

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Розробка сервісу для оптимізації логістичних
маршрутів для малого і середнього бізнесу»**

Виконав студент групи ІПЗм-24-1
РЯБОВА Єлизавета Вадимівна

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
ВАКАЛЮК Тетяна Анатоліївна

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«18» вересня 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **РЯБОВА Єлизавета Вадимівна**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Розробка сервісу для оптимізації логістичних маршрутів для малого і середнього бізнесу»,**

затверджена Наказом закладу вищої освіти від **«18» вересня 2025 р. № 491/с**

Вихідні дані для роботи: **оптимізація маршрутів доставки «останньої милі» з урахуванням вантажопідйомності транспортних засобів; математичні моделі та методи розв'язання задачі маршрутизації з використанням стратегії декомпозиції; алгоритми кластеризації замовлень та евристичної оптимізації методом табу-пошуку; проєктування архітектури веб-сервісу на базі технологій .NET та Python; автоматизація планування перевезень та візуалізація маршрутів на інтерактивній карті.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Власенко О.В.	22.09.2025р.	22.09.2025р.
2	Власенко О.В.	22.09.2025р.	22.09.2025р.
3	Власенко О.В.	22.09.2025р.	22.09.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота магістра складається з опису дослідження методів оптимізації вантажних перевезень, розробки гібридного алгоритму маршрутизації, створення веб-орієнтованої інформаційної системи підтримки прийняття рішень та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи складається з 75 сторінок, 38 ілюстрацій, 1 додатку та 30 літературних джерел.

Мета роботи – підвищення ефективності управління вантажними перевезеннями шляхом розробки гібридного алгоритму маршрутизації та реалізації на його основі веб-орієнтованої інформаційної системи підтримки прийняття рішень.

У роботі було проведено аналіз сучасного стану транспортної логістики, виявлено ключові проблеми етапу «останньої милі» та проаналізовано існуючі програмні аналоги. Досліджено математичні моделі задачі маршрутизації (VRP) та здійснено порівняльний аналіз методів її розв’язання. Розроблено гібридний алгоритм побудови маршрутів на основі стратегії «Cluster-First, Route-Second», що поєднує алгоритм кластеризації з фіксованими центрами та адаптивний табу-пошук. Також удосконалено архітектуру систем підтримки прийняття рішень у логістиці шляхом застосування гібридного мікросервісного підходу (Python для обчислень, .NET для бізнес-логіки). Спроектовано та реалізовано веб-сервіс, що забезпечує автоматизований розрахунок маршрутів, візуалізацію результатів на карті та управління довідниками логістичної мережі. У висновках було отримано результати досліджень та розробок за час виконання кваліфікаційної роботи.

КЛЮЧОВІ СЛОВА: МАРШРУТИЗАЦІЯ, ОСТАННЯ МИЛЯ, МЕТАЕВРИСТИКА, КЛАСТЕРИЗАЦІЯ, ВЕБ-СЕРВІС, ОПТИМІЗАЦІЯ.

ABSTRACT

The Master's Qualification Thesis consists of a description of research into freight transportation optimization methods, the development of a hybrid routing algorithm, the creation of a web-oriented decision support information system, and an explanatory note. The explanatory note to the qualification thesis comprises 75 pages, 38 illustrations, 1 appendix, and 30 literary sources.

The goal of the work is to increase the efficiency of freight transportation management by developing a hybrid routing algorithm and implementing a web-oriented decision support information system based on it.

In this work, an analysis of the current state of transport logistics was conducted, key challenges of the "last mile" stage were identified, and existing software analogues were analyzed. Mathematical models of the Vehicle Routing Problem (VRP) were investigated, and a comparative analysis of methods for its solution was performed. A hybrid route construction algorithm was developed based on the "Cluster-First, Route-Second" strategy, which combines a fixed-center clustering algorithm with adaptive tabu search. Furthermore, the architecture of decision support systems in logistics was refined through the application of a hybrid microservice approach (Python for calculations, .NET for business logic). A web service was designed and implemented, which ensures automated route calculation, results visualization on a map, and management of logistics network directories. Finally, the research and development results obtained during the execution of the qualification thesis were presented in the conclusions.

KEYWORDS: ROUTING, LAST MILE, METAHEURISTICS, CLUSTERING, WEB SERVICE, OPTIMIZATION.

