку маршруту, визначення потрібних алгоритмів, опис архітектури та реалізація системи.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Олександр МЄЩАНІНОВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Сергій ГАВРИЛЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «23» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Вебплатформа оренди міського електротранспорту

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	21.12.2025	24.12.2025	
2	Аналіз предметної області та постановка задачі	25.12.2025	30.01.2026	
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема огляд публікацій та аналогічних систем, щодо вебплатформи оренди електротранспорту	31.01.2026	01.03.2026	
4	Огляд існуючих архітектур вебплатформ оренди електротранспорту для вирішення поставленої задачі	02.03.2026	01.04.2026	
5	Реалізація обраних технологій з аналізом отриманих результатів	02.04.2026	24.05.2026	
6	Перший попередній захист КР на засіданні комісії кафедри	25.05.2026	25.05.2026	
7	Корегування роботи за результатами попереднього захисту	26.05.2026	04.06.2026	
8	Другий попередній захист КР на засіданні комісії кафедри	05.06.2026	05.06.2026	
9	Доробка та остаточне оформлення КР	06.06.2026	14.06.2026	
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.06.2026	19.06.2026	

Керівник роботи _____ Олександр МЄЩАНІНОВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Сергій ГАВРИЛЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2026 р.

АНОТАЦІЯ

до кваліфікаційної роботи

здобувача групи 401 ЧНУ ім. Петра Могили

Гавриленко Сергія Миколайовича

На тему: **«Вебплатформа оренди міського електротранспорту»**

Актуальність кваліфікаційної роботи визначається зростанням популярності та значущості електротранспорту для сталого розвитку міст. Створення вебплатформи для оренди міського електротранспорту стає ключовим завданням для забезпечення ефективного та екологічного транспорту. Ця робота спрямована на розвиток і застосування веб технологій у сфері оренди електротранспорту з метою підвищення ефективності та зручності пересування користувачів.

Об'єктом кваліфікаційної роботи є створення вебплатформи для оренди міського електротранспорту.

Предметом кваліфікаційної роботи є методи, моделі та алгоритми створення вебплатформи для оренди міського електротранспорту.

Метою даної кваліфікаційної роботи є підвищення ефективності функціонування мереж електротранспорту шляхом розробки застосунку для оренди міського електротранспорту та планування маршрутів на основі інтелектуальних технологій.

В результаті виконання роботи було досліджено і порівняно два методи оптимізації, а саме алгоритм мурашиної колонії і алгоритм A*, визначені основні їх переваги та недоліки, а також розроблено програмне забезпечення, в якому реалізовані відповідні методи.

Дана робота складається з трьох розділів. У першому розділі проведено аналіз предметної області, огляд останніх публікацій та аналогів, сформовано постановку задачі. Другий розділ присвячений математичним моделям і методам, що використані у роботі. В третьому – аналіз отриманих результатів, тестування. Загальний обсяг роботи – 112 сторінок. Кваліфікаційна робота містить 1 додаток, 43 рисунків, 3 таблиці і 45 джерел посилання.

Ключові слова: оптимізація транспортних маршрутів, задача комівояжера, , TSP, ACO-метод, Php, Flutter, Javascript.

ABSTRACT

to the qualification work by the student of the group 401 of Petro Mohyla Black Sea National University

Havrylenko Serhii

"Web platform for renting city electric transport"

The relevance of qualification work is determined by the growing popularity and importance of electric transport for the sustainable development of cities. The creation of a web platform for the rental of urban electric transport becomes a key task for ensuring efficient and ecological transport. This work is aimed at the development and application of web technologies in the field of electric vehicle rental in order to increase the efficiency and convenience of user movement.

The object of the qualification work is the creation of a web platform for renting city electric transport.

The subject of the qualification work is methods, models and algorithms for creating a web platform for renting city electric transport.

The purpose of this qualification work is to increase the efficiency of electric transport networks by developing an application for renting urban electric transport and planning routes based on intelligent technologies.

As a result of the work, two optimization methods were investigated and compared, namely the ant colony algorithm and the A* algorithm, their main advantages and disadvantages were determined, and software was developed in which the corresponding methods were implemented.

This work consists of three sections. In the first chapter, an analysis of the subject area, a review of the latest publications and analogues, and a statement of the problem were formed. The second chapter is devoted to mathematical models and methods used in the work. In the third - analysis of the obtained results, testing. The total volume of work is 112 pages. The qualification work contains 1 appendix, 43 figures, 3 tables and 45 reference sources.

Keywords: optimization of transport routes, traveling salesman's problem, TSP, ACO-method, Php, Flutter, Javascript.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ 4

ВСТУП..... 5

1 АНАЛІЗ ІСНУЮЧИХ ВЕБПЛАТФОРМ ДЛЯ ОРЕДИ ЕЛЕКТРОТРАНСПОРТУ 6

1.1 Актуальність кваліфікаційної роботи 6

1.2 Огляд існуючих аналогічних вебплатформ..... 6

1.3 Опис і аналіз існуючих веб-технологій 12

2 ОСОБЛИВОСТІ ЗАСТОСУВАННЯ АЛГОРИТМІВ В ВЕБПЛАТФОРМІ ОРЕНДИ ЕЛЕКТРОТРАНСПОРТУ 27

2.1Принцип природного алгоритму..... 27

2.2 Вирішення задач із застосуванням мурашиного алгоритму..... 31

2.3 Задачі комівояжера 33

2.4 Фронтенд-збірка vite.js..... 34

2.5 Базовий мурашиний алгоритм для розв'язання задачі пошуку маршруту 36

Висновки до розділу 2 43

3. РОЗРОБКА ВЕБПЛАТФОРМИ ДЛЯ ОРЕНДИ МІСЬКОГО ЕЛЕКТРОТРАНСПОРТУ НА ОСНОВІ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ... 46

3.1 Програмна реалізація єдиної точки входу (API Gateway) на базі PHP 46

3.2 Програмна реалізація компонента низькорівневого доступу до даних 53

3.3 Реалізація компонента бізнес-логіки керування транзакціями та замовленнями..... 54

3.4 Реалізація сервісного моніторингу та керування парком транспортних засобів..... 59

3.6 Програмна реалізація компонента автентифікації та профілювання користувачів 63
3.7 Реалізація і тестування застосунку 69
Висновки до розділу 3 77
ВИСНОВКИ 79
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 81
Додаток А Код реалізації вебплатформи ореди електротранспорту 84

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ЕМА — Евристичний мурашиний алгоритм

МА — Мурашиний алгоритм

ПЗ — Програмне забезпечення

ACO — Ant Colony Optimization (Оптимізація на основі мурашиних колоній)

ACS — Ant Colony System (Система мурашиних колоній)

СКБД — Система керування базами даних

HTTP — (HyperText Transfer Protocol)

ВСТУП

Багато міст у всьому світі активно впроваджують електротранспорт у рамках своїх стратегій сталого розвитку. Створення відповідного застосунку повністю узгоджується з цими глобальними тенденціями та реальними потребами. Електротранспорт характеризується нижчими експлуатаційними витратами завдяки дешевшій електроенергії порівняно з традиційним паливом, а також меншою потребою в технічному обслуговуванні через меншу кількість рухомих механізмів.

Проблематика створення вебплатформи для оренди міського електротранспорту є актуальною та широко досліджуваною у різних галузях. Для її вирішення застосовується низка алгоритмів, які забезпечують високу ефективність і точність. Успішність програмного забезпечення значною мірою залежить від обраного підходу до оптимізації. Згідно з сучасними дослідженнями, застосування методів штучного інтелекту, зокрема мурашиного алгоритму, дозволяє знаходити ефективні рішення для визначення найближчого доступного транспорту. Крім того, використання RESTful API та адаптивного клієнтського інтерфейсу забезпечує зручність користування на мобільних пристроях.

У цій роботі за основу було взято саме мурашиний алгоритм завдяки його численним перевагам. Він імітує природну поведінку мурах, які за допомогою феромонів знаходять найкоротший шлях до джерела їжі. Цей алгоритм функціонує паралельно, розглядаючи одночасно багато варіантів, що дозволяє ефективніше знаходити оптимальні рішення.

1 АНАЛІЗ ІСНУЮЧИХ ВЕБПЛАТФОРМ ДЛЯ ОРЕДИ ЕЛЕКТРОТРАНСПОРТУ

1.1 Актуальність кваліфікаційної роботи

Актуальність кваліфікаційної роботи базується у контексті росту популярності та важливості електротранспорту для сталого розвитку міст. Оптимізація вебплатформ електротранспорту стає ключовим завданням у забезпеченні ефективного та екологічно чистого транспорту. Дана робота спрямована на розвиток та застосування веб- технологій у сфері оренди електротранспорту з метою підвищення ефективності та зручності пересування для користувачів.

1.2 Огляд існуючих аналогічних вебплатформ

KOBI Electric — сервіс шерингу електросамокатів, що працює в Тернополі, Хмельницькому та Миколаєві. Користувачі можуть орендувати самокати через мобільний додаток, поповнивши електронний гаманець, знаходячи самокати на мапі та розблоковуючи їх за допомогою QR-коду (рис 1.1) [1].

Рисунок 1.1 – вебплатформа для ореди електротранспорту KOBI [1]

Тарифи**:**

– розблокування: 9 грн.;

– поїздка: 3,5 грн/хв.;
– пауза або бронювання: 1 грн/хв.

Підписки:

– безліміт на день: 450 грн.;
– безліміт на тиждень: 2250 грн.;
– знижка 50% на всі послуги: 330 грн/тиждень або 900 грн/місяць;
– безоплатне розблокування: 90 грн/7 днів або 280 грн/30 днів.

Особливості:

Зони катання позначені на мапі:

– блакитна зона — дозволено кататись та паркувати;
– сіра зона — дозволено кататись, але паркувати заборонено;
– червона зона — заїжджати заборонено; самокат автоматично блокується.

ПРОКАТайся — український сервіс оренди електросамокатів, доступний через мобільний додаток. Користувачі можуть знаходити самокати на мапі, розблоковувати їх за допомогою QR-коду та обирати зручний тариф для поїздки (рис 1.2) [2].

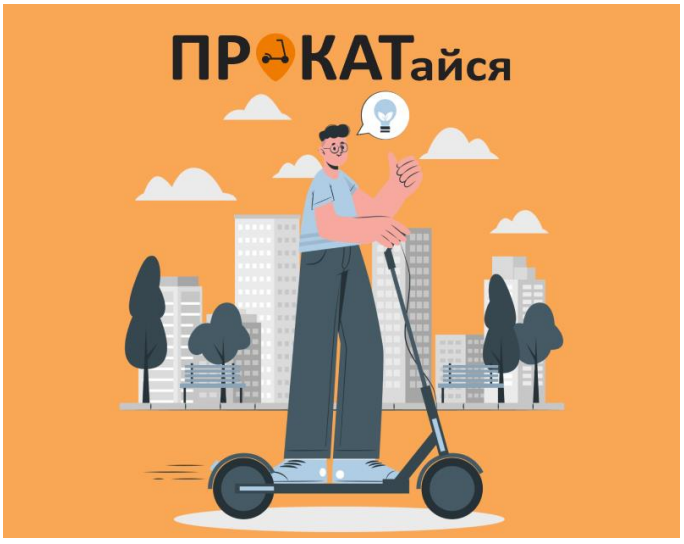

Рисунок 1.2 – вебплатформа для ореди електротранспорту ПРОКАТайся[2]

Порівняли обидві вебплатформи у таблиці 1.1.

Таблиця 1.1 – порівняльна таблиця вебплатформ KOBI та ПРОКАТайся

Характеристика	**KOBI Electric**	**ПРОКАТайся**
Міста покриття	Тернопіль, Хмельницький, Миколаїв	Множинні міста по Україні
Розблокування	9 грн	Залежить від тарифу
Вартість поїздки	3,5 грн/хв	Залежить від тарифу
Безкоштовне бронювання	Немає	5 хвилин
Пауза	1 грн/хв	1 грн/хв
Підписки	Так (різні варіанти)	Немає інформації
Зони катання	Чітко визначені зони на мапі	Інформація не вказана
Франшиза	Немає інформації	Доступна

Jet.UA — український сервіс шерингу електросамокатів, що пропонує зручний та інтуїтивно зрозумілий мобільний додаток для оренди транспорту. Сервіс орієнтований на франчайзингову модель, що дозволяє партнерам запускати власний бізнес з оренди електросамокатів (рис 1.3) [3].

Рисунок 1.3 – вебплатформа для ореди електротранспорту JET.ua[3]

Технічна реалізація

Мови програмування: Flutter (клієнтська частина), Node.js (серверна частина).

Архітектура: Кросплатформенна, з можливістю масштабування.

Інтеграції:

– платіжні системи;
– CRM-система для управління взаємодією з клієнтами;
– IoT-модулі для відстеження стану самокатів.

Користувацький інтерфейс:

– реєстрація та авторизація;
– карта з відображенням доступних самокатів;
– інформація про стан самоката та тарифи;
– оренда самоката;
– особистий кабінет з історією поїздок та балансом;
– промокоди та реферальна программа;
– повідомлення та налаштування;
– режим новачка з обмеженою швидкістю;
– інформаційна сторінка та форма зворотного зв'язку.

Адміністративна панель:

– управління користувачами та менеджерами;
– додавання та редагування самокатів;
– історія поїздок та оренди;
– створення територій та зон пересування;
– керування батареями та облік заміни акумуляторів;
– створення промокодів та розсилок;
– редагування текстового вмісту програми;
– налаштування франшизи та статистика.

Особливості:

– франчайзинг: Можливість запуску власного бізнесу з оренди електросамокатів під брендом Jet.UA;
– CRM-система: Інтегрована система для управління взаємодією з клієнтами та збору відгуків;
– IoT-інтеграція: Відстеження стану самокатів та управління ними через додаток.

FlyGo — сервіс автоматизованого прокату електросамокатів у Києві, що пропонує простий та швидкий спосіб оренди транспорту через мобільний додаток (рис 1.4) [4].

Рисунок 1.4 – вебплатформа для ореди електротранспорту FlyGo[4]

Тарифи:

Розблокування: 10 грн.

Поїздка:

– Пн–Чт: 2.5 грн/хв.;

– Пт–Нд: 3 грн/хв.

Бронювання: 5 хвилин безкоштовно, далі 1 грн/хв.

Пауза: 1 грн/хв.

Штраф за неправильне паркування: 50 грн.

Зони користування:

– паркування: Дозволено лише в спеціально відведених зонах, позначених у додатку;

– заборонені зони: Позначені темним кольором та значками заборони в додатку.

Технічна реалізація:

Мови програмування: Імовірно Flutter для кросплатформенності.

Інтеграції**:**

– платіжні системи Visa/Mastercard;

– IoT-модулі для відстеження стану самокатів.

Особливості:

– групова поїздка: Можливість орендувати до 3-х електросамокатів з одного додатку одночасно;

– безпека: Рекомендації щодо використання захисного спорядження та дотримання правил дорожнього руху;

– заряд батареї: Інформація про рівень заряду кожного самоката доступна в додатку.

Порівняли обидві вебплатформи у таблиці 1.2.

Таблиця 1.2 – порівняльна таблиця вебплатформ JET.ua та FlyGo

Характеристика	**Jet.UA**	**FlyGo**
Географія покриття	Множинні міста України	Київ
Розблокування	Залежить від тарифу	10 грн
Вартість поїздки	Залежить від тарифу	2.5–3 грн/хв

Кінець таблиці 1.2

Безкоштовне бронювання	Інформація не вказана	5 хвилин
Пауза	Інформація не вказана	1 грн/хв
Підписки	Інформація не вказана	Інформація не вказана
Зони катання	Інформація не вказана	Позначені в додатку
Франшиза	Доступна	Інформація не вказана
Технології розробки	Flutter, Node.js	Імовірно Flutter
IoT-інтеграція	Так	Так
CRM-система	Так	Інформація не вказана

1.3 Опис і аналіз існуючих веб-технологій

Алгоритм A* та мурашиний алгоритм

A* - це алгоритм, який шукає найоптимальніший шлях від початкової вершини до кінцевої, зосереджуючись на мінімізації вартості, і використовує перший найкращий збіг у процесі пошуку (рис 1.5).

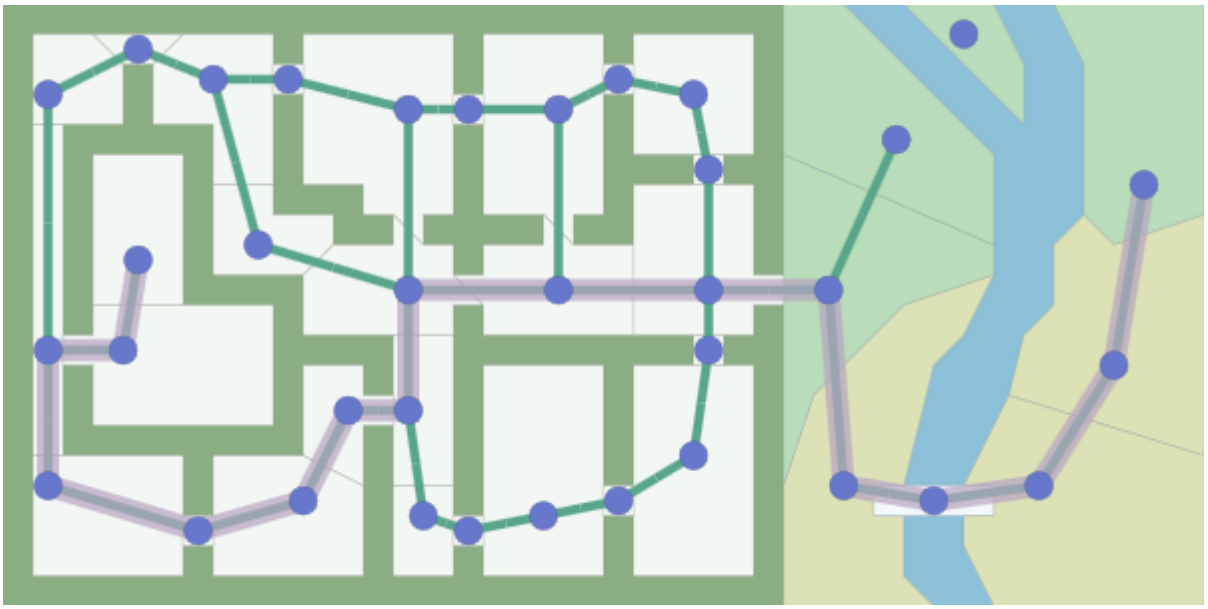

Рисунок 1.5 – Приклад роботи алгоритма A*[5]

В основі вибору маршруту в алгоритмі A* лежить спеціальна оціночна функція f(x), яку часто називають комбінацією «відстані та вартості». Вона розраховується як сума двох ключових компонентів:

– g(x) — це реальна ціна, яку ми вже заплатили, щоб дійти зBuck-початкової точки до поточної вершини x;

– h(x) — наш «прогноз» або евристична припущення про те, скільки ще доведеться пройти від x до фінішу.

Тут є важливий нюанс: функція прогнозу h(x) обов'язково має бути допустимою. Простіше кажучи, вона не повинна завищувати реальну вартість дороги. Класичний приклад — пошук маршруту на карті, де як h(x) беруть відстань по прямій («як летить птах»). Оскільки пряма — це найкоротший шлях у геометрії, ми гарантовано не переоцінимо реальну дистанцію дорогами.

A* перебирає варіанти поетапно. Він самостійно оцінює кожне відгалуження від старту до фінішу, намагаючись знайти найвигідніший шлях. Як і інші схожі методи «розумного» (інформованого) пошуку, він насамперед хапається за ті напрямки, які здаються найбільш перспективними. Проте, на відміну від простіших алгоритмів (наприклад, Best-First Search), A* не діє наосліп

— він завжди пам'ятає, скільки ресурсів уже було витрачено на пройдений відрізок шляху, і враховує це.

Покроковий процес виглядає так:

– на самому початку алгоритм «оглядається» навколо стартової точки, вираховує значення f(x) для всіх сусідніх вузлів і відкриває той, де ця цифра виявилася найменшою;

– всі знайдені, але ще не перевірені точки, закидаються у спеціальну чергу з пріоритетом. Пріоритетність якраз і визначається формулою f(x)=g(x)+h(x);

– цей цикл повторюється, доки ми не дійдемо до фінішної точки й значення її функції f(x) не стане меншим за будь-який інший варіант у черзі. Ну або поки ми просто не переберемо весь граф. Якщо варіантів фінішу кілька, система автоматично обере найдешевший.

Порада щодо оптимізації: Якщо ми точно знаємо, що вихід є, або якщо в коді прописано механізм захисту від «зациклювання», то можна сміливо ігнорувати вже відвідані вузли. Це круто економить пам'ять і час, перетворюючи звичайний пошук по графу на роботу з деревом рішень.

Властивості та умови оптимальності:

– гарантія результату**:** Алгоритм A* є універсальним — якщо шлях взагалі існує, він його знайде;

– оптимальність рішення: Якщо наша евристика h ніколи не прикрашає дійсність (тобто є допустимою) і ми не викидаємо з пам'яті вже знайдені вершини, то A* залізобетонно видасть найкращий (найкоротший) маршрут.

Проте для ідеальної роботи алгоритму потрібна ще й монотонність (або послідовність) евристичної функції. Що це означає на практиці? Якщо ми маємо три точки — A, B і C, то наше припущення щодо прямого шляху від A до C не може бути більшим, ніж сума оцінок для обхідного маршруту через точку B (тобто A→B→C). Формулою, це можна виразити як для всіх шляхів x, y:

$$g(x) + h(x) \leq g(y) + h(y) \quad (1.1)$$

Окрім усього іншого, алгоритм A* має властивість оптимальної ефективності для обраної евристики h. Це означає, що жоден інший аналогічний алгоритм не зможе знайти правильне рішення, перебравши меншу кількість вузлів, ніж A*.