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ЗМІСТ	5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ТА МЕТОДИ ОПТИМІЗАЦІЇ ЛОГІСТИЧНИХ МАРШРУТІВ	10
1.1. Аналіз стану транспортної логістики та VRP	10
1.2. Класифікація моделей та методів розв’язання VRP	12
1.3. Аналіз програмних засобів автоматизації логістики.....	14
1.4. Постановка задачі дослідження.....	21
Висновки до розділу 1	23
РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА РОЗРОБКА КОМПЛЕКСНОГО АЛГОРИТМУ МАРШРУТИЗАЦІЇ	25
2.1. Концептуальна модель системи та декомпозиція задачі	25
2.2. Дослідження методів кластеризації та евристичної оптимізації	28
2.3. Побудова узагальненого алгоритму оптимізації маршрутів	40
Висновки до розділу 2	48
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВІСУ ДЛЯ ОПТИМІЗАЦІЇ ЛОГІСТИЧНИХ МАРШРУТІВ	50
3.1. Архітектура програмного комплексу та вибір засобів розробки.....	50
3.2. Інтерфейс користувача та сценарії роботи з системою.....	56
Висновки до розділу 3	64
ВИСНОВКИ.....	65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	67
ДОДАТКИ.....	71

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СКБД – Система керування базами даних.

API – Application Programming Interface (інтерфейс прикладного програмування).

CVRP – Capacitated Vehicle Routing Problem (задача маршрутизації транспортних засобів з обмеженням вантажопідйомності).

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту).

JSON – JavaScript Object Notation (текстовий формат обміну даними).

ORM – Object-Relational Mapping (об'єктно-реляційне відображення).

REST – Representational State Transfer (архітектурний стиль взаємодії компонентів розподіленого додатка в мережі).

SPA – Single Page Application (односторінковий веб-застосунок).

TSP – Traveling Salesman Problem (задача комівояжера).

VRP – Vehicle Routing Problem (задача маршрутизації транспортних засобів).

ВСТУП

Сучасний етап розвитку світової економіки характеризується урбанізацією, глобалізацією ринків та зростанням сектору електронної комерції. Ці процеси докорінно змінюють структуру логістичних потоків, зміщуючи акцент з магістральних перевезень на адресну доставку «останньої милі». Саме цей завершальний етап є найбільш ресурсомістким у загальній вартості логістичного ланцюга.

В умовах ринкової конкуренції та підвищених вимог споживачів до швидкості й точності отримання замовлень, традиційні методи планування маршрутів втрачають свою ефективність. Логісти стикаються з проблемою обробки великих масивів даних, необхідністю врахування множини обмежень та високою динамікою змін у замовленнях. Помилки на етапі планування призводять до нераціонального використання автопарку, надмірних витрат палива, збільшення шкідливих викидів в атмосферу та зниження рівня клієнтського сервісу.

Існуючі точні методи розв'язання VRP є ефективними лише для невеликої кількості точок, проте при масштабуванні до реальних розмірів логістичних мереж вони вимагають неприйнятних обчислювальних ресурсів. Це зумовлює необхідність розробки та вдосконалення евристичних та метаевристичних алгоритмів, які здатні знаходити субоптимальні рішення за прийнятний час.

Особливої актуальності набуває питання програмної реалізації таких алгоритмів у вигляді веб-сервісів, які поєднують математичний апарат зі зручним інтерфейсом користувача. Розробка автоматизованої системи, що використовує стратегію декомпозиції задачі та сучасні методи локального пошуку, є важливою науково-прикладною задачею, вирішення якої дозволить підвищити рентабельність транспортних підприємств.

Метою роботи є підвищення ефективності управління вантажними перевезеннями шляхом розробки гібридного алгоритму маршрутизації та

реалізації на його основі веб-орієнтованої інформаційної системи підтримки прийняття рішень.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Провести аналіз сучасного стану транспортної логістики, виявити ключові проблеми етапу «останньої милі» та проаналізувати існуючі програмні аналоги.
2. Дослідити математичні моделі задачі маршрутизації та здійснити порівняльний аналіз методів її розв'язання.
3. Розробити алгоритм побудови маршрутів, що базується на комбінації методів кластеризації та метаевристичної оптимізації.
4. Спроекувати архітектуру програмного комплексу, обґрунтувати вибір технологічного стека та розробити структуру бази даних.
5. Реалізувати веб-сервіс, що забезпечує автоматизований розрахунок маршрутів, візуалізацію результатів на карті та управління довідниками логістичної мережі.

Об'єкт дослідження – процес управління маршрутизацією вантажних перевезень в умовах міської логістики.

Предмет дослідження – моделі, методи та алгоритми оптимізації розподілу замовлень і побудови маршрутів транспортних засобів з урахуванням обмежень вантажопідйомності.

У роботі використано комплексний підхід, що включає:

1. методи системного аналізу – для визначення вимог до системи та проєктування її архітектури;
2. теорію графів – для математичного моделювання дорожньої мережі та розрахунку найкоротших шляхів;
3. методи кластеризації даних – для декомпозиції глобальної задачі маршрутизації на локальні підзадачі;
4. метаевристичні методи – для ітеративної оптимізації маршрутів всередині кластерів;

5. методи об'єктно-орієнтованого проєктування та веб-програмування – для реалізації клієнт-серверної архітектури програмного засобу.

Інформаційну основу роботи склали наукові праці вчених у галузі дослідження операцій і транспортної логістики, матеріали спеціалізованих видань, відкриті картографічні дані та технічна документація використаних програмних бібліотек і фреймворків.

Наукова новизна одержаних результатів полягає в удосконаленні методу розв'язання задачі маршрутизації транспортних засобів на основі стратегії «Cluster-First, Route-Second». У роботі запропоновано поєднання алгоритму кластеризації з фіксованими центрами та адаптивного табу-пошуку з використанням оптимізованої метрики гаверсінуса, що дозволило зменшити обчислювальну складність алгоритму при збереженні точності результатів. Крім того, набула подальшого розвитку архітектура систем підтримки прийняття рішень у логістиці шляхом застосування гібридного мікросервісного підходу, де математичні обчислення делегуються окремому модулю мовою Python, а бізнес-логіка обробляється основним сервером на платформі .NET.

Практичне значення одержаних результатів визначається розробкою та програмною реалізацією веб-сервісу для оптимізації логістичних маршрутів. Впровадження цієї системи дозволяє автоматизувати процес розподілу замовлень між транспортними засобами, скоротити сумарний пробіг автопарку та зменшити транспортні витрати. Функціональні можливості сервісу забезпечують візуалізацію планових маршрутів на інтерактивній карті, а також дозволяють мінімізувати вплив людського фактора при плануванні перевезень. Створений програмний продукт може бути використаний транспортними та логістичними компаніями, службами доставки та дистриб'юторами для підвищення загальної ефективності операційної діяльності.

За результатами виконаних досліджень опубліковано 1 тези [1].

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ТА МЕТОДИ ОПТИМІЗАЦІЇ ЛОГІСТИЧНИХ МАРШРУТІВ

1.1. Аналіз стану транспортної логістики та VRP

Сучасний етап розвитку економіки характеризується стрімкою цифровізацією та глобальним зростанням сектору електронної комерції, що докорінно змінює структуру вантажопотоків у бік адресної доставки «останньої милі». Трансформація споживчої поведінки, зокрема попит на послуги «день у день» та дотримання чітких часових вікон, перетворює логістичну ефективність на критичний фактор конкурентоспроможності. Це змушує підприємства відмовлятися від традиційних моделей постачання на користь гнучких систем, здатних забезпечити високу точність та швидкість обслуговування клієнтів.

Ситуація ускладнюється процесами урбанізації, що призводять до критичного навантаження на дорожню інфраструктуру та появи адміністративних обмежень у великих містах. Висока динаміка дорожнього трафіку та непередбачуваність заторів роблять ручні методи планування неефективними, оскільки вони не здатні адаптуватися до змін у реальному часі. Це зумовлює необхідність впровадження автоматизованих систем управління маршрутами, які дозволяють мінімізувати транспортні витрати та оптимізувати використання автопарку в умовах щільної міської забудови.

Центральним викликом у сучасній логістиці є так звана проблема «останньої милі» – завершальний етап ланцюга постачання, що передбачає доставку товару від розподільчого центру безпосередньо до кінцевого споживача. Попри те, що цей відрізок є найкоротшим за відстанню, він залишається найбільш ресурсомістким та дорогим елементом логістичного циклу [2]. За оцінками експертів, витрати на організацію «останньої милі» можуть сягати до 53% від загальної вартості доставки, що робить оптимізацію цього етапу критично важливою для рентабельності бізнесу [3].