Цікаво, що класичні методи пошуку в ширину (BFS) та в глибину (DFS) можна вважати просто специфічними модифікаціями A*:

– пошук у глибину (DFS)**:** Його можна імітувати, якщо ввести глобальний лічильник, що стартує з великого числа. Під час аналізу графа кожній сусідній вершині присвоюється поточне значення цього лічильника, яке після кожного кроку зменшується на одиницю. Через це вузли, виявлені раніше, отримують штучно завищену оцінку h(x) і відсуваються в кінець черги, змушуючи алгоритм іти «вглиб»;

– алгоритм Дейкстри**:** Це ще один граничний випадок. Якщо ми повністю відмовимося від прогнозів і обнулимо евристику для всіх точок (h(x)=0), A* перетвориться на класичний алгоритм Дейкстри, який орієнтується виключно на вже пройдену відстань.

На швидкість та логіку роботи A* суттєво впливають деталі його програмування, зокрема те, як саме організовано чергу з пріоритетом, коли виникають однакові оцінки функцій:

– стратегія LIFO («перший прийшов — останній вийшов»): Якщо за однакової вартості пріоритет віддається новішим вузлам, A* локально схилятиметься до пошуку в глибину;

– стратегія FIFO («перший прийшов — перший вийшов»): Якщо першими обробляються старіші вузли, то у спірних ситуаціях алгоритм поводитиметься як пошук у ширину. Цей вибір може кардинально змінити продуктивність програми залежно від структури графа.

Важливо для оптимізації: Щоб після фінішу швидко відновити весь знайдений маршрут, кожна вершина повинна зберігати покажчик (посилання) на свого «батька» — вузол, з якого ми в неї потрапили.

Також критично важливо уникати дублювання вершин у черзі. Зазвичай перед додаванням нового елемента перевіряють, чи він уже там є. Якщо є — запис оновлюють лише тоді, коли новий шлях до цієї точки виявився дешевшим за попередній.

У багатьох стандартних структурах пошук елемента в черзі (в «кроні» дерева) триває лінійно — O(n). Щоб прискорити цей процес до O(1), паралельно з чергою варто використовувати хеш-таблицю.

Продуктивність A* напряму пов'язана з тим, наскільки точною є евристична функція.

– у ідеальному сценарії (якщо евристика бездоганна) алгоритм одразу йде чітко по маршруту без зайвих відгалужень;

– у найгіршому випадку (коли евристика невдала або неінформативна) кількість оброблених вершин починає зростати експоненціально відносно довжини фінального шляху, що сильно навантажує процесор та пам'ять.

Поліноміальною складність є, якщо евристика задовольняє наступну умову:

$$|h(x) - h^*(x)| \leq O(\log h^*(x)) \quad (1.2)$$

h^*- це оптимальна евристика, яка точно визначає відстань від вершини x до мети.

Як і будь-який інший інструмент, цей метод має свої сильні та слабкі сторони, які визначають межі його практичного застосування.

Головні переваги:

– гарантований результат: Якщо шуканий шлях у графі взагалі існує, алгоритм стовідсотково його знайде, причому це буде саме оптимальний (найкоротший) маршрут;

– максимальна вибірковість: Завдяки використанню евристики, A* аналізує та розкриває мінімально можливу кількість вузлів порівняно з іншими класичними методами перебору. Це робить його вкрай цілеспрямованим.

Основні недоліки:

– часова складнощість: Як зазначалося вище, у найгірших сценаріях швидкість обчислень може стрімко падати;

– чутливість до оцінок: Ефективність роботи повністю зав'язана на якості евристичної функції. Невдало підібрана модель прогнозування може перетворити «розумний» пошук на звичайний сліпий перебір;

– критичні вимоги до пам'яті: Це, мабуть, найбільша проблема A*. Алгоритм змушений тримати в оперативній пам'яті величезну (часто експоненціальну) кількість оброблених та очікуючих вершин.

Перешлянемо мурашиний алгоритм (рис. 1.6). В основі цього підходу лежить моделювання природної поведінки мурашиної зграї. Головна мета комах у природі — знайти оптимальний (найкоротший) маршрут між своїм гніздом та джерелом їжі, а також миттєво пристосуватися до будь-яких раптових змін довкілля.

Процес пошуку базується на дуже простому, але ефективному механізмі:

– під час руху кожна мураха маркує свій шлях за допомогою спеціальної хімічної речовини — феромону;

– інші особини з колонії відчувають цей запах і використовують його як орієнтир для вибору власного напрямку;

– чим більше комах проходить конкретною стежкою, тим сильнішим стає аромат, і тим привабливішим цей маршрут стає для решти зграї.

Таким чином, у колонії реалізовано так звану **непряму комунікацію** (або стигмергію). Мурахи не спілкуються між собою напряму, а взаємодіють через

динамічну модифікацію навколишнього середовища, що дозволяє їм діяти як єдиний злагоджений організм.

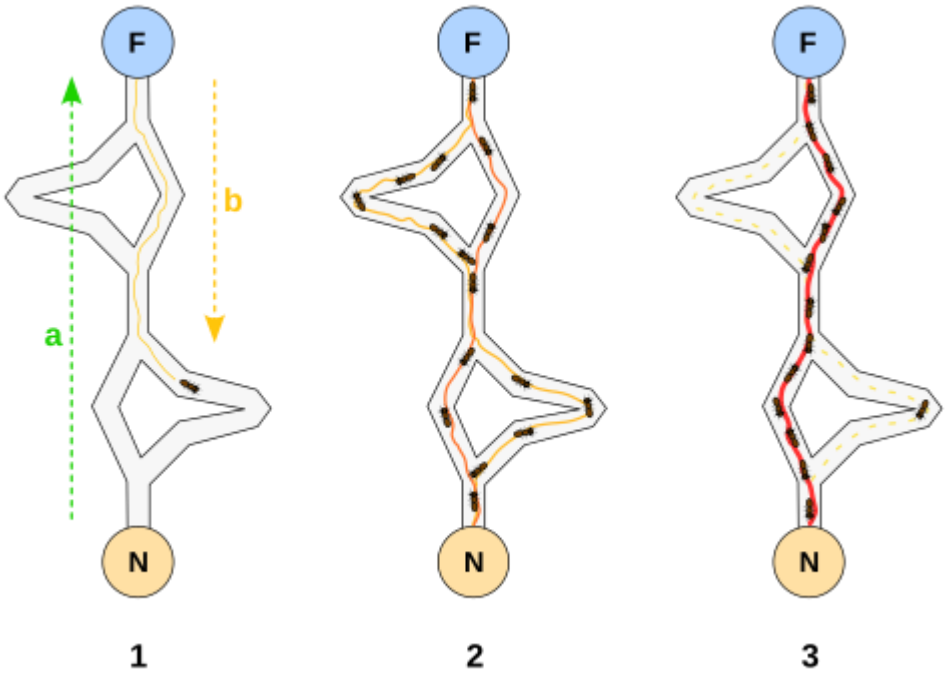

Рисунок 1.6 – Приклад роботи мурашиного алгоритм

Якщо проектувати цю модель на мову теорії графів, де кожне ребро є потенційним відрізком шляху, стає очевидним, що система тримається на балансі двох сил:

– позитивний зворотний зв'язок: Проявляється у поступовому накопиченні феромону на тих ділянках, якими комахи ходять найчастіше. Це змушує наступні покоління «агентів» обирати саме ці ребра;

– негативний зворотний зв'язок (випаровування): З часом концентрація хімічного сліду на дорогах природно зменшується.

Саме механізм випаровування відіграє ключову роль у гнучкості алгоритму. Якби запах не зникав, система майже миттєво зациклювалася б на першому-ліпшому знайденому маршруті. Завдяки ж вивітрюванню феромону, алгоритм не

«застрягає» в локальних оптимумах, а продовжує досліджувати альтернативні траєкторії графа. Це підтримує необхідний рівень різноманітності пошуку.

Фінальний результат: Найкращим (оптимальним) розв'язком завдання вважається той ланцюжок ребер графа, на якому до моменту зупинки алгоритму сформувалася найвища щільність феромонового сліду.

Flutter — це сучасний фреймворк, який дозволяє створювати кросплатформенні застосунки з єдиною кодовою базою, зосереджуючись на високій продуктивності та візуальній узгодженості інтерфейсу**.** Його головна особливість полягає в тому, що він забезпечує нативну продуктивність, використовуючи власний рушій для рендерингу графіки, не покладаючись на веб-переглядач або нативні компоненти платформи (рис. 1.7).

Рисунок 1.7 – Мова програмування Flutter

Основною мовою програмування у Flutter є **Dart** — мова, розроблена компанією Google, що поєднує об’єктно-орієнтований підхід з сучасними засобами асинхронного програмування. Dart-компіляція у машинний код дозволяє досягати високої швидкодії на Android, iOS, web та desktop-платформах. Це дозволяє розробникам створювати застосунки, які працюють швидко та стабільно навіть на пристроях із низькою продуктивністю.

Модель рендерингу Flutter заснована на власному графічному рушії Skia. Це дозволяє відображати елементи інтерфейсу без прив'язки до компонентів операційної системи, забезпечуючи стабільність вигляду на різних пристроях і версіях ОС. Компоненти інтерфейсу у Flutter, які називаються віджетами, описують не тільки структуру, а й поведінку елементів. Віджети можна комбінувати, модифікувати, а також створювати кастомні, що робить Flutter особливо гнучким для дизайнерів і розробників.

*Архітектура роботи***.** Flutter забезпечує реактивний підхід до побудови інтерфейсів, подібно до React. Зміни у стані викликають перебудову інтерфейсу, що дозволяє легко синхронізувати вигляд додатку з його логікою. Кожен користувацький інтерфейс складається з дерева віджетів, де кожен віджет може мати дочірні елементи. Взаємодія з користувачем чи зміна даних викликає оновлення відповідних гілок цього дерева, що значно знижує навантаження на систему.

*Особливості реалізації***.** Завдяки компіляції в нативний код, Flutter уникає багатьох недоліків гібридних рішень. Також він має гарну підтримку hot-reload — функції, яка дозволяє миттєво бачити зміни в коді без перезавантаження застосунку. Це пришвидшує розробку та тестування.

Продуктивність додатків, створених у Flutter, часто порівнюють з нативними застосунками. Водночас існує деяка залежність від розміру графічного рушія та бібліотек, що можуть збільшити розмір застосунку. Але завдяки активній розробці та спільноті, ці недоліки поступово зменшуються.

Переваги Flutter включають:

– єдина кодова база для багатьох платформ;
– висока продуктивність завдяки компіляції у машинний код;
– гнучкий інтерфейс побудований на віджетах;
– активна підтримка від Google;
– можливість створення гарно анімованих UI без значних витрат ресурсів.

Недоліки**:**

– розмір додатка може бути більшим, ніж у нативних рішень;

– підтримка платформ, відмінних від Android/iOS (наприклад, web або desktop), ще перебуває в стадії вдосконалення;

– залежність від Dart як основної мови — що потребує її окремого вивчення.

Node.js — це середовище виконання JavaScript, орієнтоване на розробку масштабованих та високопродуктивних серверних застосунків, зосереджуючись на подієво-орієнтованій архітектурі та неблокуючому введенні/виведенні. Його основна ідея полягає у виконанні JavaScript-коду поза межами браузера, що дозволяє створювати серверну логіку, веб-сервіси та мережеві додатки, використовуючи одну мову — JavaScript (рис. 1.8).

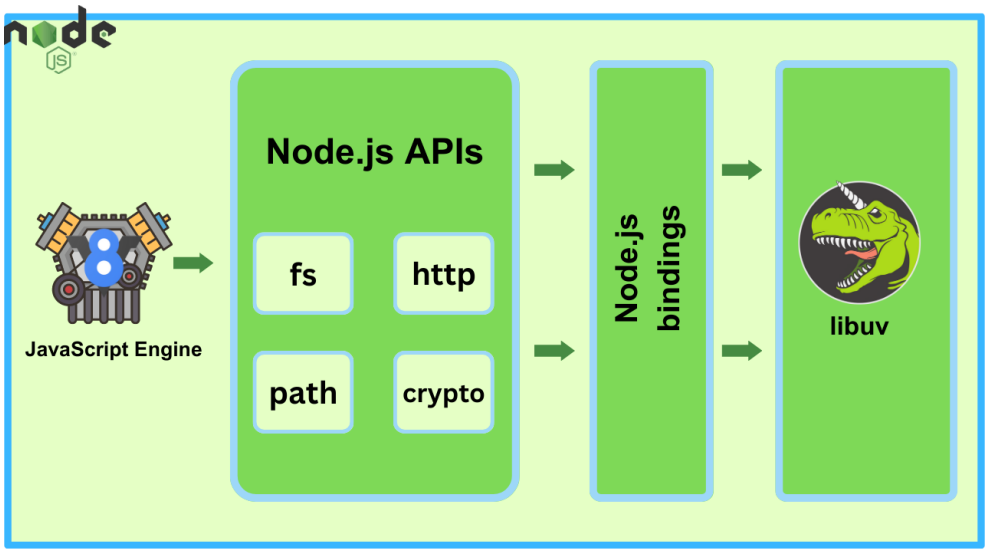

Рисунок 1.8 – Мова програмування Node.js

Node.js працює на основі високопродуктивного рушія V8, розробленого Google, який компілює JavaScript у нативний машинний код. Завдяки цьому досягається швидке виконання коду та мінімальні затримки при обробці запитів. Основною особливістю Node.js є асинхронна модель роботи, яка забезпечує

неблокуючий підхід до виконання операцій введення/виведення, таких як читання з файлу, доступ до бази даних чи робота з мережею.

*Архітектура роботи***.** Node.js використовує однопоточну модель з циклом подій (event loop), яка керує асинхронними викликами. Це означає, що всі запити обробляються у межах одного потоку, а завдяки неблокуючим функціям, події обробляються паралельно у фоновому режимі без необхідності створення додаткових потоків. В результаті сервер на Node.js може одночасно обробляти тисячі клієнтських з'єднань, не витрачаючи багато ресурсів на управління потоками.

Ця модель ефективна для високонавантажених застосунків у режимі реального часу, таких як чати, ігрові сервери, стрімінгові сервіси чи REST API, де ключовою є швидкість обробки запитів.

*Особливості реалізації***.** Бібліотека fs дозволяє працювати з файловою системою, http — створювати власні HTTP-сервери, а пакетний менеджер npm забезпечує доступ до сотень тисяч модулів, які значно прискорюють розробку. Поширені фреймворки, такі як Express.js, будуються поверх Node.js, забезпечуючи зручні API для створення веб-застосунків.

Node.js також підтримує стріми (потоки) — структури, які дозволяють обробляти дані по частинах, що значно знижує споживання пам'яті. Наприклад, сервер може відправляти відео не повністю, а порціями, зменшуючи навантаження.

Переваги Node.js:

– висока масштабованість і продуктивність завдяки неблокуючій моделі;

– можливість використання JavaScript на сервері та клієнті (повний стек);

– активна екосистема npm;

– швидке прототипування завдяки простоті API;

– ідеальне середовище для застосунків у реальному часі.

Недоліки:

– однопоточна модель ускладнює обробку обчислювально важких задач;

– слабша підтримка для CPU-інтенсивних задач;

– відсутність стандартного способу роботи з багатопоточністю (хоча з часом з'явилися worker threads);

– можлива складність у керуванні вкладеною асинхронністю (callback hell, хоча це значною мірою вирішено через async/await).

Алгоритм Джонсона (рис. 1.9) ефективний метод пошуку найкоротших відстаней між абсолютно всіма парами вершин (All-Pairs Shortest Paths) у зваженому орієнтованому графі. Головна його фішка полягає в універсальності: він чудово порається як із додатними, так і з від'ємними вагами ребер. Єдина принципова обмеженість — у графі не повинно бути циклів від'ємної вартості, адже за їх наявності задача взагалі втрачає сенс.

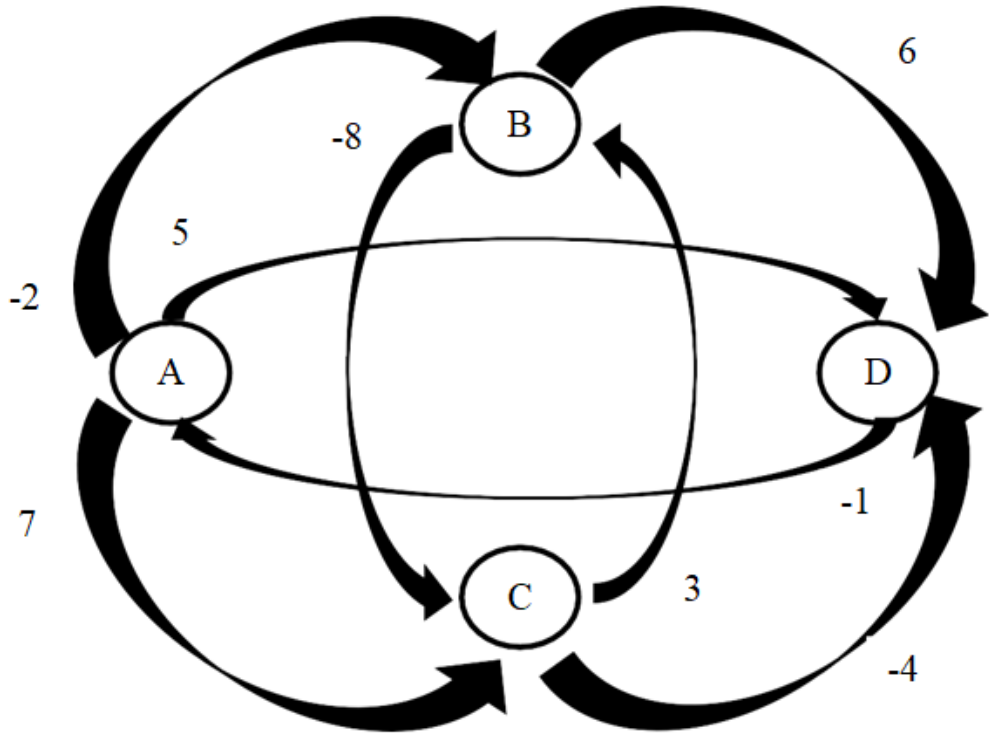

Рисунок 1.9 – Алгоритм Джонсона на орграфі з від'ємними дугами

Логіка роботи алгоритму базується на комбінації двох класичних підходів і залежить від початкових умов:

– сценарій без від'ємних ваг: Якщо граф G=(V,E) з ваговою функцією ω спочатку не містить ребер «у мінус», нам не потрібно вигадувати нічого складного. Достатньо просто запустити стандартний алгоритм Дейкстри послідовно для кожної вершини графа;

– сценарій з від'ємними вагами: Якщо ж від'ємні ребра є, алгоритм робить хитрий трюк — він математично змінює (перераховує) ваги всіх ребер так, щоб вони стали виключно невід'ємними. Після цього знову стає можливим безпечний запуск алгоритму Дейкстри для кожної точки.

Головна математична вимога до нових ваг ребер — вони мають бути коректними. Це означає виконання двох залізобетонних умов:

– невід'ємність: Після трансформації для будь-якого ребра (u,v) його нова вага має бути більшою або рівною нулю (ω^(u,v)≥0);

– збереження структури: Зміна чисел не повинна зламати логіку маршрутів. Тобто, найкоротший шлях між точками u та v, знайдений за новими вагами, має залишатися найкоротшим і за старих початкових умов. Змінюється лише числове вираження вартості, а не сама геометрія оптимального треку.

Математична база: Для реалізації такого перерахунку вводиться допоміжна потенційна функція вершин h:V→R, яка відображає кожен вузол графа на дійсне число (на практиці ці потенціали знаходять за допомогою алгоритму Белмана-Форда). Для кожного ребра $(u, v) \in E$ визначаємо:

$$\hat{\omega}(u, v) = \omega(u, v) + h(u) - h(v) \quad (1.3)$$

Для будь-якого довільного маршруту p=⟨v0,v1,…,vk⟩, що з'єднує початкову вершину v0 із кінцевою vk, зміна вагових коефіцієнтів не порушує топологію

найкоротших шляхів. Математично це означає, що шлях p залишається оптимальним (найкоротшим) після перерахунку ваг ω^ тоді і тільки тоді, коли він був таким і за початкової функції ω.

Формально, рівність вартості шляху мінімальній відстані за вихідних умов ω(p)=δ(v0,vk) є повністю еквівалентною аналогічному співвідношенню для модифікованих ваг:

$$\omega\hat{}(p)=\delta\hat{}(v0,vk) \quad (1.4)$$

Крім того, цей метод трансформації гарантує збереження структури графа щодо критичних помилок: граф G містить цикл із від'ємною вагою за початкових умов ω тоді і тільки тоді, коли цей самий цикл залишається від'ємним і після розрахунку нових ваг ω^. Тобто алгоритм не може випадково "створити" або "замаскувати" заборонені від'ємні цикли.

Практична реалізація зміни ваг (алгоритм розширення) відбувається за такою схемою:

– створення фіктивної вершини: На основі вихідного графа G=(V,E) будується допоміжний граф G′=(V′,E′). До початкової множини вершин додається одна нова ізольована точка s (V′=V∪{s});

– формування нових зв'язків: Множина ребер розширюється за рахунок спрямованих дуг від фіктивного вузла s до абсолютно всіх наявних вершин вихідного графа (E′=E∪{(s,v):v∈V});

– ініціалізація стартових ваг: Для всіх новостворених ребер, що виходять із точки s, задається нульова вартість:

$$\omega(s,v)=0 \text{для всіх} v\in V \quad (1.5)$$

– розрахунок потенціалів (функції h): Використовуючи алгоритм Белмана-Форда від стартової точки s, для кожного вузла v∈V обчислюється величина потенціалу, яка дорівнює найкоротшій відстані від s до цієї вершини:

$$h(v)=\delta(s,v) \qquad (1.6)$$

– фінальний перерахунок ребер: Нові невід'ємні ваги для кожного базового ребра (u,v) обчислюються за формулою:

$$\omega\hat{}(u,v)=\omega(u,v)+h(u)-h(v)\geq 0 \qquad (1.7)$$

Попри свою витонченість та універсальність, алгоритм Джонсона має суттєвий мінус — це його часова складність. Оскільки підготовчий етап вимагає запуску ресурсомісткого алгоритму Белмана-Форда (для обчислення потенціалів h), а фінальний етап передбачає багаторазовий (за кількістю вершин |V|) перезапуск алгоритму Дейкстри, загальний час виконання програми на великих або щільних графах може бути надто великим, що обмежує його використання в системах реального часу.