Висока вартість та складність даного процесу зумовлені низкою об'єктивних факторів неефективності. Перш за все, це значна географічна

розпорошеність точок доставки, що змушує транспортні засоби здійснювати складні маневри та долати великі відстані між зупинками заради передачі невеликих вантажів. Ситуацію ускладнює непередбачуваність дорожніх заторів у мегаполісах, яка призводить до порушення графіків та збільшення часу перебування в дорозі. Додатковим фактором втрат є вимушені простої транспорту під час розвантаження, очікування клієнта або пошуку місця для паркування, що знижує коефіцієнт корисного використання рухомого складу [4].

Історично управління транспортною логістикою здійснювалося переважно вручну, покладаючись на досвід та інтуїцію диспетчерів. Традиційний підхід до планування маршрутів характеризується значною залежністю від людського фактора, що неминуче призводить до помилок та субоптимальних рішень [5]. Логіст фізично не здатен одночасно врахувати сотні динамічних змінних: пропускну здатність доріг, часові вікна клієнтів, вантажопідйомність різних автомобілів та графіки роботи водіїв. Крім того, ручне формування маршрутних листів є надзвичайно трудомістким процесом, який може займати декілька годин щодня, відволікаючи персонал від стратегічних завдань [6].

Зі зростанням масштабів бізнесу та кількості замовлень неефективність ручного управління стає критичним бар'єром для розвитку. Це зумовлює об'єктивну необхідність переходу до використання автоматизованих систем управління транспортом. Впровадження спеціалізованого програмного забезпечення дозволяє не лише мінімізувати час на планування з годин до хвилин, але й забезпечує математичну точність розрахунків, гарантуючи дотримання всіх логістичних обмежень та суттєве скорочення експлуатаційних витрат [7].

Ключовим аргументом на користь автоматизації логістичних процесів є безпосередній вплив якості маршрутизації на собівартість транспортних послуг. Ефективна побудова маршрутів дозволяє досягти суттєвого скорочення сумарного пробігу автопарку, що прямо корелює зі зменшенням

витрат на паливно-мастильні матеріали та технічне обслуговування транспортних засобів [8]. Окрім прямої економії ресурсів, оптимізація сприяє зниженню амортизаційних відрахувань та подовженню терміну експлуатації рухомого складу. Важливим економічним ефектом є також підвищення оборотності транспорту: за рахунок ущільнення графіка доставки один автомобіль здатен обслуговувати більшу кількість замовлень за зміну, що дозволяє підприємству виконувати зростаючі обсяги роботи без необхідності розширення автопарку.

Практична необхідність мінімізації логістичних витрат та підвищення рівня обслуговування клієнтів вимагає формалізації управлінських процесів мовою точних наук. У контексті автоматизації реальна бізнес-задача організації доставки трансформується з організаційної площини в математичну задачу знаходження оптимальних траєкторій руху на графі дорожньої мережі. Такий підхід дозволяє абстрагуватися від фізичної природи перевезень та застосувати універсальний алгоритмічний апарат для вирішення прикладних проблем [9]. Це створює передумови для моделювання логістичних процесів як варіацій класичних комбінаторних задач, де критерії ефективності бізнесу (час, гроші) виражаються через цільові функції мінімізації вартості маршруту.

1.2. Класифікація моделей та методів розв'язання VRP

Фундаментом сучасних геоінформаційних та логістичних систем є математичний апарат теорії графів, де транспортна мережа формалізується як зважений граф $G=(V,E)$. У цій моделі множина вершин V представляє ключові об'єкти логістичної інфраструктури, такі як склади, точки доставки та транспортні розв'язки, а множина ребер E відображає дорожні зв'язки між ними. Кожному ребру приписується вагова характеристика, яка може виражатися у фізичній відстані, часі проходження маршруту або вартості перевезення [10].

Першочерговою задачею у процесі маршрутизації є визначення вартості переміщення між вузлами графа, що зводиться до задачі пошуку найкоротшого шляху. Класичним підходом до її вирішення є використання точних алгоритмів, еталоном серед яких вважається алгоритм Дейкстри. Цей метод належить до категорії жадібних алгоритмів і гарантує знаходження оптимального маршруту шляхом послідовного розширення області пошуку від початкової вершини до всіх інших. Незважаючи на гарантовану точність та незалежність від структури простору, алгоритм Дейкстри має суттєвий недолік – високу обчислювальну складність, оскільки він змушений перевіряти значну кількість вузлів, що робить його малоефективним для роботи з масштабними дорожніми графами міст у реальному часі [11].