2 ОСОБЛИВОСТІ ЗАСТОСУВАННЯ АЛГОРИТМІВ В ВЕБПЛАТФОРМІ ОРЕНДИ ЕЛЕКТРОТРАНСПОРТУ

2.1 Принцип природного алгоритму

У дикій природі багато видів комах (зокрема, мурахи, терміти, окремі представники бджіл та ос) об'єднуються у величезні спільноти. Життя в колонії базується на феномені колективного інтелекту: разом вони здатні розв'язувати складні оптимізаційні задачі, які абсолютно не під силу окремому індивіду. До таких завдань належать ефективний розподіл праці всередині гнізда, його кластеризація та облаштування, а головне — розрахунок найкоротших маршрутів до джерел їжі.

Щоб така чисельна група діяла як єдиний організм, комахам необхідна чітка система комунікації, яка, залежно від біологічного виду, поділяється на два типи:

– пряма комунікація: Базується на безпосередньому контакті між особинами. Класичним прикладом є «танець» бджіл: розвідниця, яка знайшла нектар, за допомогою специфічних рухів передає іншим бджолам координати та відстань до їжі. Щоб отримати цю інформацію, інші члени рою мають наочно бачити цей танець. До цієї ж категорії належать фізичні дотики, обмін кормом (трофалаксис) або сигнальними рідинами;

– непряма комунікація: Полягає у взаємодії через модифікацію навколишнього простору. Комаха змінює середовище таким чином, що це автоматично коригує поведінку її наступників. Найяскравіший приклад — феромонові траси мурах. Повертаючись ізі здобиччю, мураха маркує землю хімічним слідом, який спонукає інших особин йти тією ж дорогою.

Процес, за якого діяльність однієї особини стимулює активність іншої через зміну довкілля, у науці називають стигмергією. Саме цей механізм є фундаментом для самоорганізації — явища, коли з хаотичних дій простих агентів (окремих комах) народжується складна і доцільна групова поведінка.

Розподілений характер самоорганізації робить колонію неймовірно життєздатною та ефективною. Система працює стабільно і без збоїв, навіть якщо частина мурах тимчасово випадає з процесу або не бере у ньому участі — тут немає єдиного керівного центру, а логіка дій закладена в локальних взаємодіях.

Щоб детально вивчити, як саме мурашина спільнота прораховує найкращі маршрути, вчені провели низку лабораторних досліджень.

Під час першого тесту аргентинським мурахам запропонували два абсолютно однакові за довжиною шляхи до годівниці. Спочатку комахи розділилися порівну, проте за короткий проміжок часу вся колонія сфокусувалася лише на одному варіанті, повністю проігнорувавши альтернативний. Це сталося через випадкове коливання: на одній із гілок на початку опинилося трохи більше мурах, вони залишили більше феромону, що лавиноподібно привабило решту зграї.

Для перевірки здатності колонії обирати саме *найкоротшу* траєкторію, умови ускладнили за допомогою експерименту «подвійний міст». Перед комахами поставили вибір між довгою та короткою дорогами.

Оскільки короткий шлях мурахи долали швидше, вони поверталися ним частіше за одиницю часу. Відповідно, концентрація феромону на короткому «мосту» зростала в рази швидше, ніж на довгому. У результаті позитивного зворотного зв'язку колонія за лічені хвилини повністю перемкнулася на геометрично найоптимальніший маршрут (рис. 2.1).

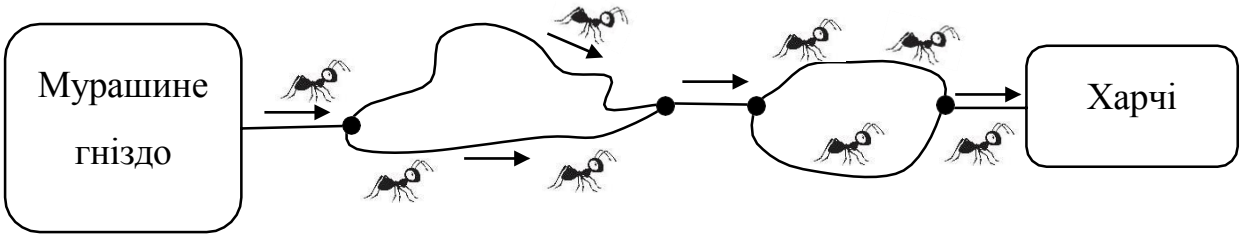

Рисунок 2.1 – Подвійний експеримент моста [24]

Оскільки деякі мурахи є практично сліпими, вони не спроможні візуально оцінити геометрію простору. Попри це, колонія демонструє дивовижну здатність безпомилково визначати найкоротший маршрут між гніздом і джерелом їжі.

Цей процес відбувається за принципом самоорганізації:

– початковий хаос: На старті всі комахи перебувають у гнізді. Коли перша група вирушає на пошуки їжі, кожна особина залишає на землі хімічний маркер — феромон. Дійшовши до першої розвилки (точки A), агенти, не маючи жодних орієнтирів, розділяються випадковим чином: половина повертає ліворуч, інша половина — праворуч;

– ефект швидкості (розвилка B)**:** Ті, хто обрав коротшу гілку, досягають наступного перехрестя (B) значно швидше. Не маючи інформації про подальший рух, частина з них біжить далі до їжі, а частина повертає назад до гнізда;

– асиметрія концентрації: Комахи, які пішли довгим шляхом, теж зрештою дістаються точки B. Проте на той момент на короткій ділянці вже встигло пробігти значно більше комах (в обох напрямках), тому концентрація запаху там у рази вища. Наступні хвилі фуражирів, орієнтуючись на сильніший аромат, інстинктивно обирають саме короткий відрізок;

– каскадне посилення: Описана логіка повторюється на всіх наступних ділянках (наприклад, між точками C і D). Оскільки на короткому маршруті інтенсивність виділення феромону вища, система починає самопосилюватися. З кожним новим колом короткий шлях стає дедалі привабливішим, повністю затьмарюючи альтернативні варіанти.

Роль негативного зворотного зв'язку: Феромон має властивість поступово випаровуватися. Це критично важливо, адже довгі або неефективні маршрути, якими комахи майже не ходять, швидко втрачають свій хімічний слід. Вивітрювання запобігає зацикленню алгоритму на застарілих даних і стимулює вибір оптимізованих траєкторій.

Історично першим серйозним випробуванням для мурашиного алгоритму стала класична задача комівояжера, де необхідно знайти найкоротший замкнений обхід кількох міст. Пряма аналогія між пошуком фізичного шляху в природі та пошуком мінімального ребра на графі рішень дозволила ефективно адаптувати цей біологічний підхід.

З-поміж перших концепцій було запропоновано три варіанти оновлення феромонового сліду на ребрах:

– щільнісний (Density) метод: Агенти оновлюють рівень феромону безпосередньо в процесі переходу з одного вузла до іншого;

– кількісний (Quantity) метод: Обсяг залишеного ферменту залежить від довжини поточного кроку;

– циклічний (Cycle) метод: Мурахи маркують ребра лише після того, як повністю сформують замкнений маршрут і буде оцінено його загальну вартість.

Експерименти довели абсолютну перевагу **циклічного підходу**. Два інші методи виявилися малоефективними і були відкинуті, тому сьогодні під класичною «мурашиною системою» за замовчуванням розуміють саме циклічний алгоритм.

Для забезпечення об'єктивного пошуку на початку кожної ітерації популяція мурах рівномірно розподіляється по всіх вузлах мережі. Це гарантує, що кожне місто має рівні шанси стати стартовою точкою, оскільки заздалегідь визначити найбільш вигідний початок обходу неможливо.

Коли мураха k перебуває в місті i, ймовірність її переходу в наступний пункт j визначається трьома основними чинниками:

– пам'ять агента (Список заборон / Tabu List): Це індивідуальний бітовий масив, який фіксує вже відвідані міста. Алгоритм вимагає, щоб кожна вершина обходилася суворо один раз. Поточна траєкторія записується у вектор Path, а множина міст, які мурасі k ще належить відвідати з точки i, позначається як J(i,k). Після завершення повної ітерації цей список повністю очищується;

– видимість (Евристичне наближення ηij)**:** Цей параметр є оберненим до фізичної відстані між вузлами і та j:

$$\eta_{ij} = \frac{1}{d_{ij}} \quad (2.1)$$

Це локальний статичний показник, який змушує мураху віддавати перевагу найближчим сусіднім містам (жадібна стратегія). Проте самої лише видимості недостатньо для пошуку глобального оптимуму;

– віртуальний слід феромону (τij): Динамічний показник, що відображає накопичений досвід усієї колонії щодо доцільності руху за ребром (ij). Обсяг феромону, який додається на цьому відрізку після завершення циклу, прямо пропорційний якості (короткості) згенерованого мурахою маршруту. Це забезпечує колективне навчання системи та спрямовує пошук у бік найкращих рішень.

2.2 Вирішення задач із застосуванням мурашиного алгоритму

Завдяки своїй гнучкості та здатності працювати у динамічних середовищах, концепція мурашиних колоній (ACO) вийшла далеко за межі класичної задачі комівояжера. Сьогодні цей підхід успішно розв'язує складні комбінаторні задачі у різних галузях.

Планування та розклад робіт (Job Scheduling Problem)

Суть задачі полягає в оптимальному розподілі черги технологічних завдань між наявними ресурсами — виробничими потужностями, верстатами або персоналом. Головна мета — мінімізувати сумарний час виконання всього обсягу робіт (makespan), знизити фінансові витрати або уникнути затримок.

– реалізація моделі: Кожен окремий ресурс (наприклад, машина) інтерпретується як відрізок маршруту, яким рухаються віртуальні агенти;

– результат: Кожна мураха формує свій варіант графіку, намагаючись оптимізувати цільову функцію. З кожною новою ітерацією ефективні схеми розподілу завдань отримують більше феромону, що зрештою приводить систему до найбільш збалансованого та вигідного розкладу.

Маршрутизація транспорту (Vehicle Routing Problem, VRP)

Ця проблема є прямим, але значно ускладненим розширенням задачі комівояжера. Тут необхідно скоординувати роботу цілого логістичного автопарку так, щоб розвести товари групі клієнтів із мінімальним сумарним пробігом. При цьому враховуються жорсткі обмеження: вантажопідйомність машин, об'єм кузова та узгоджені з клієнтами часові коридори (time windows) для доставки.

– реалізація моделі: Комахи самостійно прокладають індивідуальні траєкторії для кожної одиниці техніки;

– результат: Спираючись на баланс накопиченого феромону та локальних евристичних підказок (наприклад, географічної близькості точок), агенти швидко знаходять оптимальні або близькі до них логістичні ланцюжки. Корекція хімічного сліду звужує зону пошуку до найбільш перспективних географічних напрямків.

Динамічне керування трафіком у телекомунікаціях

У сучасних мережах передачі даних критично важливо оперативно визначати найкращі шляхи для пакетів інформації, щоб мінімізувати затримки (ping), уникати втрати пакетів та максимізувати загальну пропускну здатність каналів.

– реалізація моделі: Замість статичних таблиць тут працюють мобільні цифрові агенти. Вони безперервно мігрують мережею, аналізуючи реальний стан ліній зв'язку, швидкість відгуку та рівень завантаженості конкретних вузлів;

– результат: На найшвидших і найменш завантажених напрямках агенти залишають віртуальні феромони. Це дає змогу мережевому обладнанню миттєво адаптуватися до раптових стрибків інтернет-трафіку чи аварій на лініях,

перенаправляючи потоки даних оптимізованими маршрутами в режимі реального часу.

Кластеризація та інтелектуальний аналіз даних

У сфері машинного навчання (Data Mining) існує фундаментальна задача розподілу масиву об'єктів за групами (кластерами). Головна умова — елементи всередині одного кластера мають бути максимально схожими, а між різними групами — кардинально відрізнятися.

– реалізація моделі: Кожен об'єкт аналізу розглядається як окремий вузол графа, між якими штучні мурахи прокладають зв'язки;

– результат: Феромонний слід у цій моделі виступає мірою метричної схожості: чим ближчі характеристики об'єктів, тим сильніше зафарбовується ребро між ними. Крок за кроком колонія групує елементи навколо стійких центрів тяжіння, формуючи чіткі та природні кластери.

2.3 Задачі комівояжера

Задача комівояжера (Travelling Salesperson Problem, TSP) — це одна з найбільш фундаментальних та складних проблем у галузі комбінаторної оптимізації. Її класичне формулювання доволі просте: є певна множина міст (або точок), які з'єднані між собою дорогами відомої довжини (ваги). Необхідно прокласти такий замкнений маршрут, за якого мандрівник відвідає кожне місто суворо один раз і наприкінці повернеться у вихідну точку, а загальна дистанція (або фінансові витрати) буде мінімальною.

Попри простоту опису, математична природа задачі є надзвичайно підступною через дві ключові особливості:

– комбінаторний вибух: Кількість можливих варіантів обходу міст зростає факторіально (або експоненціально, залежно від умов задачі) із додаванням кожної нової точки. Якщо для 5 міст варіантів перебору зовсім небагато, то для 50 міст кількість комбінацій перевищує кількість атомів у Всесвіті;

– NP-повна складність: На сьогодні не існує (і теоретично не може існувати) точного алгоритму, який був би здатний знайти ідеальне рішення для будь-якої великої кількості вхідних даних за прийнятний, реальний час. Перебір усіх комбінацій «в лоб» швидко паралізує навіть найпотужніші суперкомп'ютери;

– відсутність єдиного розв'язку: У TSP часто немає одного унікального маршруту. Різні алгоритми за однакових умов можуть знаходити абсолютно різні геометрії шляхів, які при цьому матимуть однакову сумарну вартість (довжину).

Через високу обчислювальну складність дослідники розділили підходи до вирішення TSP на дві великі групи:

– точні методи (наприклад, метод гілок та меж): Гарантують знаходження математично бездоганного, найкоротшого шляху. Проте їх можна ефективно застосовувати лише для невеликих графів (зазвичай до кількох сотен чи тисяч точок, залежно від потужності техніки). На масштабних базах даних вони стають абсолютно непрактичними;

– метаевристичні та наближені методи: Сюди відносять генетичні, мурашині та імунні алгоритми, метод імітації відпалу тощо. Вони не дають стовідсоткової гарантії, що знайдений шлях є абсолютно найкращим із можливих, але дозволяють за лічені секунди чи хвилини знайти якісне, близьке до оптимального рішення, що повністю задовольняє практичні потреби;

– гібридні системи (AI + ML): Новітнім трендом є інтеграція оптимізаційних алгоритмів із методами машинного навчання та штучного інтелекту. Нейромережі допомагають миттєво відсікати завідомо неефективні напрямки, суттєво прискорюючи метаевристичний пошук у надскладних умовах.

2.4 Фронтенд-збірка vite.js

Тривалий час золотим стандартом для збірки вебдодатків (особливо на React, Vue чи Angular) був Webpack. Проте із розростанням проєктів розробники зіштовхнулися із серйозною проблемою: запуск локального сервера для розробки

(dev server) та кожна дрібна зміна в коді (HMR — Hot Module Replacement) починали тривати хвилинами.

Vite.js (від французького «швидко», вимовляється як */віт/*) — це інструмент збірки нового покоління, створений Еваном Ю (автором Vue.js), який кардинально змінив правила гри. Замість того, щоб намагатися оптимізувати старі підходи, Vite повністю переосмислив логіку роботи з кодом під час розробки, розділивши весь процес на два абсолютно різні етапи.

Головний секрет швидкості Vite полягає у використанні сучасних браузерних стандартів та надшвидких низькорівневих інструментів. Математика його продуктивності тримається на трьох китах:

– нативні ES-модулі (ESM): Webpack перед запуском сервера змушений повністю змапити та «склеїти» (bundle) весь ваш проєкт в один великий файл. Vite під час розробки цього взагалі не робить. Він віддає браузеру сирий вихідний код через нативні import / export. Браузер сам аналізує, який саме файл йому потрібен у цей момент, і Vite підвантажує лише його. Нульова робота на старті — миттєвий запуск;

– попереднє збирання залежностей через esbuild: Ваші залежності з node_modules (наприклад, величезні бібліотеки типу lodash чи react) змінюються вкрай рідко, але містять тисячі дрібних модулів. Vite один раз на початку «спресовує» їх за допомогою компілятора esbuild. Оскільки esbuild написаний мовою Go, він обробляє код у 10–100 разів швидше, ніж традиційні збирачі на Node.js;

– розумне кешування та HMR: Коли ви змінюєте один рядок у коді, Vite оновлює лише цей конкретний модуль. Зв'язок із браузером тримається через сокети, а старі модулі інвалідуються в кеші миттєво. Швидкість оновлення сторінки взагалі не залежить від загального розміру вашого проєкту.

Як і будь-яка інженерна технологія, Vite не є абсолютною панацеєю, хоча плюси суттєво переважують мінуси.

Переваги:

– старт за мілісекунди: Навіть проєкт на кілька тисяч компонентів запускається локально майже миттєво;

– конфігурація «з коробки» (Out-of-the-box): Він одразу без додаткових плагінів розуміє TypeScript, JSX, CSS-модулі та препроцесори (Sass/Less);

– екосистема плагінів: Vite побудований на базі архітектури Rollup, тому він сумісний із більшістю плагінів від Rollup, що дає величезний вибір готових рішень;

– універсальність: Не прив'язаний до конкретного фреймворку — однаково круто працює з Vue, React, Svelte, Solid чи чистим Vanilla JS.

Недоліки та нюанси:

– різниця середовищ: Оскільки в розробці працює esbuild, а на продакшні — Rollup, у дуже рідкісних випадках специфічний баг може вилізти саме після фінальної збірки, хоча на локалці все працювало ідеально;

– проблеми з застарілим кодом: Якщо ваш проєкт сильно залежить від старих бібліотек, які використовують формат CommonJS (require()), Vite доведеться додатково конфігурувати для їхньої трансформації в ESM, що інколи викликає труднощі.

2.5 Базовий мурашиний алгоритм для розв'язання задачі пошуку маршруту

Сучасні високопродуктивні обчислення часто спираються на стратегії, запозичені з живої природи. Окремий клас у цій сфері становлять еволюційні та біонічні підходи. Якщо перші (наприклад, генетичні алгоритми) копіюють механізми спадковості та мутацій людського геному, то другі моделюють поведінкові патерни фауни. Головна мета таких алгоритмів — знайти компромісні, економічно вигідні розв'язки для математичних задач, які є занадто складними для класичних аналітичних методів перебору.

На стику цих підходів сформувався напрям метаевристик (від грецьких *meta* — «понад / вище» та *heuriskein* — «знаходити»). Метаевристика — це

універсальна алгоритмічна архітектура високого рівня, яка не прив'язана до специфіки конкретної задачі, а задає загальну стратегію для побудови евристичних методів.

Яскравими представниками цього класу є алгоритми ройового інтелекту (Swarm Intelligence), зокрема метод рою часток (PSO) та мурашина оптимізація (ACO). Вони описують поведінку децентралізованих багатоагентних систем, натхненних колективними діями соціальних комах (термітів, бджіл, ос, мурах) або зграй птахів і риб.

Штучні мурашині колонії, вперше запропоновані Марко Доріго на початку 1990-х років, повністю відтворюють природну стратегію фуражування. Мурахи належать до еусоціальних істот, де інтереси виживання всієї спільноти домінують над потребами окремої особини. У таких кооперативних групах діє чіткий розподіл ролей, де більшість членів колонії є безплідними робочими особинами, які забезпечують захист, догляд за потомством та пошук продовольства.

Координація дій всередині колонії відбувається за допомогою звуків, дотиків та феромонів — летких органічних сполук, що виділяються в навколишнє середовище і провокують специфічну поведінкову реакцію у сородичів. Оскільки робочі особини переміщуються переважно по землі, вони маркують поверхню ґрунту хімічним слідом, який слугує динамічним дороговказом для решти зграї.

Базовий принцип мурашиного алгоритму (Ant Colony Optimization) полягає в моделюванні пошуку найкоротшого шляху між гніздом і джерелом їжі на графі. У контексті задачі комівояжера такий маршрут представляє собою послідовність суміжних ребер, де фініш попереднього відрізка є стартом для наступного, а загальна довжина визначається сумою ваг цих ребер.

Процес формування оптимальної траєкторії зведено до чотирьох послідовних фаз:

– стартовий хаос: У початковий момент часу всі агенти перебувають у гнізді. Віртуальне середовище чисте — будь-які хімічні маркери на ребрах графа

відсутні. Комахи починають хаотичний пошук, обираючи напрямки абсолютно випадково;

– первинне маркування: Випадковий перебір відкриває кілька альтернативних маршрутів до цілі. Повертаючись із їжею назад, агенти починають секретувати феромон на пройдених ребрах. Для спрощення уявимо дві альтернативні гілки — коротку і довгу;

– фактор часу та швидкості: Оскільки коротку дистанцію агенти долають швидше, за одиницю часу вони встигають зробити більше рейсів туди й назад. Як наслідок, концентрація хімічного маркеру на короткій ділянці починає зростати значно швидше, ніж на довгій. Це автоматично підвищує ймовірність вибору цього шляху наступними поколіннями фуражирів;

– стабілізація та витіснення (Кристалізація розв'язку): Завдяки позитивному зворотному зв'язку більшість популяції за короткий час концентрується на геометрично найкоротшому маршруті. Паралельно запускається негативний зворотний зв'язок — випаровування феромону. Оскільки довга гілка використовується дедалі рідше, інтенсивність запаху на ній падає через природне вивітрювання, поки траса повністю не зникне. Система самоочищується від неефективних рішень, фіксуючи глобальний оптимум.

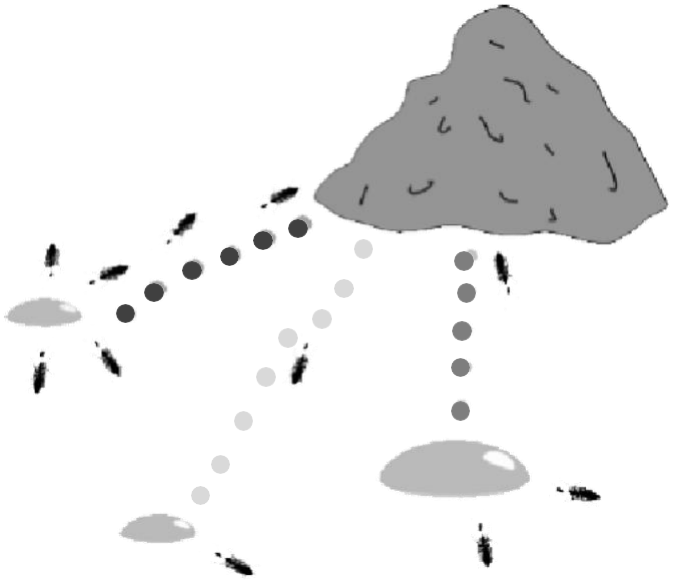

Рисунок 2.2 – Схематичне зображення роботи мурашиного алгоритму[25]

Описана поведінкова стратегія природних колоній слугує ідеальним базисом для проектування архітектури штучних оптимізаційних алгоритмів. Якщо абстрагуватися від біологічних деталей та спростити модель до одного гнізда й одного джерела їжі, з'єднаних парою альтернативних доріг, то цей сценарій легко формалізується за допомогою апарату теорії графів.

Трансформація природних сутностей у цифрові компоненти відбувається за такою схемою:

– вершини (вузли) графа: Відображають ключові географічні точки — безпосередньо мурашине гніздо (стартова позиція) та локацію з продовольством (цільова точка);

– ребра графа: Уособлюють собою доступні фізичні маршрути або варіанти переміщення між цими об'єктами;

– ваги ребер (динамічний параметр): Моделюють поточну концентрацію феромону на кожній ділянці. Ця величина постійно оновлюється в процесі роботи програми: вона інкрементується (збільшується), коли штучний

агент успішно долає ребро, і декрементується (зменшується) з кожною ітерацією внаслідок математичного моделювання процесу випаровування.

Саме через таку динамічну зміну вагових коефіцієнтів система зваженого графа здатна до самонавчання, що в підсумку й дозволяє обчислити глобально оптимальний маршрут.

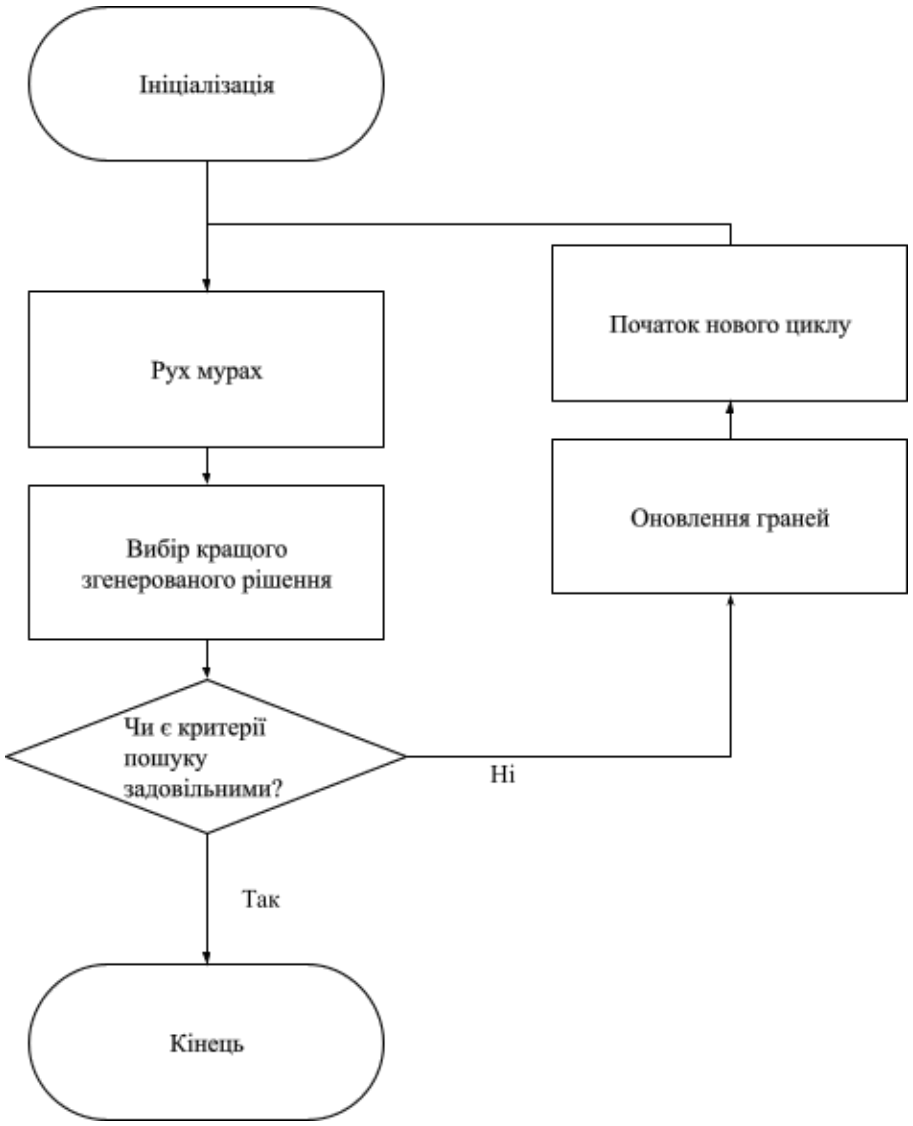

Рисунок 2.3 – Схема роботи мурашиного алгоритму[26]

Ось перероблений варіант цього математичного опису. Текст адаптовано під суворий, але природний стиль науково-дослідницької роботи (диплому чи статті), усунуто дрібні граматичні та логічні помилки вихідного тексту (наприклад, фразу

«для вершин E1 і E2», хоча E — це ребра), а всі змінні оформлено через стандартизовані математичні теги.

Для побудови математичної моделі уявимо неорієнтований граф G=(V,E), де V — множина вершин (вузлів), а E — множина ребер (зв'язків). Відповідно до базових умов нашого дослідження, задамо такі параметри системи:

– цільові вузли (V): Множина вершин містить дві ключові точки: Vs (виступає як джерело або стартова локація віртуальної колонії) та Vd (кінцева точка призначення, що моделює джерело продовольства);

– метричні ребра (E): Зв'язок між вузлами Vs та Vd забезпечується двома альтернативними ребрами — E1 та E2. Кожному з цих ребер присвоєно ваговий коефіцієнт геометричної довжини — L1 та L2 відповідно;

– показники інтенсивності феромону (R): Динамічна щільність хімічного сліду, яка визначає поточну привабливість маршруту для агентів, задається як змінні R1 (для ребра E1) та R2 (для ребра E2).

Така формалізація дозволяє перевести біологічні вектори поведінки комах у площину числових ітераційних обчислень, де ймовірність вибору ребра буде прямо пропорційною значенню R та обернено пропорційною коефіцієнту L.

Таким чином утворюється рівняння (2.2):

$$P_i = \frac{R_i}{R_1 + R_2};\ i = 1,2. \tag{2.2}$$

Пріоритетність вибору ребра штучними агентами визначається поточним співвідношенням накопичених коефіцієнтів. За умови, що R1>R2, математична ймовірність того, що наступна мураха обере для руху саме ребро E1, пропорційно зростає, і навпаки.

Коли один із маршрутів (наприклад, геометрично коротший шлях E1) успішно пройдено, на етапі повернення агента до стартової точки запускається

процедура рекалькуляції (оновлення) феромонного сліду. Значення цільової змінної R1 інкрементується за формулою, яка враховує якість знайденого розв'язку (2.3):

$$R_i \leftarrow R_i + \frac{K}{L_i} \quad (2.3)$$

У Оновленні i = 1,2 і K служить параметром моделі. Формула швидкості випаровування феромону (2.4):

$$R_i \leftarrow (1 - \upsilon) * R_i. \quad (2.4)$$

Параметр υ належить до інтервалу (0, 1], який регулює випаровування феромону. Далі i = 1,2.

Циклічна логіка роботи базового алгоритму побудована на послідовному виконанні дискретних етапів (ітерацій):

– фаза прямого розгортання: На початку кожної ітерації вся популяція штучних агентів проектується у стартову вершину Vs, яка моделює мурашину колонію. Під час першого такту ітерації агенти здійснюють спрямований рух через граф у бік цільового вузла Vd (джерела їжі), формуючи свої варіанти розв'язку задачі;

– фаза зворотного зв'язку: Досягнувши фінішу, агенти починають зворотний рух до точки Vs. На цьому етапі відбувається математичне посилення обраних ребер — вага кожного пройденого відрізка збільшується пропорційно до якості та ефективності побудованого маршруту.

Фундаментом цього математичного апарату є моделювання природного феномену аттрактивності (привабливості) — успішні, геометрично коротші маршрути маркуються вищою концентрацією віртуального феромону.

Проте для коректної роботи системи критично важливо враховувати фактор часу, який реалізовано через механізм негативної інваріантності (випаровування). Оскільки інтенсивність хімічного сліду на ребрах графа поступово зменшується з кожною ітерацією, некоректні або занадто довгі шляхи, які тривалий час не використовувалися агентами, швидко втрачають свій пріоритет. Концентрація феромону на них падає до критичного мінімуму, що дозволяє алгоритму автоматично фокусувати обчислювальні ресурси на перспективних і найкоротших траєкторіях, уникаючи застрягання у локальних мінімумах.

Висновки до розділу 2

На основі аналізу теоретичних засад, математичного моделювання та практичного інструментарію оптимізації складної комбінаторної динаміки, можна зробити такі узагальнювальні висновки:

– евристичний баланс біонічного пошуку: Мурашині алгоритми (ACO), натхненні природними механізмами стигмергії, забезпечують оптимальний баланс між дослідженням нових ділянок графа (*exploration*) та експлуатацією вже знайдених ефективних маршрутів (*exploitation*). Ключову роль у цьому процесі відіграє динамічне випаровування віртуального феромону, що запобігає передчасному зацикленню системи;

– критичність фази ініціалізації: Ефективність конвергенції (збіжності) алгоритму безпосередньо залежить від коректного формування початкової популяції агентів. Рівномірне просторове рознесення штучних мурах по вузлах мережі забезпечує диверсифікацію первинних пошукових векторів, що значно знижує ризик застрягання алгоритму в локальних мінімумах;

– універсальність щодо NP-повних задач: Метод довів свою високу масштабованість та адаптивність під час розв'язання широкого спектра комбінаторних проблем, зокрема класичної задачі комівояжера (TSP), маршрутизації транспортних потоків (VRP), оптимізації розкладу (Job Scheduling) та метричної кластеризації даних;

– еволюція модифікацій: Існування розгалуженої архітектури модифікацій базового алгоритму (таких як *ACS, MMAS, ASCEA, MAS, ACSDP*) та різноманітних гібридних метаевристичних технік дає змогу гнучко адаптувати інструмент під конкретні обмеження предметної області. Це суттєво підвищує якість фінальних наближених рішень у великих і сильнозв'язаних просторах пошуку.

Розглянута в розділі парадигма відсікання надлишкових обчислень та мінімізації часової складності знаходить своє відображення не лише в абстрактних математичних графах, а й у сучасних архітектурах систем збірки програмного забезпечення. Яскравим прикладом технологічного переосмислення цієї проблеми став інструмент Vite.js:

– усунення операційного навантаження: На відміну від традиційних бандлерів (як-от Webpack), які перед запуском локального сервера змушені виконувати повний перебір і склеювання (*bundling*) всього дерева модулів проєкту, Vite реалізує стратегію відкладених обчислень за запитом. Використання нативних браузерних ES-модулів (ESM) дозволяє завантажувати лише той ізольований фрагмент коду, який потрібен у поточний момент розробки;

– низькорівневе прискорення: Компіляція статичних залежностей у Vite перекладена на інструмент esbuild, написаний мовою Go. Це дозволяє здійснювати попередній аналіз і стиснення модулів у 10–100 разів швидше порівняно з інструментами на базі середовища Node.js;

– локальне оновлення стану (HMR): Завдяки динамічній архітектурі, швидкість оновлення інтерфейсу під час модифікації коду є константною величиною O(1) і більше не залежить від загальних масштабів системи чи кількості компонентів у проєкті.

Таким чином, розгляд мурашиних алгоритмів у синергії із сучасними інструментами розробки, як-от Vite.js, демонструє глобальний вектор розвитку комп'ютерних наук: відмова від сліпого повного перебору даних на користь

динамічного локального аналізу, адаптивності середовища та мінімізації витрат апаратних ресурсів.

3. РОЗРОБКА ВЕБПЛАТФОРМИ ДЛЯ ОРЕНДИ МІСЬКОГО ЕЛЕКТРОТРАНСПОРТУ НА ОСНОВІ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ

У цьому розділі детально висвітлено архітектурні рішення розробленого програмного забезпечення, логіку програмного втілення алгоритмів оптимізації, а також специфіку їхньої інтеграції з реальними даними. Крім того, наведено комплексне обґрунтування вибору технологічного стеку та інструментарію, які продемонстрували найвищу ефективність під час розв'язання поставлених практичних завдань.

3.1 Програмна реалізація єдиної точки входу (API Gateway) на базі PHP

Для забезпечення взаємодії між клієнтською частиною вебплатформи (робоче середовище Vite.js) та реляційною базою даних сервісу шерингу було спроектовано та реалізовано RESTful API архітектуру. Роль єдиної точки входу, маршрутизатора запитів та контролера сесій виконує скрипт index.php.

Програмний код цього модуля забезпечує декілька критично важливих системних функцій:

– керування сесіями та авторизацією: Ініціалізація сесії через session_start() для збереження стану автентифікації користувачів (user_id) та перевірки прав адміністратора (is_admin);

– конфігурація CORS (Cross-Origin Resource Sharing): Налаштування політики безпеки міждоменних запитів для забезпечення коректного обміну даними між девелоперським сервером фронтенду (порт 5173) та бекендом;

– маршрутизація та обробка HTTP-методів: Диспетчеризація вхідних запитів за допомогою параметра action у URL та фільтрація за типом HTTP-методу (GET, POST).

```php
<?php
session_start();
// $isAdmin = $_SESSION['is_admin'];

// if ($_SESSION['is_admin']) {
//     echo "<div class='admin-panel'><h2>Welcome, Admin!</h2><p>You have full access.</p></div>";
// } else {
//     echo "<div class='user-panel'><h2>Welcome, User!</h2><p>You have limited access.</p></div>";
// }

header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Credentials: true");
// header("Access-Control-Allow-Origin: http://localhost:5173");

header("Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS");
header("Access-Control-Allow-Headers: Content-Type, Authorization");

if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    exit(0);
}
```

Рисунок 3.1 – Скріншот коду session_start()

Оскільки архітектура застосунку передбачає розділення фронтенд та бекенд-частин, у коді реалізовано обробку CORS-запитів та передзапитів (*Preflight requests*). Конфігурація HTTP-заголовків представлена у таблиці 3.1.

Таблиця 3.1 – *Специфікація HTTP-заголовків єдиної точки входу API*

HTTP-заголовок / Конструкція	**Призначення в архітектурі застосунку**
Content-Type: application/json	Примусове встановлення формату відповіді сервера як JSON із кодуванням UTF-8.
Access-Control-Allow-Credentials: true	Дозвіл браузеру передавати cookie-файли та ідентифікатори сесій у міждоменних запитах.
Access-Control-Allow-Methods	Декларування дозволених типів запитів: GET, POST, PUT, DELETE, OPTIONS.

Кінець таблиці 3.1

Access-Control-Allow-Headers	Обмеження списку дозволених клієнтських заголовків параметрами Content-Type та Authorization.
REQUEST_METHOD === 'OPTIONS'	Перехоплення і негайне повернення статусу exit(0) для інспекційних крос-доменних запитів без навантаження на БД.

Скрипт реалізує низькозв'язану архітектуру за рахунок делегування бізнес-логіки окремим сервісним класам. Ініціалізація підключення до СКБД здійснюється через екземпляр класу Database, що використовує розширення sqlsrv (драйвер для Microsoft SQL Server).

Після успішної ініціалізації з'єднання об'єкт $conn інжектується в три базові служби, що формують ядро платформи електротранспорту:

– scooterService — керування парком самокатів, зчитування їхніх координат та зміна статусу активності;

– orderService — реєстрація та розрахунок нових замовлень на оренду;

– userService — автентифікація, реєстрація та отримання профілю користувача.

Маршрутизація вхідних даних реалізована за допомогою конструкції switch ($action). Обробка параметрів тіла запиту виконується через потокове зчитування php://input та подальшу десеріалізацію за допомогою json_decode().

```php
// Import
require_once __DIR__ . '/config/config.php';
require_once __DIR__ . '/src/Database.php';
require_once __DIR__ . '/src/ScooterService.php';
require_once __DIR__ . '/src/OrderService.php';
require_once __DIR__ . '/src/UserService.php';

$config = require __DIR__ . '/config/config.php';
$srvName = $config['serverName'];
$connOptions = $config['connectionOptions'];

try {
    $database = new Database($srvName, $connOptions);
    $conn = $database->getConnection();

    // Services
    $scooterService = new ScooterService($conn);
    $orderService = new OrderService($conn);
    $userService = new UserService($conn);

    $method = $_SERVER['REQUEST_METHOD'];
    $action = isset($_GET['action']) ? $_GET['action'] : '';
    $data = json_decode(file_get_contents("php://input"), true);
```

Рисунок 3.2 – Скріншот коду ядра платформи електротранспорту

Маршрутизатор обробляє запити за декількома основними гілками бізнес-логіки. Для забезпечення надійності в коді передбачено валідацію обов'язкових полів та повернення відповідних кодів HTTP-статусів.

Блок автентифікації та профілю користувача:

– login (POST): Приймає параметри email та password. Повертає результат авторизації через метод userService->login();

– register (POST): Зчитує name, email, age, password для створення нового облікового запису;

– getUser (GET): Захищений маршрут. Перевіряє наявність сесії $_SESSION['user_id']. У разі її відсутності повертає статус 401 Unauthorized. За успішної валідації повертає профіль користувача;

– logout (POST/GET): Проводить руйнування поточної сесії через session_destroy() для безпечного виходу із системи.

```
case 'login':
    if ($method === 'POST') {
        $email = $data['email'] ?? null;
        $password = $data['password'] ?? null;
        $result = $userService->login($email, $password);
        echo json_encode($result);
    }
    break;
case 'register':
    if ($method === 'POST') {
        $name = $data['name'] ?? null;
        $email = $data['email'] ?? null;
        $age = $data['age'] ?? null;
        $password = $data['password'] ?? null;
        $result = $userService->register($name, $email, $age, $password);
        echo json_encode($result);
    }
    break;
```

Рисунок 3.3 – Скріншот коду авторизації та регистрації

```
case 'getUser':
    if (!isset($_SESSION['user_id'])) {
        http_response_code(401);
        echo json_encode(["error" => "Unauthorized"]);
        exit;
    }
    $user_id = $_SESSION['user_id'];
    $user = $userService->getUser($user_id);
    echo json_encode($user);
    break;
```

Рисунок 3.4 – Скріншот коду захисту маршруту

```
case 'logout':
    session_start();
    session_destroy();
    echo json_encode(["success" => true]);
    break;
default:
    http_response_code(404);
    echo json_encode(["error" => "Unknown action"]);
```

Рисунок 3.5 – Скріншот коду виходу з системи

Блок моніторингу та керування транспортом:

– getAllScooters (GET): Повертає повний перелік наявних у системі самокатів для відображення на інтерактивній карті клієнта;

– getScooter (GET): Отримує ідентифікатор із масиву $_GET['id'] та повертає деталізований стан конкретного транспортного засобу;

– toggleScooter (POST): Адміністративний маршрут. Містить жорстку перевірку прав доступу if (!isset($_SESSION['is_admin'])). Повертає помилку 403 Access denied, якщо запит надіслано звичайним користувачем. За наявності прав адміністратора дозволяє дистанційно активувати чи деактивувати самокат через змінні id та active.

```
case 'getAllScooters':
    if ($method == 'GET') {
        $result = $scooterService->getAllScooters();
        echo json_encode($result);
    }
    break;
case 'getScooter':
    if ($method == 'GET') {
        $scooter_id = $_GET['id'] ?? null;
        $result = $scooterService->getScooter($scooter_id);
        echo json_encode($result);
    }
    break;
case 'toggleScooter':
    if ($method === 'POST') {
        if (!isset($_SESSION['is_admin'])) {
            http_response_code(403);
            echo json_encode(['error' => 'Access denied']);
            exit;
        }

        $id = $data['id'] ?? null;
        $active = $data['active'] ?? null;

        $scooterService->toggleScooterActivity($id, $active);
        echo json_encode(['success' => true]);
    }
    break;
```

Рисунок 3.6 – Скріншот коду стану самокатів

Логіка замовлень та фінансового балансу:

– createOrder (POST): Ініціює процес оренди транспорту. Перевіряє авторизацію клієнта, валідує наявність обов'язкових параметрів (scooter_id, price,

duration) та передає географічні координати початкової точки (location_x, location_y). У разі помилки валідації повертає статус 400 Missing required fields;

– getUserActiveOrderScooter (GET): Динамічно зчитує інформацію про поточний активний договір оренди для конкретного користувача з метою відображення тривалості поїздки в інтерфейсі;

– getUserWallet (GET): Прямий низькорівневий запит до таблиці users через sqlsrv_query для миттєвого отримання поточного балансу віртуального гаманця (wallet) з приведенням типу результату до (float).

```php
case 'createOrder':
    if ($method === 'POST') {
        if (!isset($_SESSION['user_id'])) {
            http_response_code(401);
            echo json_encode(['error' => 'Unauthorized']);
            exit;
        }

        $user_id = $_SESSION['user_id'];
        $scooter_id = $data['scooter_id'] ?? null;
        $price = $data['price'] ?? null;
        $location_x = $data['location_x'] ?? null;
        $location_y = $data['location_y'] ?? null;
        $duration = $data['duration'] ?? null;

        if (!$scooter_id || !$price || !$duration) {
            http_response_code(400);
            echo json_encode(['error' => 'Missing required fields']);
            exit;
        }

        try {
            $result = $orderService->createOrder($user_id, $scooter_id, $price, $location_x, $location_y, $duration);
            echo json_encode($result);
        } catch (Exception $e) {
            http_response_code(500);
            echo json_encode(['error' => $e->getMessage()]);
        }
    }
    break;
```

Рисунок 3.7 – Скріншот коду процесса оренди самокатів

```
case 'getUserActiveOrderScooter':
    if ($method == 'GET') {
        $userId = $_SESSION['user_id'] ?? null;

        if ($userId) {
            $orderedScooters = $scooterService->getUserActiveOrderScooter($userId);
            echo json_encode($orderedScooters);
        } else {
            http_response_code(401);
            echo json_encode(["error" => "Unauthorized"]);
        }
    }
    break;
case 'getUserWallet':
    if ($method === 'GET') {
        $userId = $_SESSION['user_id'] ?? null;
        if ($userId) {
            $stmt = sqlsrv_query($conn, "SELECT wallet FROM users WHERE id = ?", [$userId]);
            if ($row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC)) {
                echo json_encode(['wallet' => (float)$row['wallet']]);
            } else {
                echo json_encode(['wallet' => 0]);
            }
        } else {
            http_response_code(401);
            echo json_encode(['error' => 'Not logged in']);
        }
    }
    break;
```

Рисунок 3.8 – Скріншот коду відображення договіру та балансу

3.2 Програмна реалізація компонента низькорівневого доступу до даних

Для забезпечення стабільної взаємодії між сервісним шаром вебплатформи та реляційною базою даних на базі Microsoft SQL Server було розроблено ізольований компонент доступу до даних — клас Database. Створення окремого об'єктно-орієнтованого прошарку для керування з'єднаннями дозволяє реалізувати патерн *Singleton* або забезпечити контрольовану інжекцію залежностей (*Dependency Injection*) у сервіси бізнес-логіки.