Більш адаптованим для прикладних навігаційних задач є алгоритм A^* , який є модифікацією методу Дейкстри, що використовує евристичну функцію для спрямування пошуку в бік цільової вершини. У контексті транспортних задач найчастіше застосовується евристика на основі формули гаверсінуса (Haversine), яка розраховує геодезичну відстань між координатами точок. Такий підхід дозволяє відсікати неперспективні напрямки та суттєво скорочувати простір пошуку, забезпечуючи значний приріст швидкодії при збереженні прийнятної точності результату [12]. При масштабуванні задачі від пошуку шляху між двома точками до необхідності об'їзду множини клієнтів, проблема трансформується у класичну задачу комівояжера, метою якої є побудова найкоротшого замкненого маршруту, що проходить через кожну точку рівно один раз. У реальних логістичних сценаріях задача ускладнюється додатковими обмеженнями: наявністю парку транспортних засобів, їхньою вантажопідйомністю, часовими вікнами доставки та прив'язки до кількох складів, що визначається як задача маршрутизації транспортних засобів [13].

Оскільки VRP належить до класу NP-важких задач комбінаторної оптимізації, застосування точних методів є неефективним через експоненціальне зростання часу обчислень при збільшенні кількості точок, тому основна увага дослідників зосереджена на наближених методах. До них

належать конструктивні евристичні, що формують маршрут послідовно, крок за кроком (наприклад, метод «найближчого сусіда»), та метавристичні алгоритми. Серед метавристичних виділяють генетичні алгоритми, що базуються на принципах еволюційного відбору та демонструють високу швидкість, мурашині алгоритми, які імітують колективну поведінку комах та забезпечують високу адаптивність, а також метод табу-пошуку. Останній реалізує стратегію локального перебору з використанням механізму короткострокової пам'яті («табу-списків») для уникнення зациклень, забезпечуючи оптимальний баланс між точністю та швидкістю [1, 14].

Для зменшення розмірності задачі ефективним є застосування стратегії декомпозиції «Cluster-First, Route-Second», яка полягає у попередньому розбитті множини точок на групи за допомогою алгоритмів кластеризації. Класичний алгоритм K-Means забезпечує швидке групування, але може створювати нерівномірні за розміром кластери. У логістиці часто застосовується модифікація з фіксованими центрами, де центроїди кластерів жорстко прив'язуються до координат реальних складів, що дозволяє уникнути абстрактних центрів обслуговування. Також перспективним є метод збалансованої кластеризації, який, окрім географічної близькості, враховує обмеження на кількість точок у кластері для рівномірного розподілу навантаження між транспортними засобами [15]. Сучасним трендом є використання гібридних підходів, що поєднують методи декомпозиції для спрощення структури задачі та метавристичні алгоритми для фінальної оптимізації маршрутів у межах виділених зон, що дозволяє досягти компромісу між складністю та якістю планування.

1.3. Аналіз програмних засобів автоматизації логістики

Ринок сучасного програмного забезпечення для транспортної логістики перебуває на стадії активного розвитку, пропонуючи широкий спектр рішень, що варіюються від простих мобільних планувальників до комплексних

корпоративних платформ. Сучасні тенденції в цій сфері свідчать про поступовий відхід від статичних настільних додатків у бік хмарних SaaS-рішень, які забезпечують доступ до обчислювальних ресурсів та алгоритмів штучного інтелекту без необхідності розгортання власної серверної інфраструктури. В основі більшості передових продуктів лежать складні комбінації евристичних методів, алгоритмів пошуку найкоротшого шляху на графах та технологій машинного навчання, спрямованих на вирішення класичних задач маршрутизації з урахуванням динамічних обмежень реального часу, таких як затори, часові вікна доставки та специфікації вантажу.

Одним із найбільш показових прикладів таких систем є глобальна хмарна платформа Route4Me, яка орієнтована на підприємства різного масштабу – від локальних служб доставки до транснаціональних логістичних операторів. Згідно з технічною документацією та патентною базою компанії, ключовою перевагою сервісу є використання запатентованої технології SmartZone Routing. Цей підхід базується на методах адаптивної кластеризації та динамічного програмування, що дозволяє автоматично розбивати великі масиви адрес на оптимальні географічні зони обслуговування, закріплюючи за кожним водієм певну територію для підвищення ефективності «останньої милі». Окрім геометричної оптимізації, система активно застосовує методи машинного навчання для прогнозування часу прибуття, аналізуючи історичні дані про трафік та час обслуговування клієнтів у подібних локаціях, що дозволяє мінімізувати похибки планування.

Аналізуючи переваги Route4Me, варто відзначити високу швидкість генерації маршрутів та унікальну модульну архітектуру, яка дозволяє користувачам самостійно конструювати функціонал системи, додаючи або відключаючи окремі модулі залежно від потреб бізнесу. Така гнучкість робить платформу універсальною, проте водночас породжує низку недоліків. Зокрема, надмірна насиченість функціоналом та складність інтерфейсу часто призводять до того, що новим користувачам потрібен значний час для

навчання та адаптації до системи. Крім того, критичним моментом залишається залежність мобільного додатку від стабільності інтернет-з'єднання та GPS-сигналу, що може негативно впливати на точність виконання маршрутів у зонах зі слабким покриттям мережі або щільною міською забудовою

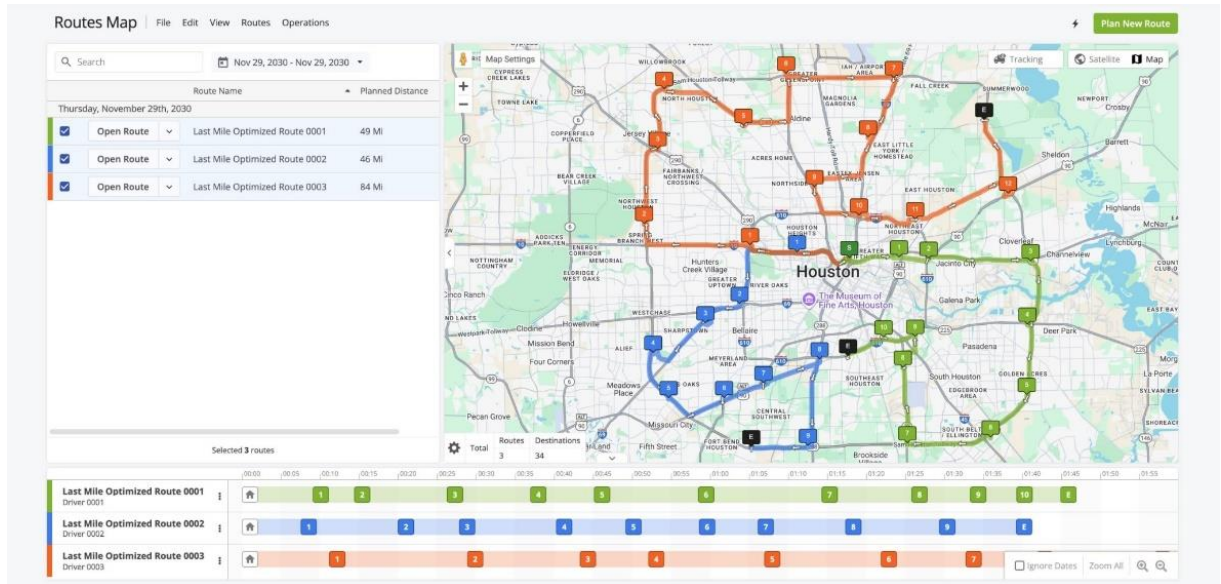


Рис.1.1. Інтерфейс застосунку Route4Me.

У сегменті рішень, орієнтованих на оптимізацію доставки «останньої милі» для середніх автопарків із навантаженням до 500 точок на маршрут, значного поширення набув інструмент Zeo Route Planner. Ця платформа позиціонує себе як доступне рішення, алгоритмічна база якого спирається на сучасні алгоритми на основі штучного інтелекту. Технічно ядро системи реалізує гібридну модель, що поєднує модифіковані евристичні «найближчого сусіда» для швидкої генерації початкового плану з методами динамічної перевірки обмежень, таких як часові вікна доставки та пріоритетність клієнтів. Така архітектура дозволяє системі не лише оперативно розподіляти замовлення між водіями, але й здійснювати «розумний» динамічний перерахунок маршрутів у режимі реального часу. Це означає, що у випадку раптової зміни дорожньої ситуації, скасування замовлення або затримки водія, алгоритм автоматично перебудовує послідовність наступних зупинок для мінімізації загального запізнення.