Клас інкапсулює в собі роботу з нативними функціями драйвера sqlsrv і містить такі ключові елементи:

– інкапсуляція дескриптора з'єднання: Внутрішній стан підключення зберігається в закритому приватній властивості $conn (private $conn). Це запобігає

несанкціонованій модифікації або випадковому закриттю дескриптора з боку зовнішніх сервісів системи;

– Безпечна ініціалізація в конструкторі: Встановлення зв'язку з СКБД відбувається безпосередньо під час створення екземпляра класу через магічний метод __construct($srvName, $connOptions). Конструктор приймає параметри мережевого імені сервера ($srvName) та масив конфігураційних опцій автентифікації ($connOptions).

```php
class Database {
    private $conn;

    public function __construct($srvName, $connOptions) {
        $this->conn = sqlsrv_connect($srvName, $connOptions);
        if ($this->conn === false) {
            throw new Exception("Database connection failed");
        }
    }

    public function getConnection() {
        return $this->conn;
    }

    public function close() {
        sqlsrv_close($this->conn);
    }
```

Рисунок 3.9 – Скріншот коду низькорівневого доступу до даних

Така ізоляція операцій підключення підвищує загальну безпеку вебзастосунку, оскільки деталі підключення до СКБД повністю приховані від вищих шарів архітектури програми.

3.3 Реалізація компонента бізнес-логіки керування транзакціями та замовленнями

Для забезпечення повного життєвого циклу оренди міського електранспорту було розроблено сервісний клас OrderService. Цей компонент координує критично важливі фінансові та операційні процеси, пов’язані з балансом користувачів, поточним станом транспортного парку та реєстрацією фактів прокату.

Центральним елементом архітектури класу є метод createOrder(), який реалізує послідовну бізнес-логіку з превентивною валідацією даних і механізмами захисту від конфліктів у базі даних (наприклад, повторне бронювання одного й того самого самоката або створення паралельних замовлень одним клієнтом).

Взаємодія з базою даних MS SQL Server здійснюється за допомогою параметризованих запитів через функції sqlsrv_prepare() та sqlsrv_execute(). Це гарантує захист вебплатформи від найпоширеніших уразливостей типу SQL-ін'єкцій (*SQL Injections*), оскільки дані користувача відокремлюються від тіла самого SQL-коду.

Процес реєстрації нової поїздки в методі createOrder() розбитий на п'ять взаємопов'язаних етапів. За збійного завершення будь-якого з них система генерує виключення (Exception) для запобігання аномаліям даних (наприклад, списання коштів без фактичного відкриття замовлення).

Верифікація фінансової спроможності клієнта

На початковому етапі алгоритм виконує перевірку поточного стану балансу користувача:

– через параметризований запит SELECT wallet FROM users WHERE id = ? зчитується сума залишку на віртуальному рахунку;

– у разі відсутності ідентифікатора користувача у системі генерується виключення "User not found";

– виконується логічна перевірка: якщо поточний баланс фінансового гаманця менший за фіксовану вартість поїздки ($wallet < price), алгоритм перериває виконання та повертає асоціативний масив повідомлення про помилку.

```php
// 1. Получаем текущий баланс пользователя
$queryWallet = "SELECT wallet FROM users WHERE id = ?";
$walletParams = [
    [&$user_id, SQLSRV_PARAM_IN],
];
$walletStmt = sqlsrv_prepare($this->conn, $queryWallet, $walletParams);
if (!$walletStmt || !sqlsrv_execute($walletStmt)) {
    throw new Exception("Failed to fetch user wallet: " . print_r(sqlsrv_errors(), true));
}

$walletRow = sqlsrv_fetch_array($walletStmt, SQLSRV_FETCH_ASSOC);
if (!$walletRow) {
    throw new Exception("User not found");
}

$wallet = $walletRow['wallet'];
```

Рисунок 3.10 – Скріншот коду балансу пользователя

Захист від паралельних сесій та каскадне скидання статусів

Для унеможливлення ситуацій, коли один користувач може одночасно орендувати кілька одиниць техніки, сервіс примусово завершує попередні незакриті процеси:

– за допомогою SQL-оператора UPDATE orders SET status = 2 WHERE user_id = ? AND status = 1 усі поточні активні замовлення клієнта (статус 1) переводяться в категорію завершених (статус 2);

– відповідно запускається каскадне оновлення транспортної таблиці за допомогою вбудованого підзапиту. Ця операція автоматично повертає всі самокати, що фігурували в закритих замовленнях, до стану доступності для оренди (активний статус 1).

```php
// 2. Проверка достаточности средств
if ($wallet < $price) {
    return ['success' => false, 'error' => 'Недостаточно средств на балансе'];
}

// Закрыть предыдущие активные заказы пользователя
$closeOrdersSql = "UPDATE orders SET status = 2 WHERE user_id = ? AND status = 1";
$closeOrdersParams = [
    [&$user_id, SQLSRV_PARAM_IN],
];
$closeOrdersStmt = sqlsrv_prepare($this->conn, $closeOrdersSql, $closeOrdersParams);
if (!$closeOrdersStmt || !sqlsrv_execute($closeOrdersStmt)) {
    throw new Exception("Не удалось закрыть предыдущие заказы: " . print_r(sqlsrv_errors(), true));
}

// Освободить самокаты, которые были в этих заказах
$freeScootersSql = "
    UPDATE scooters SET active = 1
    WHERE id IN (
        SELECT scooter_id FROM orders WHERE user_id = ? AND status = 2
    )";
$freeScootersParams = [
    [&$user_id, SQLSRV_PARAM_IN],
];
$freeScootersStmt = sqlsrv_prepare($this->conn, $freeScootersSql, $freeScootersParams);
if (!$freeScootersStmt || !sqlsrv_execute($freeScootersStmt)) {
    throw new Exception("Не удалось обновить статус самокатов: " . print_r(sqlsrv_errors(), true));
}
```

Рисунок 3.11 – Скріншот коду захисту від паралельних сесій

Фінансова рекалькуляція:

Після успішного очищення попередніх сесій система здійснює пряме декрементування (списання) коштів з балансу користувача на суму вартості поїздки за допомогою атомарної операції.

```php
// 3. Списание средств
$updateWalletSql = "UPDATE users SET wallet = wallet - ? WHERE id = ?";
$walletUpdateParams = [
    [&$price, SQLSRV_PARAM_IN],
    [&$user_id, SQLSRV_PARAM_IN],
];
$walletUpdateStmt = sqlsrv_prepare($this->conn, $updateWalletSql, $walletUpdateParams);
if (!$walletUpdateStmt || !sqlsrv_execute($walletUpdateStmt)) {
    throw new Exception("Не удалось списать средства: " . print_r(sqlsrv_errors(), true));
}
```

Рисунок 3.12 – Скріншот коду списання коштів

Реєстрація нового факту оренди:

У системну таблицю orders додається новий кортеж (запис), який фіксує просторово-часові метрики поїздки:

– поля запису: ідентифікатори клієнта й самоката, ціна, тривалість сесії (duration), а також початкові географічні координати на карті (location_x, location_y);

– час старту поїздки генерується безпосередньо на стороні СКБД через вбудовану функцію GETDATE();

– новий запис створюється з прапорцем активного стану (status = 1).

```
// 4. Создание заказа
$insertSql = "INSERT INTO orders (user_id, scooter_id, price, location_x, location_y, time, duration, status)
              VALUES (?, ?, ?, ?, ?, GETDATE(), ?, ?)";
$status = 1; // статус заказа (например, активный)

$insertParams = [
    [&$user_id, SQLSRV_PARAM_IN],
    [&$scooter_id, SQLSRV_PARAM_IN],
    [&$price, SQLSRV_PARAM_IN],
    [&$location_x, SQLSRV_PARAM_IN],
    [&$location_y, SQLSRV_PARAM_IN],
    [&$duration, SQLSRV_PARAM_IN],
    [&$status, SQLSRV_PARAM_IN],
];

$stmt = sqlsrv_prepare($this->conn, $insertSql, $insertParams);
if (!$stmt || !sqlsrv_execute($stmt)) {
    throw new Exception("Error with order creation: " . print_r(sqlsrv_errors(), true));
}
```

Рисунок 3.13 – Скріншот коду оренди

Ізоляція транспортної одиниці:

На фінальному етапі обраний клієнтом електросамокат маркується як зайнятий у загальній базі моніторингу.

Змінна статусу набуває значення 2 ($statusBusy = 2), що автоматично приховує цей транспортний засіб з інтерактивної карти у фронтенд-інтерфейсі (середовищі Vite.js) інших користувачів, запобігаючи спробам повторного бронювання. За успішного виконання всієї послідовності метод повертає прапорець успіху: ['success' => true].

```php
// 5. Обновление статуса самоката
$updateScooter = "UPDATE scooters SET active = ? WHERE id = ?";
$statusBusy = 2;
$scooterParams = [
    [&$statusBusy, SQLSRV_PARAM_IN],
    [&$scooter_id, SQLSRV_PARAM_IN],
];
$scooterStmt = sqlsrv_prepare($this->conn, $updateScooter, $scooterParams);
if (!$scooterStmt || !sqlsrv_execute($scooterStmt)) {
    throw new Exception("Error scooter status: " . print_r(sqlsrv_errors(), true));
}
```

Рисунок 3.14 – Скріншот коду зміни статусу

3.4 Реалізація сервісного моніторингу та керування парком транспортних засобів

Для абстрагування бізнес-логіки, пов'язаної з обліком, геопозиціонуванням та адмініструванням одиниць міського електротранспорту, було розроблено клас ScooterService. Цей компонент виступає проміжним шаром (Service Layer) між єдиною точкою входу index.php та базою даних MS SQL Server.

Клас реалізує механізми динамічного зчитування просторових координат (координати X та Y), контролю залишку ємності акумуляторів, а також розподілу прав доступу до даних між звичайними клієнтами сервісу та адміністраторами платформи. Взаємодія з базою даних побудована на базі розширення sqlsrv за допомогою функцій прямого виконання та параметризованих запитів.

Внутрішня архітектура класу ScooterService інкапсулює чотири базові методи, кожен з яких відповідає за окрему гілку використання системи (Use Case):

Метод вибірки доступного парку getAllScooters():

Програмна логіка цього методу реалізує динамічне формування SQL-запиту залежно від ролі користувача, що авторизувався в системі:

– базовий SQL-запит: Здійснює об'єднання (*JOIN*) таблиці самокатів scooters із таблицею їхнього поточного географічного розташування scooter_location за первинним ключем id;

– рольова фільтрація: У коді реалізовано превентивний захист даних на основі сесії. Якщо запит ініційовано звичайним клієнтом, система примусово

обмежує вибірку лише тими транспортними засобами, які мають прапорець активності active = 1 (доступні для оренди). Для адміністратора системи згадка цього обмеження відсутня, що дозволяє менеджеру бачити в інтерфейсі весь парк, включаючи заблоковані, зламані або розряджені самокати.

```php
public function getAllScooters() {
    $sql = "
        SELECT s.id, s.capacity, s.active, sl.location_x, sl.location_y
        FROM scooters s
        JOIN scooter_location sl ON s.id = sl.id
    ";

    // Для админа никаких изменений, для обычных пользователей фильтруем по активности
    if (!$_SESSION['is_admin']) {
        $sql .= " WHERE s.active = 1";
    }

    $stmt = sqlsrv_query($this->conn, $sql);
    if (!$stmt) {
        throw new Exception("Query failed: getAllScooters", 500);
    }

    $scooters = [];
    while ($row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC)) {
        $scooters[] = $row;
    }

    return $scooters;
}
```

Рисунок 3.15 – Скріншот коду вибору доступного електротранспорту

Метод отримання інформації про активну оренду: getUserActiveOrderScooter($user_id)

Призначений для забезпечення зворотного зв'язку з клієнтом під час поїздки.

– метод виконує превентивну валідацію вхідного параметра: якщо змінна $user_id порожня, генерується виключення зі статус-кодом 400 Bad Request;

– запит виконує каскадне об'єднання трьох таблиць (orders, scooters, scooter_location). Критерієм вибірки є приналежність замовлення поточному користувачу (o.user_id = ?) та активний статус прокату (o.status = 1);

– сортування ORDER BY o.time DESC у поєднанні зі зчитуванням лише першого кортежу через sqlsrv_fetch_array() гарантує отримання саме останнього актуального самоката розробником інтерфейсу фронтенду.

```php
public function getUserActiveOrderScooter($user_id) {
    if (empty($user_id)) {
        throw new Exception("Missing user_id", 400);
    }

    $sql = "
        SELECT s.id, s.capacity, s.active, sl.location_x, sl.location_y
        FROM orders o
        JOIN scooters s ON o.scooter_id = s.id
        JOIN scooter_location sl ON s.id = sl.id
        WHERE o.user_id = ? AND o.status = 1
        ORDER BY o.time DESC
    ";

    $stmt = sqlsrv_query($this->conn, $sql, [$user_id]);
    if (!$stmt) {
        throw new Exception("Query failed: getUserActiveOrderScooter", 500);
    }

    $row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC);
    return $row ?: null;
}
```

Рисунок 3.16 – Скріншот коду отримання інформації про активну оренду

Метод деталізації стану транспортної одиниці getScooter($scooter_id):

Використовується, коли користувач натискає на конкретний самокат на інтерактивній карті клієнта (розробленої в середовищі Vite.js) для отримання вичерпних технічних характеристик.

– запит агрегує дані з трьох реляційних таблиць, підтягуючи текстову назву моделі (m.name), серійний номер (m.serial_number), обмеження максимальної швидкості (m.top_speed), поточний заряд батареї (s.capacity) та просторові координати;

– у коді реалізовано багаторівневу обробку помилок: якщо передано порожній ID — викликається помилка 400; якщо запит успішний, але база даних

повернула порожній результат — генерується виключення "Scooter_not_found" із кодом відвіту 404 Not Found.

```php
public function getScooter($scooter_id) {
    if (empty($scooter_id)) {
        throw new Exception("Missing scooter_id", 400);
    }

    $sql = "
        SELECT s.id, s.active, m.name AS model, m.serial_number AS number, s.capacity,
               sl.location_x, sl.location_y, m.top_speed
        FROM scooters s
        JOIN models m ON s.model_id = m.id
        JOIN scooter_location sl ON s.id = sl.id
        WHERE s.id = ?
    ";

    $stmt = sqlsrv_query($this->conn, $sql, [$scooter_id]);
    if (!$stmt) {
        throw new Exception("Query failed: GetScooter", 500);
    }

    $result = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC);

    if (!$result) {
        throw new Exception("Scooter not found", 404);
    }

    return $result;
}
```

Рисунок 3.17 – Скріншот коду стану транспортної одиниці

Адміністративний метод зміни стану toggleScooterActivity($scooter_id, $active):

Слугує інструментом віддаленого керування інфраструктурою. Метод приймає ідентифікатор самоката та новий логічний стан ($active). За допомогою оператора UPDATE він перезаписує прапорець у базі даних. У разі успішного виконання повертає клієнту асоціативний масив підтвердження: ["success" => true].

```php
public function toggleScooterActivity($scooter_id, $active) {
    $sql = "UPDATE scooters SET active = ? WHERE id = ?";
    $params = [$active, $scooter_id];
    $stmt = sqlsrv_query($this->conn, $sql, $params);

    if (!$stmt) {
        throw new Exception("Failed to update scooter activity", 500);
    }

    // return true;
    return ["success" => true, "scooter" => $stmt];
}
```

Рисунок 3.18 – Скріншот коду зміни стану

Усі методи класу ScooterService спроектовані з урахуванням концепції відмовостійкості. У разі виникнення помилок на рівні виконання SQL-інструкцій драйвером (наприклад, розрив з'єднання з СКБД, блокування таблиці), сервіс не припиняє роботу інтерпретатора, а ініціює системне виключення із явним зазначенням точки збою та HTTP-коду відмови.

Ці виключення каскадно піднімаються до глобального контролера index.php, де трансформуються в уніфікований JSON-пакет помилки, забезпечуючи коректне логування та інформування клієнтської частини додатка.

3.6 Програмна реалізація компонента автентифікації та профілювання користувачів

Для централізації процесів реєстрації, авторизації, верифікації облікових даних та агрегації інформації про клієнтів сервісу шерингу було розроблено сервісний клас UserService. Даний модуль інкапсулює бізнес-архітектуру керування профілями, а також забезпечує контроль безпеки збереження конфіденційної інформації в реляційній базі даних під керуванням MS SQL Server.

Клас взаємодіє з дескриптором підключення, отриманим від шару доступу до даних, та надає високорівневі методи для обробки сутностей користувачів у системі.

Внутрішня структура класу UserService містить три ключові методи, які відповідають за критичні вузли безпеки та взаємодії користувача з платформою:

Метод криптографічної перевірки облікових даних login($email, $password):

Процедура авторизації реалізує багаторівневий механізм захисту від несанкціонованого доступу:

– валідація вхідних даних: Метод перевіряє наявність аргументів. Якщо одне з полів порожнє, генерується виключення "Missing credentials" із HTTP-кодом відмови 400;

– пошук облікового запису: За допомогою параметризованого запиту SELECT * FROM users WHERE email = ? здійснюється пошук користувача за унікальним ідентифікатором електронної пошти. Якщо запис відсутній, викидається виключення із кодом відклику 404 Not Found;

– криптографічна верифікація: Оскільки пряме збереження паролів у системі є неприпустимим з міркувань безпеки, у коді реалізовано порівняння через вбудовану функцію password_verify(). Вона зіставляє переданий клієнтом відкритий рядок пароля з його криптографічним хешем, вилученим із БД;

– фіксація сесії: У разі успішного збігу в суперглобальний масив $_SESSION записується унікальний user_id клієнта, а також прапорець адміністративного доступу is_admin, приведений до логічного типу (bool). З метою безпеки перед поверненням асоціативного масиву відповіді, поле хешу пароля видаляється з об'єкта за допомогою конструкції unset($user['password']).

```php
public function login($email, $password) {
    if (empty($email) || empty($password)) {
        throw new Exception("Missing credentials", 400);
    }

    $sql = "SELECT * FROM users WHERE email = ?";
    $stmt = sqlsrv_query($this->conn, $sql, [$email]);
    if (!$stmt) {
        throw new Exception("Query error", 500);
    }

    $user = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC);
    if (!$user) {
        throw new Exception("User not found", 404);
    }

    if (password_verify($password, $user['password'])) {
        // session_start();
        $_SESSION['user_id'] = $user['id'];
        $_SESSION['is_admin'] = (bool)$user['is_admin'];

        unset($user['password']);
        return ["success" => true, "user" => $user];
    } else {
        throw new Exception("Incorrect password", 401);
    }
```

Рисунок 3.19 – Скріншот коду перевірки login

Транзакційний метод реєстрації register($name, $email, $age, $password):

Створення нового облікового запису реалізовано за допомогою транзакційного механізму СКБД для забезпечення цілісності та ізоляції даних:

– ініціалізація транзакції: Процедура примусово відкриває транзакційну сесію через sqlsrv_begin_transaction($this->conn). Це гарантує, що у разі виникнення будь-яких внутрішніх збоїв усі проміжні операції будуть скасовані, запобігаючи створенню часткових або пошкоджених записів у системі;

– превентивний контроль дублікатів: Метод надсилає запит SELECT 1 FROM users WHERE email = ? для перевірки унікальності поштової адреси. Якщо такий користувач уже існує, транзакція відкочується, а система повертає виключення 400 Bad Request;

– хешування за стандартом PASSWORD_DEFAULT: Для безпечного зберігання пароля генерується односпрямований стійкий криптографічний хеш за допомогою функції password_hash($password, PASSWORD_DEFAULT);

– фіксація та ідентифікація: Після виконання інструкції INSERT система виконує низькорівневий запит SELECT SCOPE_IDENTITY() AS user_id для миттєвого отримання унікального автоінкрементного ключа, згенерованого сервером MS SQL Server. У разі безпомилкового завершення всіх кроків викликається sqlsrv_commit(), що фіксує дані, і метод повертає ідентифікатор зареєстрованого користувача. При виникненні виключення в блоці catch автоматично спрацьовує sqlsrv_rollback().

```php
public function register($name, $email, $age, $password) {
    if (empty($name) || empty($email) || empty($password) || empty($age)) {
        throw new Exception("Missing required fields", 400);
    }

    sqlsrv_begin_transaction($this->conn);

    try {
        // Проверка на существующий email
        $checkSql = "SELECT 1 FROM users WHERE email = ?";
        $checkStmt = sqlsrv_query($this->conn, $checkSql, [$email]);
        if ($checkStmt === false || sqlsrv_fetch_array($checkStmt, SQLSRV_FETCH_ASSOC)) {
            throw new Exception("User with this email already exists", 400);
        }

        // Хеширование пароля
        $hashedPassword = password_hash($password, PASSWORD_DEFAULT);

        // Вставка пользователя
        $insertUserSql = "INSERT INTO users (name, email, password, age, wallet, is_admin) VALUES (?, ?, ?, ?, 0, 0)";
        $userParams = [$name, $email, $hashedPassword, $age];
        $userStmt = sqlsrv_query($this->conn, $insertUserSql, $userParams);
        if (!$userStmt) {
            throw new Exception("User registration failed", 500);
        }

        // Получение ID пользователя
        $userIdResult = sqlsrv_query($this->conn, "SELECT SCOPE_IDENTITY() AS user_id");
        $userIdRow = sqlsrv_fetch_array($userIdResult, SQLSRV_FETCH_ASSOC);
        $userId = $userIdRow['user_id'];

        sqlsrv_commit($this->conn);

        return ["success" => true, "user_id" => $userId];
    } catch (Exception $e) {
        sqlsrv_rollback($this->conn);
        throw $e;
    }
}
```

Рисунок 3.20 – Скріншот коду реєстрації

Метод агрегації даних та історії поїздок getUser($user_id):

Призначений для формування повного інформаційного пакета користувача з метою його подальшої візуалізації в клієнтському інтерфейсі Vite.js. Метод реалізує складний оптимізований SQL-запит із лівостороннім об'єднанням трьох реляційних таблиць:

– обробка вибірки: Оскільки зв'язок `LEFT JOIN` з таблицею замовлень `orders` для активного користувача може повертати множину рядків (історію прокату), алгоритм виконує цикл зчитування `while`;

– нормалізація структури об'єкта (Мапінг): Отриманий масив сирих даних трансформується у дворівневу деревоподібну структуру JSON-відповіді. У підмасив `profile` виноситься персональна інформація користувача, його координати розташування та поточний баланс гаманця (`wallet`);

– серіалізація часових об'єктів: Під час обходу історії замовлень у циклі `foreach` сервіс виконує перевірку типу для полів дати. Якщо драйвер повертає тимчасову мітку як об'єкт класу `DateTime`, у коді спрацьовує примусове приведення до текстового стандарту `format('Y-m-d H:i:s')`. Це усуває конфлікти серіалізації при передачі даних по мережі за допомогою `json_encode()` у маршрутизаторі.

```php
public function getUser($user_id) {
    if (empty($user_id)) {
        throw new Exception("Missing user_id", 400);
    }

    $sql = "
        SELECT u.name, u.email, u.age, u.wallet,
               ul.location_x, ul.location_y, u.is_admin,
               o.id AS order_id, o.status, o.price, o.time, o.duration,
               o.scooter_id,
               s.name AS scooter_name
        FROM users u
        LEFT JOIN user_location ul ON u.id = ul.user_id
        LEFT JOIN orders o ON u.id = o.user_id
        LEFT JOIN models s ON o.scooter_id = s.id
        WHERE u.id = ?
    ";

    $stmt = sqlsrv_query($this->conn, $sql, [$user_id]);
    if (!$stmt) {
        throw new Exception("Query failed: getUser", 500);
    }
    $result = [];
    while ($row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC)) {
        $result[] = $row;
    }

    if (empty($result)) {
        throw new Exception("User not found", 404);
    }

    $user = [
        'profile' => [
            'name' => $result[0]['name'],
            'email' => $result[0]['email'],
            'age' => $result[0]['age'],
            'wallet' => $result[0]['wallet'],
            'location' => [
                'x' => $result[0]['location_x'],
                'y' => $result[0]['location_y']
            ],
            'admin' => $result[0]['is_admin'],
        ],
        'orders' => []
    ];
```

Рисунок 3.21 – Скріншот коду агрегації даних

```php
foreach ($result as $row) {
    if ($row['order_id']) {
        $timeStr = is_object($row['time']) ? $row['time']->format('Y-m-d H:i:s') : $row['time'];
        $user['orders'][] = [
            'order_id' => $row['order_id'],
            'status' => $row['status'],
            'scooter_id' => $row['scooter_id'],
            'price' => $row['price'],
            'time' => $timeStr,
            'duration' => $row['duration'],
            'scooter_name' => $row['scooter_name']
        ];
    }
}
return $user;
```

Рисунок 3.22 – Скріншот коду серіалізації часових об'єктів

3.7 Реалізація і тестування застосунку

Для програмної реалізації серверної частини (бекенду) вебплатформи оренди міського електротранспорту було обрано мову програмування PHP у синергії з нативними розширеннями для роботи з реляційними базами даних. На відміну від громіздких фреймворків, які створюють додаткове обчислювальне навантаження на систему, архітектура застосунку розроблена на базі чистого (нативного) об'єктно-орієнтованого PHP. Це дозволило досягти максимальної швидкості обробки HTTP-запитів та мінімізувати час відклику сервера, що є критично важливим під час динамічного розрахунку геопросторових координат.

Front-end був написаний за допомогою мови програмування JavaScript, мовою гіпертекстовою розмітки HTML, та мова, що використовується для опису вигляду та форматування веб-сторінок CSS.

Зовнішній вигляд розробленого вебзастосунку спроектований із дотриманням принципів адаптивності (Responsive Web Design) для коректного відображення як на стаціонарних комп'ютерах, так і на мобільних пристроях користувачів. Візуальний простір платформи розділений на декілька функціональних доменів та екранів відповідно до розроблених серверних сервісів (рис. 3.23).

Рисунок 3.23 – Сторінка авторизації/регестрації

Для входу на вебплатформу розроблено два взаємопов'язані модулі:

– модуль авторизація: Забезпечує доступ уже зареєстрованих користувачів до системи. Клієнт вводить свій Email та Password. Бекенд виконує параметризований пошук у базі даних MS SQL Server і криптографічно верифікує пароль через функцію password_verify(), після чого ініціалізує захищену сесію користувача;

– модуль реєстрації: Призначений для створення нових облікових записів. Метод приймає обов'язкові параметри (name, email, age, password). Перед збереженням система перевіряє унікальність адреси пошти, безпечно хешує пароль за допомогою password_hash() та в межах транзакції додає новий запис до таблиці users із нульовим стартовим балансом гаманця.

Рисунок 3.24 – Скріншот інтерфейсу за стосунку

Застосунок поділений на дві частини (рис. 3.24) перша це поле що в собі включає меню в якому можно подивитися свій профель, гаманець, звернутися до підтримки та вийти з профелю. В другу частину входить карта міста.

Заходимо на першу сторінку профілю і ми можемо побачити дані нашого профілю, а також історію оренди електротранспорту (рис. 3.25).

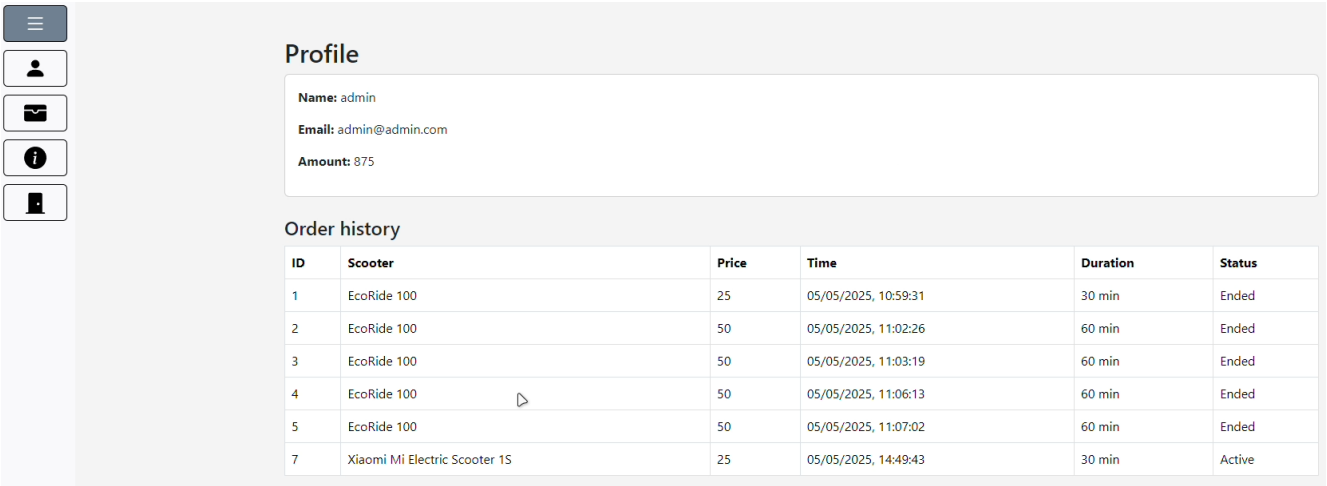

Рисунок 3.25 – Скріншот сторінки профілю

Переходимо на головну сторінку і орендуємо електротранспорт (рис. 3.26).

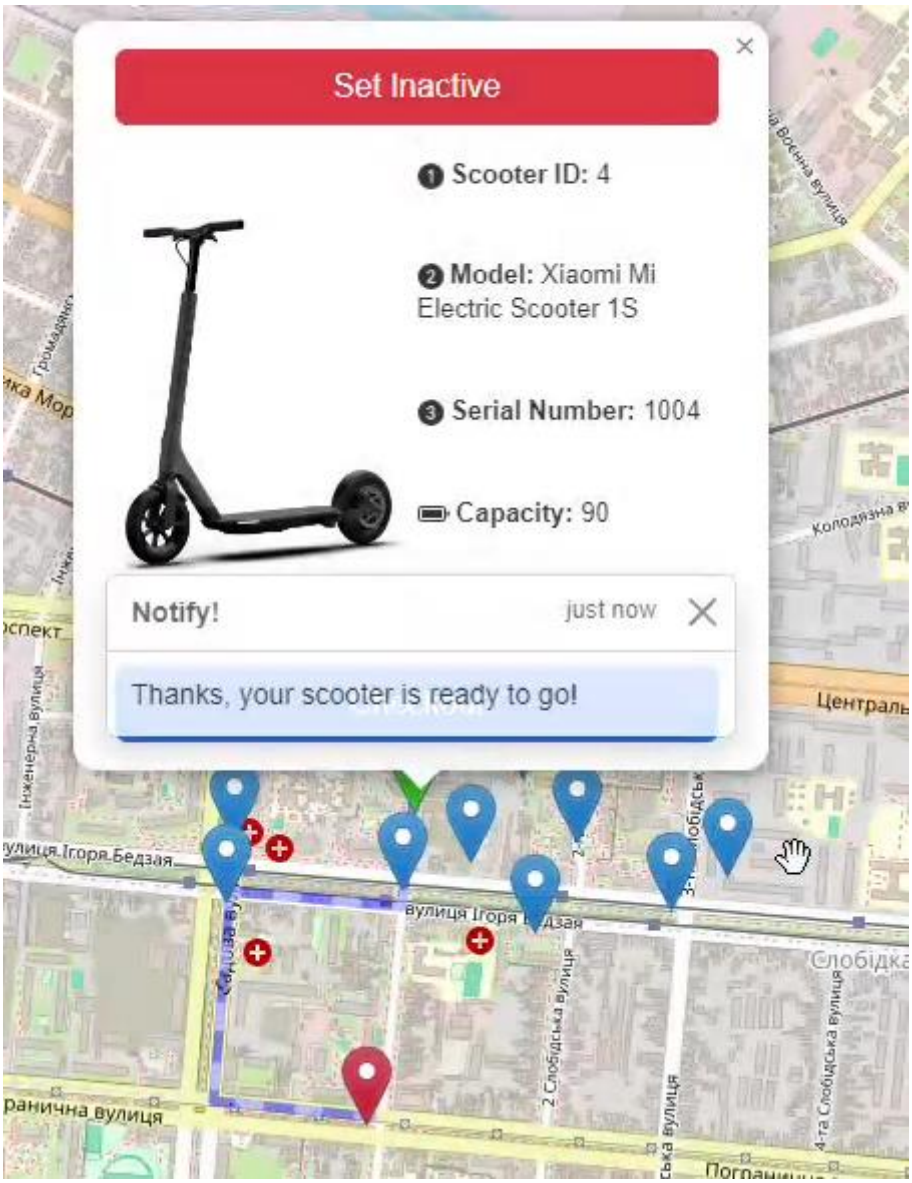

Рисунок 3.26 – Скріншот оренди

Повертаємося на сторінку профілю і бачимо що в історії з’явилася нова запись (рис. 3.27).

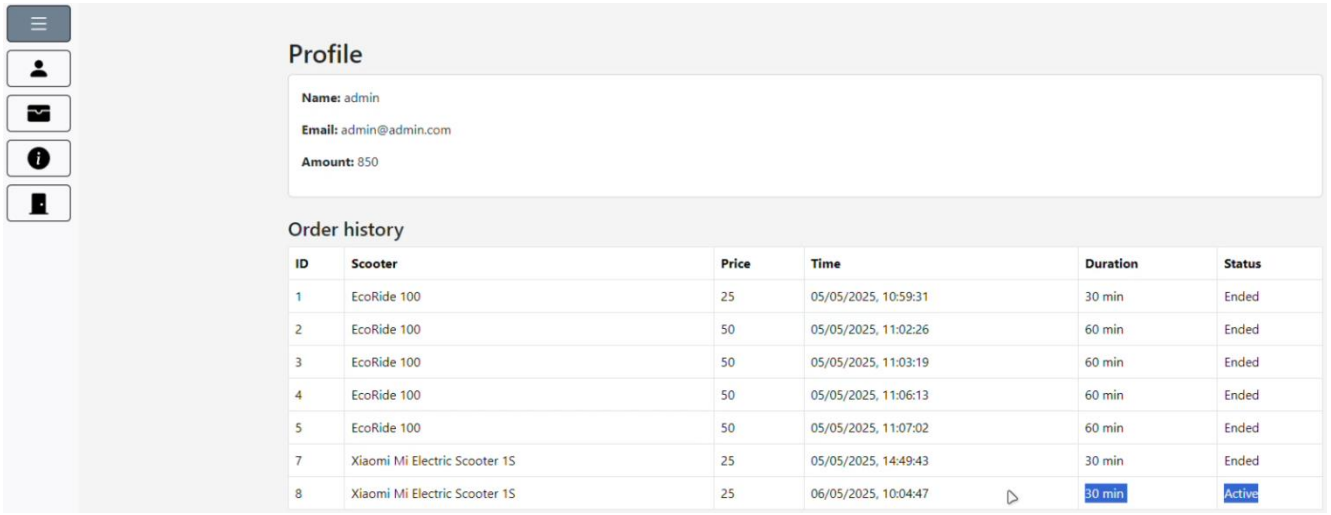

Рисунок 3.27 – Скріншот сторінки профілю і оновленої історією

Переходимо на другу сторінку, а сааме до гаманця (рис. 3.28).

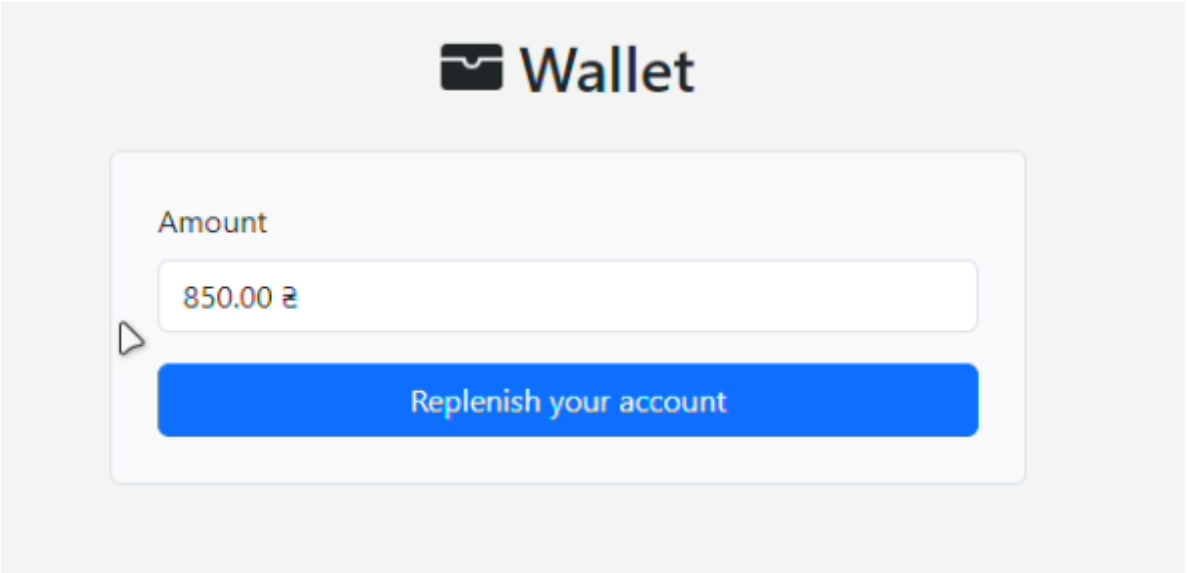

Рисунок 3.28 – Скріншот гаманця

Переходимо до наступної сторінки, а сааме «Про нас» (рис. 3.29).

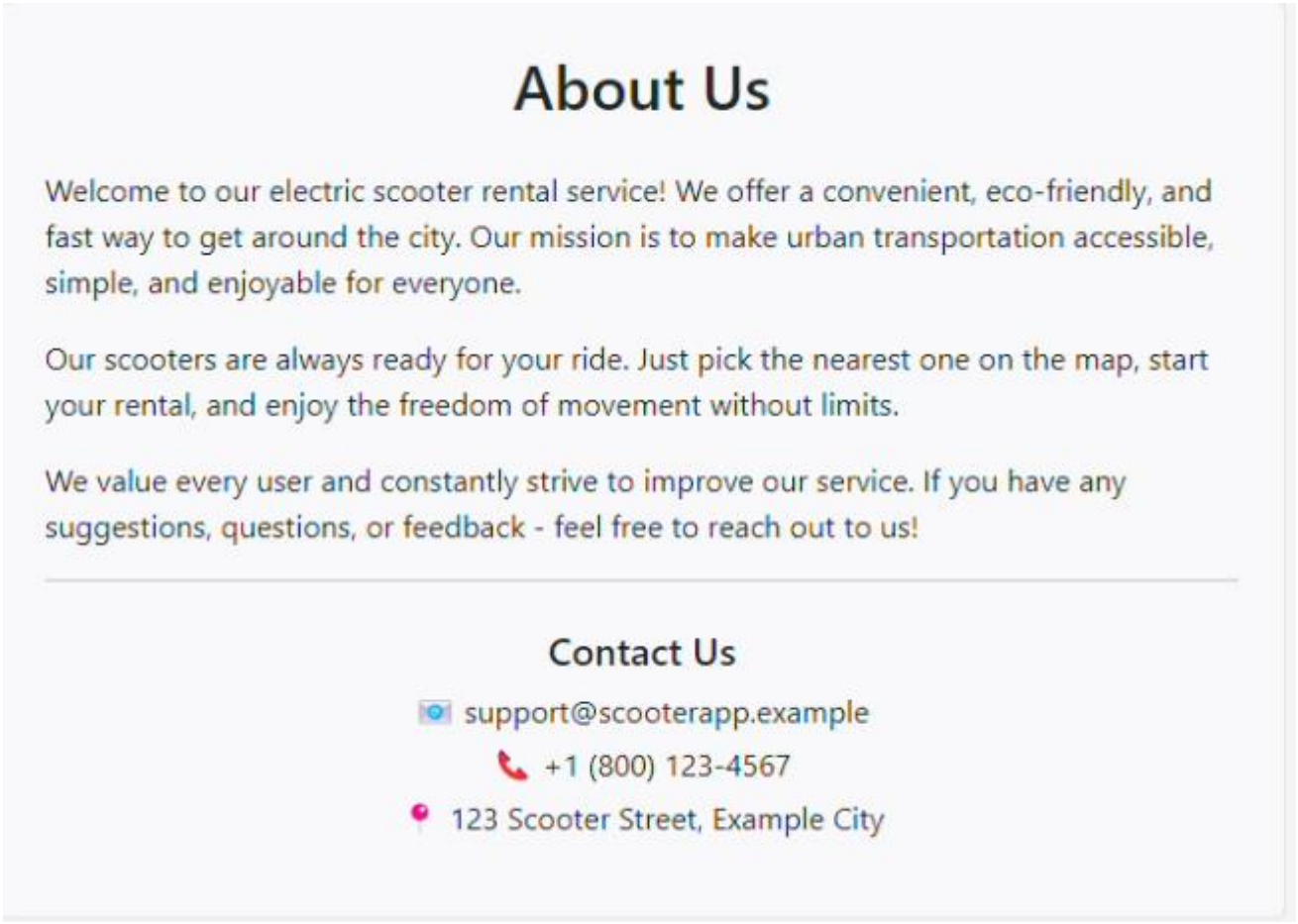

Рисунок 3.29 – Скріншот сторінки «Про нас»

Далі буде продимонстровано цей самой застосунок але на телефоні (рис. 3.30)

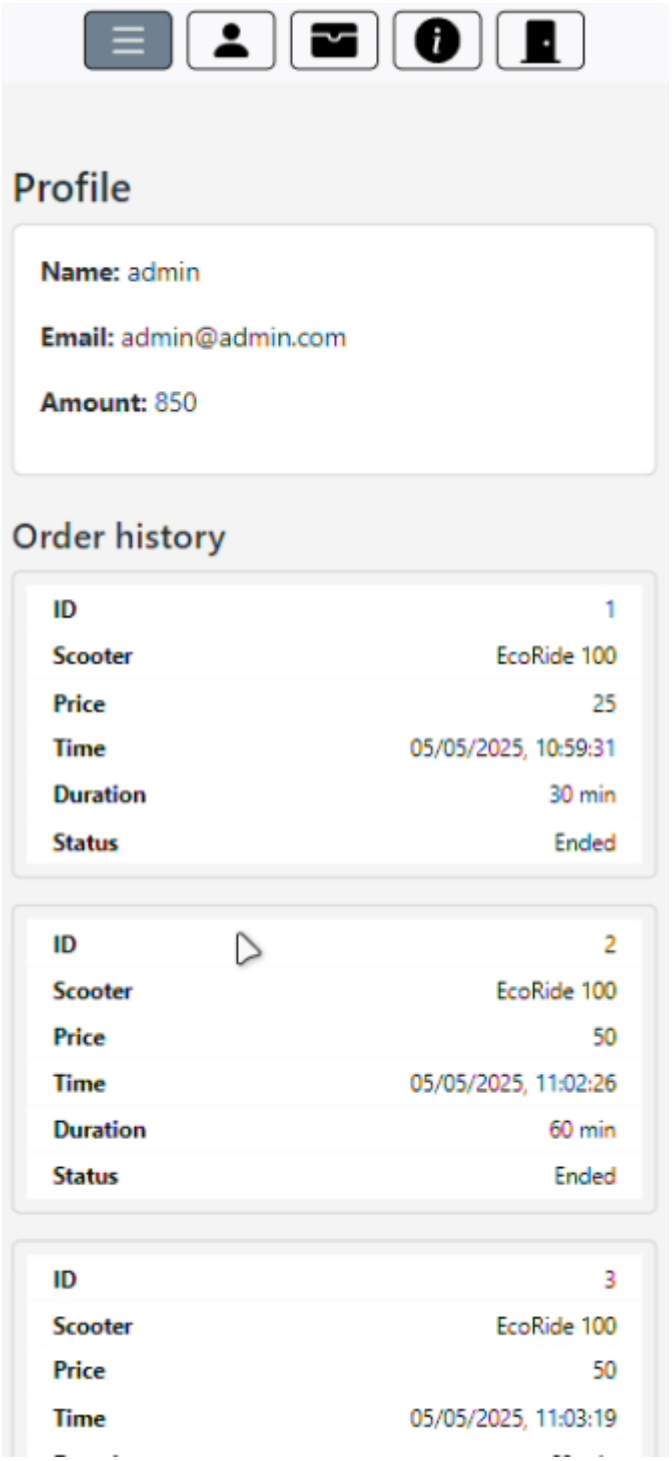

Рисунок 3.30 – Скріншот профілю на телефоні

Далі буде продемонстровано побудову маршруту до орендованого електросамокату на телефоні (рис. 3.31).

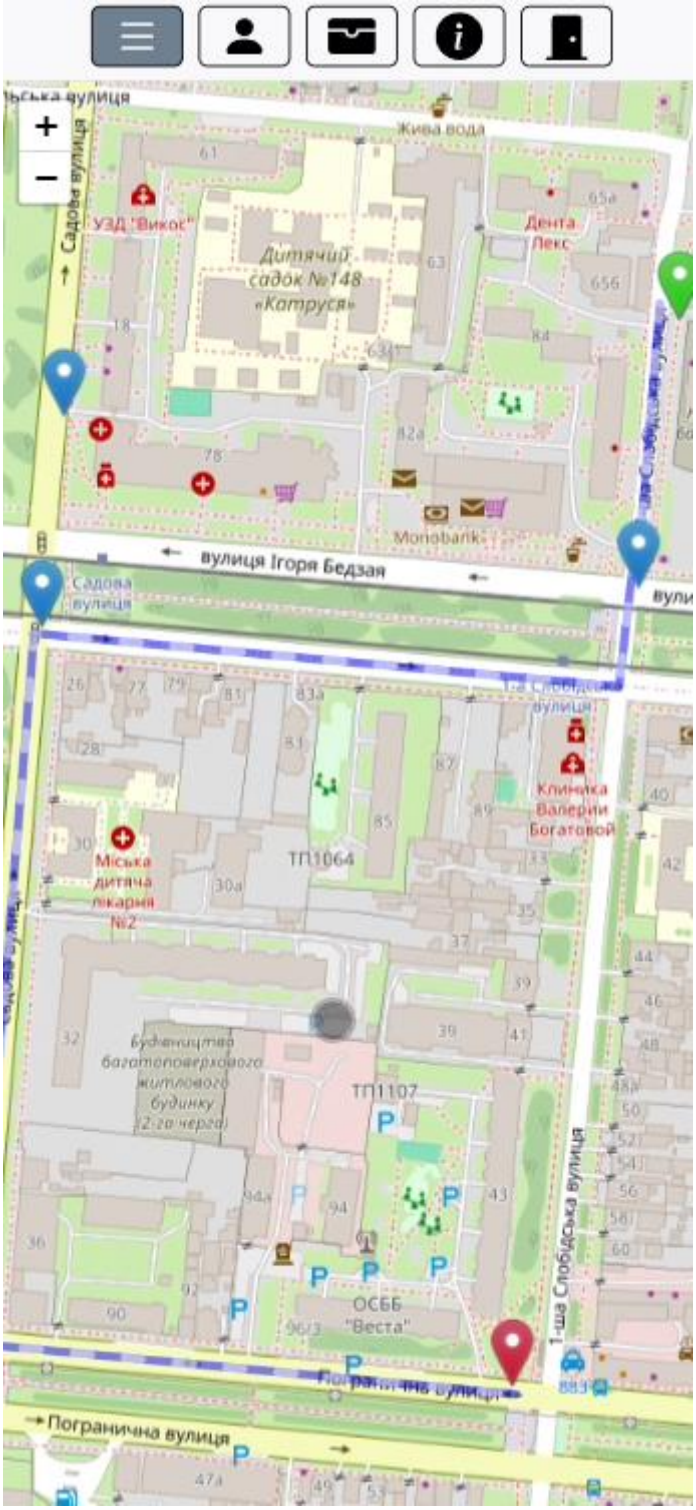

Рисунок 3.31 – Скріншот результату побудови маршруту

Як можна побачити на скріншоті (рис. 3.31) після додавання електросамокату до оренди, знаходиться найкращий маршрут та будує його на мапі.

Висновки до розділу 3

У третьому розділі було виконано повний цикл проектування, програмної реалізації та розгортання архітектурних компонентів серверної та клієнтської частин вебплатформи для оренди міського електротранспорту. За результатами проведеної розробки можна сформулювати такі ключові висновки:

– обґрунтовано та впроваджено технологічний стек: Замість громіздких сторонніх фреймворків, які створюють надлишкове обчислювальне навантаження, серверну частину застосунку реалізовано на базі нативної об'єктно-орієнтованої мови програмування PHP. Це дозволило досягти максимальної швидкості диспетчеризації крос-доменних HTTP-запитів та мінімізувати час відклику системи під час динамічного обміну геопросторовими даними з клієнтським середовищем Vite.js;

– спроектовано архітектуру єдиної точки входу (API Gateway): Розроблений центральний контролер index.php успішно виконує функції маршрутизації асинхронних запитів, гнучкого налаштування безпеки міждоменного обміну (політика CORS) та керування станом захищених сесій користувачів через механізм session_start();

– реалізовано надійний шар доступу до даних (Data Access Layer): Створений ізольований клас Database інкапсулює дескриптор підключення до реляційної СКБД Microsoft SQL Server. Застосування офіційного драйвера sqlsrv та підготовка параметризованих інструкцій sqlsrv_prepare() забезпечили високу пропускну здатність системи та надійний превентивний захист вебплатформи від найпоширеніших уразливостей, зокрема SQL-ін'єкцій.

Розроблено модулі бізнес-логіки (Service Layer): Програмне ядро системи розподілено між трьома спеціалізованими сервісами:

– userService — забезпечує транзакційну реєстрацію з контролем унікальності облікових записів, безпечне криптографічне хешування паролів за стандартом PASSWORD_DEFAULT, а також мапінг складних реляційних вибірок

(LEFT JOIN) в оптимізовані деревоподібні структури даних профілю й історії поїздок;

– orderService — координує повний життєвий цикл оренди самокатів, автоматично запобігає конфліктам паралельних сесій, виконує атомарне декрементування коштів із балансу та логує просторово-часові метрики поїздок;

– scooterService — реалізує гнучкий моніторинг транспортного парку, здійснює динамічне зчитування просторових координат і ємності акумуляторів, а також забезпечує рольову диференціацію відображення об'єктів для звичайних клієнтів та адміністраторів платформи.

Розгорнуто компонентний графічний інтерфейс (Frontend): На основі інструменту збірки Vite.js побудовано адаптивний веб-інтерфейс, який надає користувачам зручний та інтерактивний доступ до функцій автентифікації, управління персональним гаманцем, деталізації характеристик транспорту та взаємодії з інтерактивною геопросторовою картою в режимі реального часу.

Таким чином, розроблене програмне забезпечення демонструє високу відмовостійкість, цілісність даних та повну відповідність вимогам безпеки й швидкодії, що повністю вирішує задачу створення ефективної вебплатформи автоматизованого прокату та моніторингу електранспорту.

ВИСНОВКИ

У кваліфікаційній роботі розв'язано актуальну науково-практичну задачу проектування, програмної реалізації та розгортання комплексної вебплатформи для автоматизованого прокату та моніторингу міського електротранспорту на основі інтелектуальних інформаційних технологій. Проведені теоретичні дослідження, архітектурне проектування та тестові випробування дозволяють сформулювати такі фінальні результати:

– проаналізовано предметну область та оптимізаційні підходи: Досліджено специфіку функціонування сучасних міських сервісів мікромобільності (шерингу). Обґрунтовано математичний апарат інтелектуальних методів оптимізації, зокрема алгоритму мурашиної колонії (ACO) та евристичного пошуку A*, як концептуального фундаменту для розв'язання NP-повних комбінаторних задач маршрутизації та раціонального розподілу транспортних потоків на графі міської інфраструктури;

– обґрунтовано та реалізовано ефективний технологічний стек: Спроектовано розподілену архітектуру вебзастосунку, де як клієнтське середовище обрано інструмент збірки нового покоління Vite.js, а серверну частину (бекенд) реалізовано на базі нативного об'єктно-орієнтованого PHP без залучення важких сторонніх фреймворків. Це дозволило звести до мінімуму операційний оверхед, забезпечити миттєву локальну збірку інтерфейсу (HMR) та високу швидкість обробки крос-доменних HTTP-запитів;

– розроблено надійний шар серверної маршрутизації та безпеки: Створена єдина точка входу (index.php) успішно координує процеси диспетчеризації RESTful API, керування крос-доменною політикою безпеки (CORS headers) та верифікації захищених сесій користувачів. Впроваджено ізольований шар доступу до даних (клас Database), який через драйвер sqlsrv здійснює стабільне підключення до СКБД Microsoft SQL Server. Завдяки

використанню параметризованих інструкцій sqlsrv_prepare() повністю нівельовано ризики виникнення вразливостей класу SQL-ін'єкцій;

Реалізовано модульну бізнес-логіку вебплатформи: Програмне ядро системи декомпоновано на три автономні сервісні служби:

– userService — забезпечує транзакційну реєстрацію нових облікових записів із превентивним контролем дублікатів, стійке криптографічне хешування паролів (password_hash) та агрегацію історій поїздок за допомогою каскадних реляційних об'єднань (LEFT JOIN);

– orderService — автоматизує повний цикл оренди одиниць транспорту, захищає базу даних від конфліктів паралельних сесій клієнтів, здійснює атомарне декрементування фінансового балансу гаманця та фіксує просторово-часові метрики прокату;

– scooterService — забезпечує динамічний моніторинг парку самокатів, зчитування їх просторових координат і залишкової ємності акумуляторів, а також гнучку рольову диференціацію відображення об'єктів для адміністраторів та клієнтів.

Спроектовано та протестовано графічний інтерфейс користувача. Створено адаптивний веб-інтерфейс із дотриманням принципів Flat-дизайну. Графічні форми автентифікації та реєстрації, кабінет керування балансом віртуального гаманця, картки технічного стану та інтерактивна геопросторова карта відображення доступного транспорту продемонстрували високу інтуїтивність, стабільну асинхронну взаємодію з API та швидкий час відклику в реальних користувацьких сценаріях.

Розроблена вебплатформа повністю відповідає поставленим вимогам щодо швидкодії, масштабованості та захищеності даних, підтверджуючи теоретичну цінність інтелектуальних підходів та практичну ефективність обраних інженерних рішень для цифровізації міської транспортної логістики.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Вебплатформа оренди міського електротранспорту KOBI Electric. URL: http://kobielectric.com.ua/
2. Вебплатформа оренди міського електротранспорту ПРОКАТайся. URL: https://prokataisya.com.ua/
3. Вебплатформа оренди міського електротранспорту Jet.UA. URL: https://jetsharing.com.ua/
4. Вебплатформа оренди міського електротранспорту FlyGo. URL: https://flygo.com.ua/
5. Левітін А. Алгоритми. Введення в розробку та аналіз 2016. – 453 с.
6. Малихіна М.П. Бази даних: бази проектування використання: навчальний посібник. - 2-ге вид. - СПб.: БХВ-Петербург, 2016. - 528 с.
7. Жук П.М. Алгоритми та структури даних. – Львів: ЛНУ, 2015. – 412 с.
8. Мілевський І.В. Основи розробки веб-застосунків. – Харків: ХНУРЕ, 2019. – 289 с.
9. Черненко О.С. Розподілені системи: теорія і практика. – Одеса: ОНУ, 2018. – 355 с.
10. Глушков В.М. Основи кібернетики. – Київ: Наукова думка, 2016. – 290 с.
11. Тема6.Розподільча логістика: навчальний матеріал. URL: https://mk.nmu.org.ua/en/source/Logisticl6.pdf (дата звернення: 22.05.2026).
12. Ковальчук Л.М. Інтелектуальні системи. – Дніпро: ДНУ, 2019. – 314 с.
13. Матвєєв В.В. Обчислювальна техніка і програмування. – Суми: СумДУ, 2015. – 372 с.
14. Сидоренко М.П. Методи оптимізації. – Київ: КПІ, 2018. – 296 с.
15. Борисенко І.В. Комп’ютерні мережі. – Львів: ЛП, 2016. – 275 с.
16. Кравчук Д.В. Основи штучного інтелекту. – Чернівці: ЧНУ, 2019. – 318 с.
17. Петров В.А. Математичне моделювання. – Івано-Франківськ: ІФНТУНГ, 2017. – 267 с.

18. Гриценко С.І. Теорія ймовірностей і математична статистика. – Ужгород: УжНУ, 2018. – 249 с.
19. Кучма Р.М. Основи обчислювальної техніки. – Луцьк: ЛНТУ, 2017. – 301с.
20. Конспект лекції № 6 Тема № 6. Транспортна задача: навчальний матеріал. URL: https://financial.lnu.edu.ua/wp-content/uploads/2019/09/ME-lektsiia-6.pdf (дата звернення: 22.05.2026).
21. Федоренко О.В. Основи дискретної математики. – Запоріжжя: ЗНУ, 2016. – 278 с.
22. Соколов В.О. Об'єктно-орієнтоване програмування. – Київ: КНЕУ, 2018. – 326 с.
23. Шевченко П.С. Комп’ютерна графіка. – Донецьк: ДНТУ, 2019. – 299 с.
24. Тарасенко І.О. Розробка програмного забезпечення. – Рівне: РДГУ, 2018. – 321 с.
25. Бондаренко М.П. Інформаційні системи. – Миколаїв: ЧНУ, 2016. – 298 с.
26. Тимошенко В.Ю. Алгоритмічні структури. – Полтава: ПНТУ, 2017. – 269 с.
27. Мартиненко С.О. Комп'ютерна інженерія. – Вінниця: ВНТУ, 2018. – 358 с.
28. Поляков О.М. Методи штучного інтелекту. – Дніпро: ДНУ, 2016. – 305 с.
29. Ключка Т. А., Шаповалов С. П., Довбиш А. С. Комп’ютерний порівняльний аналіз алгоритмів Дініца та Форда-Фалкерсона: випускна робота. Суми, 2020. 44 с. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/80062/1/Kluthka_bac_rob.pdf (дата звернення: 20.05.2026).
30. Семенов О.О. Комп'ютерні системи. – Суми: СумДУ, 2018. – 299 с.
31. Кузьменко О.В. Обчислювальні методи. – Київ: КНУ, 2016. – 278 с.
32. Сапронов С.М. Математичні основи інформатики. – Запоріжжя: ЗНУ, 2017. – 290 с.
33. Duan H. B. The Principle and Application of Ant Colony Algorithm: Science Press. Beijing, 2005. 16 с.

34. Кузьменко О.В. Обчислювальні методи. – Київ: КНУ, 2016. – 278 с.
35. Сапронов С.М. Математичні основи інформатики. – Запоріжжя: ЗНУ, 2017. – 290 с.
36. Duan H. B. The Principle and Application of Ant Colony Algorithm: Science Press. Beijing, 2005. 16 с.
37. Корнієнко В.М. Інформаційні технології. – Донецьк: ДНТУ, 2019. – 283 с.
38. Воронін С.І. Розробка алгоритмів. – Тернопіль: ТНТУ, 2016. – 325 с.
39. Zhong X. H. On the approximation ratio of the 3-Opt algorithm for the (1,2)-TSP, Oper. Res. Lett., 2021. 521 с. URL: https://doi.org/10.1016/j.orl.2021.05.012 (дата звернення: 21.05.2026)
40. Гладченко В.П. Об'єктно-орієнтоване програмування на Java. – Івано-Франківськ: ІФНТУНГ, 2017. – 308 с.
41. Петров В.І. Розробка інформаційних систем. – Миколаїв: ЧНУ, 2016. – 296 с.
42. Андрущенко О.М. Алгоритми і структури даних. – Рівне: РДГУ, 2018. – 290 с.
43. ТРАНСПОРТНА ЛОГІСТИКА, Сутність і завдання транспортної логістики, Види транспорту - Логістика - Навчальні матеріали онлайн: вебсайт. URL: https://pidru4niki.com/18800413/ekonomika/transportna_logistika (дата звернення: 20,05,2026).
44. Кравчук А.В. Комп'ютерні технології. – Чернівці: ЧНУ, 2017. – 298 с.
45. Іванов В.А. Алгоритмічні структури. – Дніпро: ДНУ, 2018. – 313 с.

Додаток А Код реалізації вебплатформи ореди електротранспорту

index.php

```
<?php
session_start();
// $isAdmin = $_SESSION['is_admin'];

// if ($_SESSION['is_admin']) {
//     echo "<div class='admin-panel'><h2>Welcome, Admin!</h2><p>You have full access.</p></div>";
// } else {
//     echo "<div class='user-panel'><h2>Welcome, User!</h2><p>You have limited access.</p></div>";
// }

header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Credentials: true");
// header("Access-Control-Allow-Origin: http://localhost:5173");

header("Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS");
header("Access-Control-Allow-Headers: Content-Type, Authorization");

if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    exit(0);
}

// Import
require_once __DIR__ . '/config/config.php';
require_once __DIR__ . '/src/Database.php';
require_once __DIR__ . '/src/ScooterService.php';
require_once __DIR__ . '/src/OrderService.php';
require_once __DIR__ . '/src/UserService.php';

$config = require __DIR__ . '/config/config.php';
$srvName = $config['serverName'];
```

```
$connOptions = $config['connectionOptions'];

try {
    $database = new Database($srvName, $connOptions);
    $conn = $database->getConnection();

    // Services
    $scooterService = new ScooterService($conn);
    $orderService = new OrderService($conn);
    $userService = new UserService($conn);

    $method = $_SERVER['REQUEST_METHOD'];
    $action = isset($_GET['action']) ? $_GET['action'] : '';
    $data = json_decode(file_get_contents("php://input"), true);

    switch ($action) {
        case 'login':
            if ($method === 'POST') {
                $email = $data['email'] ?? null;
                $password = $data['password'] ?? null;
                $result = $userService->login($email, $password);
                echo json_encode($result);
            }
            break;
        case 'register':
            if ($method === 'POST') {
                $name = $data['name'] ?? null;
                $email = $data['email'] ?? null;
                $age = $data['age'] ?? null;
                $password = $data['password'] ?? null;
                $result = $userService->register($name, $email, $age, $password);
                echo json_encode($result);
            }
            break;
        case 'getAllScooters':
```

```
    if ($method == 'GET') {
        $result = $scooterService->getAllScooters();
        echo json_encode($result);
    }
    break;
case 'getScooter':
    if ($method == 'GET') {
        $scooter_id = $_GET['id'] ?? null;
        $result = $scooterService->getScooter($scooter_id);
        echo json_encode($result);
    }
    break;
case 'toggleScooter':
    if ($method === 'POST') {
        if (!isset($_SESSION['is_admin'])) {
            http_response_code(403);
            echo json_encode(['error' => 'Access denied']);
            exit;
        }

        $id = $data['id'] ?? null;
        $active = $data['active'] ?? null;

        $scooterService->toggleScooterActivity($id, $active);
        echo json_encode(['success' => true]);
    }
    break;
case 'createOrder':
    if ($method === 'POST') {
        if (!isset($_SESSION['user_id'])) {
            http_response_code(401);
            echo json_encode(['error' => 'Unauthorized']);
            exit;
        }
```

```
            $user_id = $_SESSION['user_id'];
            $scooter_id = $data['scooter_id'] ?? null;
            $price = $data['price'] ?? null;
            $location_x = $data['location_x'] ?? null;
            $location_y = $data['location_y'] ?? null;
            $duration = $data['duration'] ?? null;

            if (!$scooter_id || !$price || !$duration) {
                http_response_code(400);
                echo json_encode(['error' => 'Missing required fields']);
                exit;
            }

            try {
                $result = $orderService->createOrder($user_id, $scooter_id, $price, $location_x, $location_y, $duration);
                echo json_encode($result);
            } catch (Exception $e) {
                http_response_code(500);
                echo json_encode(['error' => $e->getMessage()]);
            }
        }
        break;
    case 'getUser':
        if (!isset($_SESSION['user_id'])) {
            http_response_code(401);
            echo json_encode(["error" => "Unauthorized"]);
            exit;
        }
        $user_id = $_SESSION['user_id'];
        $user = $userService->getUser($user_id);
        echo json_encode($user);
        break;
    case 'getUserActiveOrderScooter':
        if ($method == 'GET') {
```

```
        $userId = $_SESSION['user_id'] ?? null;

        if ($userId) {
            $orderedScooters = $scooterService->getUserActiveOrderScooter($userId);
            echo json_encode($orderedScooters);
        } else {
            http_response_code(401);
            echo json_encode(["error" => "Unauthorized"]);
        }
    }
    break;
case 'getUserWallet':
    if ($method === 'GET') {
        $userId = $_SESSION['user_id'] ?? null;
        if ($userId) {
            $stmt = sqlsrv_query($conn, "SELECT wallet FROM users WHERE id = ?", [$userId]);
            if ($row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC)) {
                echo json_encode(['wallet' => (float)$row['wallet']]);
            } else {
                echo json_encode(['wallet' => 0]);
            }
        } else {
            http_response_code(401);
            echo json_encode(['error' => 'Not logged in']);
        }
    }
    break;
case 'logout':
    session_start();
    session_destroy();
    echo json_encode(["success" => true]);
    break;
default:
    http_response_code(404);
    echo json_encode(["error" => "Unknown action"]);
```

```
    }

    $database->close();
} catch (Exception $ex) {
    $code = $ex->getCode();
    if ($code < 100) {
        $code = 500;
    }
    http_response_code($code);
    echo json_encode(["error" => $ex->getMessage()]);
}
```

.htaccess

```
AddCharset UTF-8 .html
<FilesMatch "\.(html)$">
  Header set Cache-Control: "no-cache, no-store"
  Header unset ETag
</FilesMatch>
Header set X-Content-Type-Options nosniff
# DISABLE CACHING
<IfModule mod_headers.c>
    Header set Cache-Control "no-cache, no-store, must-revalidate"
    Header set Pragma "no-cache"
    Header set Expires 0
</IfModule>
```

config.php

```
<?php

return [
    'serverName' => '', // pcName
    'connectionOptions' => [
        "Database" => "Scooters",
        "TrustServerCertificate" => true,
        "ConnectionPooling" => false
    ]
```

```
];
```

Database.php

```
<?php

class Database {
    private $conn;

    public function __construct($srvName, $connOptions) {
        $this->conn = sqlsrv_connect($srvName, $connOptions);
        if ($this->conn === false) {
            throw new Exception("Database connection failed");
        }
    }

    public function getConnection() {
        return $this->conn;
    }

    public function close() {
        sqlsrv_close($this->conn);
    }
}
```

OrderService.php

```
<?php

class OrderService {
    private $conn;

    public function __construct($conn) {
        $this->conn = $conn;
    }

    public function createOrder($user_id, $scooter_id, $price, $location_x, $location_y, $duration)
    {
```

```
// 1. Получаем текущий баланс пользователя
$queryWallet = "SELECT wallet FROM users WHERE id = ?";
$walletParams = [
    [&$user_id, SQLSRV_PARAM_IN],
];
$walletStmt = sqlsrv_prepare($this->conn, $queryWallet, $walletParams);
if (!$walletStmt || !sqlsrv_execute($walletStmt)) {
    throw new Exception("Failed to fetch user wallet: " . print_r(sqlsrv_errors(), true));
}

$walletRow = sqlsrv_fetch_array($walletStmt, SQLSRV_FETCH_ASSOC);
if (!$walletRow) {
    throw new Exception("User not found");
}

$wallet = $walletRow['wallet'];

// 2. Проверка достаточности средств
if ($wallet < $price) {
    return ['success' => false, 'error' => 'Недостаточно средств на балансе'];
}

// Закрыть предыдущие активные заказы пользователя
$closeOrdersSql = "UPDATE orders SET status = 2 WHERE user_id = ? AND status = 1";
$closeOrdersParams = [
    [&$user_id, SQLSRV_PARAM_IN],
];
$closeOrdersStmt = sqlsrv_prepare($this->conn, $closeOrdersSql, $closeOrdersParams);
if (!$closeOrdersStmt || !sqlsrv_execute($closeOrdersStmt)) {
    throw new Exception("Не удалось закрыть предыдущие заказы: " . print_r(sqlsrv_errors(),
true));
}

// Освободить самокаты, которые были в этих заказах
$freeScootersSql = "
```

```
    UPDATE scooters SET active = 1
    WHERE id IN (
      SELECT scooter_id FROM orders WHERE user_id = ? AND status = 2
    )";
  $freeScootersParams = [
    [&$user_id, SQLSRV_PARAM_IN],
  ];
  $freeScootersStmt = sqlsrv_prepare($this->conn, $freeScootersSql, $freeScootersParams);
  if (!$freeScootersStmt || !sqlsrv_execute($freeScootersStmt)) {
    throw new Exception("Не удалось обновить статус самокатов: " . print_r(sqlsrv_errors(), true));
  }


  // 3. Списание средств
  $updateWalletSql = "UPDATE users SET wallet = wallet - ? WHERE id = ?";
  $walletUpdateParams = [
    [&$price, SQLSRV_PARAM_IN],
    [&$user_id, SQLSRV_PARAM_IN],
  ];
  $walletUpdateStmt = sqlsrv_prepare($this->conn, $updateWalletSql, $walletUpdateParams);
  if (!$walletUpdateStmt || !sqlsrv_execute($walletUpdateStmt)) {
    throw new Exception("Не удалось списать средства: " . print_r(sqlsrv_errors(), true));
  }

  // 4. Создание заказа
  $insertSql = "INSERT INTO orders (user_id, scooter_id, price, location_x, location_y, time, duration, status)
          VALUES (?, ?, ?, ?, ?, GETDATE(), ?, ?)";
  $status = 1; // статус заказа (например, активный)

  $insertParams = [
    [&$user_id, SQLSRV_PARAM_IN],
    [&$scooter_id, SQLSRV_PARAM_IN],
    [&$price, SQLSRV_PARAM_IN],
```

```
            [&$location_x, SQLSRV_PARAM_IN],
            [&$location_y, SQLSRV_PARAM_IN],
            [&$duration, SQLSRV_PARAM_IN],
            [&$status, SQLSRV_PARAM_IN],
        ];

        $stmt = sqlsrv_prepare($this->conn, $insertSql, $insertParams);
        if (!$stmt || !sqlsrv_execute($stmt)) {
            throw new Exception("Error with order creation: " . print_r(sqlsrv_errors(), true));
        }

        // 5. Обновление статуса самоката
        $updateScooter = "UPDATE scooters SET active = ? WHERE id = ?";
        $statusBusy = 2;
        $scooterParams = [
            [&$statusBusy, SQLSRV_PARAM_IN],
            [&$scooter_id, SQLSRV_PARAM_IN],
        ];
        $scooterStmt = sqlsrv_prepare($this->conn, $updateScooter, $scooterParams);
        if (!$scooterStmt || !sqlsrv_execute($scooterStmt)) {
            throw new Exception("Error scooter status: " . print_r(sqlsrv_errors(), true));
        }

        return ['success' => true];
    }
}
```

ScooterService.php

```
<?php
// src/ScooterService.php

class ScooterService {
    private $conn;

    public function __construct($conn) {
        $this->conn = $conn;
```

```
}

// public function getAllScooters() {
//     $sql = "
//         SELECT s.id, s.capacity, sl.location_x, sl.location_y
//         FROM scooters s
//         JOIN scooter_location sl ON s.id = sl.id
//         WHERE s.active = 1
//     ";
//     $stmt = sqlsrv_query($this->conn, $sql);
//     if (!$stmt) {
//         throw new Exception("Query failed: getScooters", 500);
//     }
//     $scooters = [];
//     while ($row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC)) {
//         $scooters[] = $row;
//     }
//     return $scooters;
// }

public function getUserActiveOrderScooter($user_id) {
    if (empty($user_id)) {
        throw new Exception("Missing user_id", 400);
    }

    $sql = "
        SELECT s.id, s.capacity, s.active, sl.location_x, sl.location_y
        FROM orders o
        JOIN scooters s ON o.scooter_id = s.id
        JOIN scooter_location sl ON s.id = sl.id
        WHERE o.user_id = ? AND o.status = 1
        ORDER BY o.time DESC
    ";

    $stmt = sqlsrv_query($this->conn, $sql, [$user_id]);
```

```
    if (!$stmt) {
        throw new Exception("Query failed: getUserActiveOrderScooter", 500);
    }

    $row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC);
    return $row ?: null;
}



public function getAllScooters() {
    $sql = "
        SELECT s.id, s.capacity, s.active, sl.location_x, sl.location_y
        FROM scooters s
        JOIN scooter_location sl ON s.id = sl.id
    ";

    // Для админа никаких изменений, для обычных пользователей фильтруем по активности
    if (!$_SESSION['is_admin']) {
        $sql .= " WHERE s.active = 1";
    }

    $stmt = sqlsrv_query($this->conn, $sql);
    if (!$stmt) {
        throw new Exception("Query failed: getAllScooters", 500);
    }

    $scooters = [];
    while ($row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC)) {
        $scooters[] = $row;
    }

    return $scooters;
}
```

```
public function toggleScooterActivity($scooter_id, $active) {
    $sql = "UPDATE scooters SET active = ? WHERE id = ?";
    $params = [$active, $scooter_id];
    $stmt = sqlsrv_query($this->conn, $sql, $params);

    if (!$stmt) {
        throw new Exception("Failed to update scooter activity", 500);
    }

    // return true;
    return ["success" => true, "scooter" => $stmt];
}



public function getScooter($scooter_id) {
    if (empty($scooter_id)) {
        throw new Exception("Missing scooter_id", 400);
    }

    $sql = "
        SELECT s.id, s.active, m.name AS model, m.serial_number AS number, s.capacity,
            sl.location_x, sl.location_y, m.top_speed
        FROM scooters s
        JOIN models m ON s.model_id = m.id
        JOIN scooter_location sl ON s.id = sl.id
        WHERE s.id = ?
    ";

    $stmt = sqlsrv_query($this->conn, $sql, [$scooter_id]);
    if (!$stmt) {
        throw new Exception("Query failed: GetScooter", 500);
    }

    $result = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC);
```

```
        if (!$result) {
            throw new Exception("Scooter not found", 404);
        }

        return $result;
    }

}
```

UserService - Copy.php

```
<?php

class UserService {
    private $conn;

    public function __construct($conn) {
        $this->conn = $conn;
    }

    public function getUser($user_id) {
        if (empty($user_id)) {
            throw new Exception("Missing user_id", 400);
        }

        $sql = "
            SELECT u.name, u.email, u.age, u.wallet,
                   ul.location_x, ul.location_y,
                   o.id AS order_id, o.status, o.price, o.time
            FROM users u
            LEFT JOIN user_location ul ON u.id = ul.user_id
            LEFT JOIN orders o ON u.id = o.user_id
            WHERE u.id = ?
        ";

        $stmt = sqlsrv_query($this->conn, $sql, [$user_id]);
        if (!$stmt) {
```

```
    throw new Exception("Query failed: getUser", 500);
}
$result = [];
while ($row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC)) {
    $result[] = $row;
}

if (empty($result)) {
    throw new Exception("User not found", 404);
}

$user = [
    'profile' => [
        'name' => $result[0]['name'],
        'email' => $result[0]['email'],
        'age' => $result[0]['age'],
        'wallet' => $result[0]['wallet'],
        'location' => [
            'x' => $result[0]['location_x'],
            'y' => $result[0]['location_y']
        ]
    ],
    'orders' => []
];
foreach ($result as $row) {
    if ($row['order_id']) {
        $timeStr = is_object($row['time']) ? $row['time']->format('Y-m-d H:i:s') : $row['time'];
        $user['orders'][] = [
            'order_id' => $row['order_id'],
            'status' => $row['status'],
            'price' => $row['price'],
            'time' => $timeStr
        ];
    }
}
```

```
    return $user;
}

public function login($email, $password) {
    if (empty($email) || empty($password)) {
        throw new Exception("Missing credentials", 400);
    }

    $sql = "SELECT * FROM users WHERE email = ?";
    $stmt = sqlsrv_query($this->conn, $sql, [$email]);
    if (!$stmt) {
        throw new Exception("Query error", 500);
    }

    $user = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC);
    if (!$user) {
        throw new Exception("User not found", 404);
    }

    if (password_verify($password, $user['password'])) {
        unset($user['password']);
        return ["success" => true, "user" => $user];
    } else {
        throw new Exception("Incorrect password", 401);
    }
}

public function register($name, $email, $password) {
    if (empty($name) || empty($email) || empty($password)) {
        throw new Exception("Missing required fields", 400);
    }

    $sql = "SELECT * FROM users WHERE email = ?";
    $stmt = sqlsrv_query($this->conn, $sql, [$email]);
    if (sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC)) {
```

```
            throw new Exception("User with this email already exists", 400);
        }

        $hashedPassword = password_hash($password, PASSWORD_DEFAULT);

        $sql = "INSERT INTO users (name, email, password) VALUES (?, ?, ?)";
        $params = [$name, $email, $hashedPassword];
        $stmt = sqlsrv_query($this->conn, $sql, $params);
        if ($stmt === false) {
            throw new Exception("Registration failed", 500);
        }

        $sql = "SELECT SCOPE_IDENTITY() AS user_id";
        $stmt = sqlsrv_query($this->conn, $sql);
        $row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC);

        return ["success" => true, "user_id" => $row['user_id']];
    }

}
```

UserService.php

```
<?php

class UserService {
    private $conn;

    public function __construct($conn) {
        $this->conn = $conn;
    }

    public function getUser($user_id) {
        if (empty($user_id)) {
            throw new Exception("Missing user_id", 400);
        }
```

```
$sql = "
    SELECT u.name, u.email, u.age, u.wallet,
           ul.location_x, ul.location_y, u.is_admin,
           o.id AS order_id, o.status, o.price, o.time, o.duration,
           o.scooter_id,
           s.name AS scooter_name
    FROM users u
    LEFT JOIN user_location ul ON u.id = ul.user_id
    LEFT JOIN orders o ON u.id = o.user_id
    LEFT JOIN models s ON o.scooter_id = s.id
    WHERE u.id = ?
";

$stmt = sqlsrv_query($this->conn, $sql, [$user_id]);
if (!$stmt) {
    throw new Exception("Query failed: getUser", 500);
}
$result = [];
while ($row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC)) {
    $result[] = $row;
}

if (empty($result)) {
    throw new Exception("User not found", 404);
}

$user = [
    'profile' => [
        'name' => $result[0]['name'],
        'email' => $result[0]['email'],
        'age' => $result[0]['age'],
        'wallet' => $result[0]['wallet'],
        'location' => [
            'x' => $result[0]['location_x'],
            'y' => $result[0]['location_y']
```

```
            ],
            'admin' => $result[0]['is_admin'],
        ],
        'orders' => []
    ];

    foreach ($result as $row) {
        if ($row['order_id']) {
            $timeStr = is_object($row['time']) ? $row['time']->format('Y-m-d H:i:s') : $row['time'];
            $user['orders'][] = [
                'order_id' => $row['order_id'],
                'status' => $row['status'],
                'scooter_id' => $row['scooter_id'],
                'price' => $row['price'],
                'time' => $timeStr,
                'duration' => $row['duration'],
                'scooter_name' => $row['scooter_name']
            ];
        }
    }
    return $user;
}



// public function getUser($user_id) {
//     if (empty($user_id)) {
//         throw new Exception("Missing user_id", 400);
//     }

//     $sql = "
//         SELECT u.name, u.email, u.age, u.wallet,
//                ul.location_x, ul.location_y, u.is_admin,
//                o.id AS order_id, o.status, o.price, o.time
//         FROM users u
//         LEFT JOIN user_location ul ON u.id = ul.user_id
```

```
//         LEFT JOIN orders o ON u.id = o.user_id
//         WHERE u.id = ?
//     ";

//     $stmt = sqlsrv_query($this->conn, $sql, [$user_id]);
//     if (!$stmt) {
//         throw new Exception("Query failed: getUser", 500);
//     }
//     $result = [];
//     while ($row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC)) {
//         $result[] = $row;
//     }

//     if (empty($result)) {
//         throw new Exception("User not found", 404);
//     }

//     $user = [
//         'profile' => [
//             'name' => $result[0]['name'],
//             'email' => $result[0]['email'],
//             'age' => $result[0]['age'],
//             'wallet' => $result[0]['wallet'],
//             'location' => [
//                 'x' => $result[0]['location_x'],
//                 'y' => $result[0]['location_y']
//             ],
//             'admin' => $result[0]['is_admin'],
//         ],
//         'orders' => []
//     ];
//     foreach ($result as $row) {
//         if ($row['order_id']) {
//             $timeStr = is_object($row['time']) ? $row['time']->format('Y-m-d H:i:s') : $row['time'];
//             $user['orders'][] = [
```

```
//                 'order_id' => $row['order_id'],
//                 'status' => $row['status'],
//                 'price' => $row['price'],
//                 'time' => $timeStr
//             ];
//         }
//     }
//     return $user;
// }

public function login($email, $password) {
    if (empty($email) || empty($password)) {
        throw new Exception("Missing credentials", 400);
    }

    $sql = "SELECT * FROM users WHERE email = ?";
    $stmt = sqlsrv_query($this->conn, $sql, [$email]);
    if (!$stmt) {
        throw new Exception("Query error", 500);
    }

    $user = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC);
    if (!$user) {
        throw new Exception("User not found", 404);
    }

    if (password_verify($password, $user['password'])) {
        // session_start();
        $_SESSION['user_id'] = $user['id'];
        $_SESSION['is_admin'] = (bool)$user['is_admin'];

        unset($user['password']);
        return ["success" => true, "user" => $user];
    } else {
        throw new Exception("Incorrect password", 401);
```

```
    }
  }

  public function register($name, $email, $age, $password) {
    if (empty($name) || empty($email) || empty($password) || empty($age)) {
      throw new Exception("Missing required fields", 400);
    }

    sqlsrv_begin_transaction($this->conn);

    try {
      // Проверка на существующий email
      $checkSql = "SELECT 1 FROM users WHERE email = ?";
      $checkStmt = sqlsrv_query($this->conn, $checkSql, [$email]);
      if ($checkStmt === false || sqlsrv_fetch_array($checkStmt, SQLSRV_FETCH_ASSOC)) {
        throw new Exception("User with this email already exists", 400);
      }

      // Хеширование пароля
      $hashedPassword = password_hash($password, PASSWORD_DEFAULT);

      // Вставка пользователя
      $insertUserSql = "INSERT INTO users (name, email, password, age, wallet, is_admin) VALUES (?, ?, ?, ?, 0, 0)";
      $userParams = [$name, $email, $hashedPassword, $age];
      $userStmt = sqlsrv_query($this->conn, $insertUserSql, $userParams);
      if (!$userStmt) {
        throw new Exception("User registration failed", 500);
      }

      // Получение ID пользователя
      $userIdResult = sqlsrv_query($this->conn, "SELECT SCOPE_IDENTITY() AS user_id");
      $userIdRow = sqlsrv_fetch_array($userIdResult, SQLSRV_FETCH_ASSOC);
      $userId = $userIdRow['user_id'];
```

```
        sqlsrv_commit($this->conn);

        return ["success" => true, "user_id" => $userId];
    } catch (Exception $e) {
        sqlsrv_rollback($this->conn);
        throw $e;
    }
}

// public function register($name, $email, $password) {
//     if (empty($name) || empty($email) || empty($password)) {
//         throw new Exception("Missing required fields", 400);
//     }

//     $sql = "SELECT * FROM users WHERE email = ?";
//     $stmt = sqlsrv_query($this->conn, $sql, [$email]);
//     if (sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC)) {
//         throw new Exception("User with this email already exists", 400);
//     }

//     $hashedPassword = password_hash($password, PASSWORD_DEFAULT);

//     $sql = "INSERT INTO users (name, email, password) VALUES (?, ?, ?)";
//     $params = [$name, $email, $hashedPassword];
//     $stmt = sqlsrv_query($this->conn, $sql, $params);
//     if ($stmt === false) {
//         throw new Exception("Registration failed", 500);
//     }

//     $sql = "SELECT SCOPE_IDENTITY() AS user_id";
//     $stmt = sqlsrv_query($this->conn, $sql);
//     $row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC);

//     return ["success" => true, "user_id" => $row['user_id']];
// }
```

```
}
```