Серед суттєвих переваг Zeo Route Planner варто виділити акцент на мобільності та зручності інтерфейсу, який розроблений з урахуванням потреб водіїв, а не лише диспетчерів, що спрощує впровадження системи на підприємстві. Однак, попри високу автоматизацію, коректність роботи сервісу має пряму і критичну кореляцію з якістю картографічних даних, які використовуються для геокодування адрес. В окремих випадках, особливо в умовах щільної міської забудови, нових житлових масивів або некоректно внесених адрес у глобальних базах, система може генерувати маршрути з помилками у фінальній точці прибуття. Це створює ситуації, коли автоматично побудований маршрут потребує ручної верифікації та втручання диспетчера для коригування локації, що дещо знижує рівень повної автономності процесу планування.

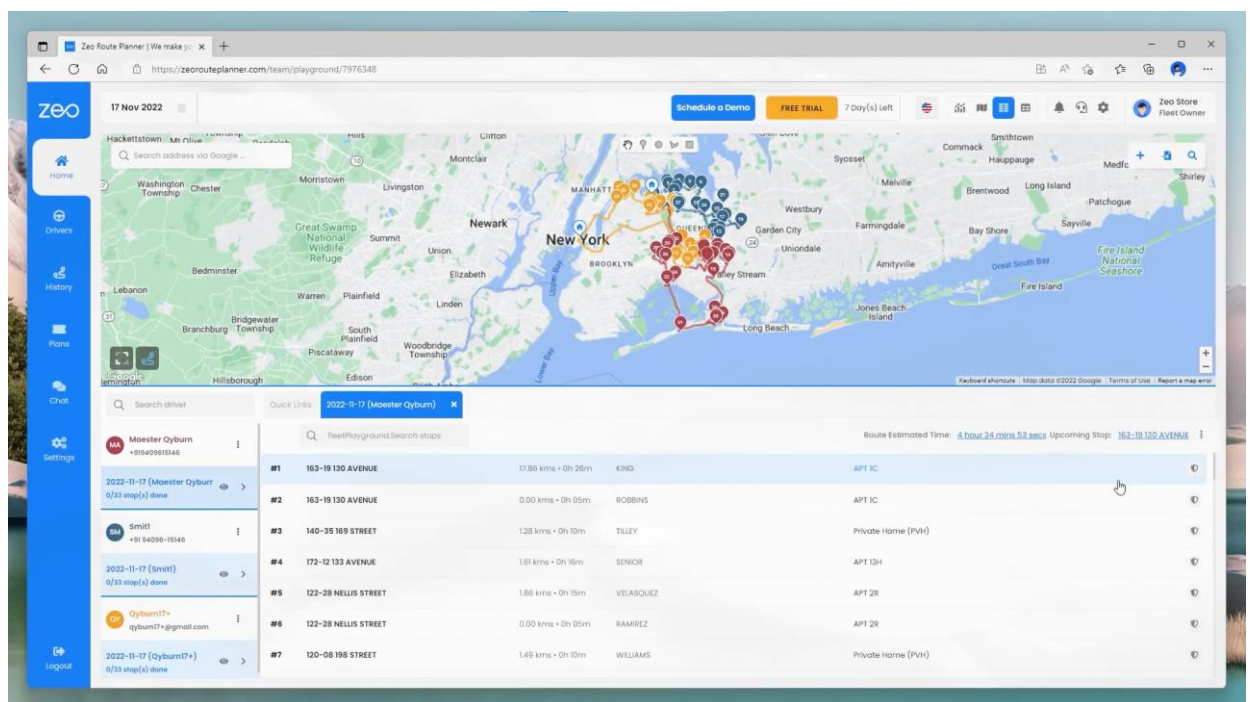


Рис.1.2. Інтерфейс застосунку Zeo Route Planner.

У сегменті професійних рішень для управління масштабними автопарками та складними багатоланковими логістичними ланцюгами окреме місце посідає програмний комплекс RoadWarrior. Ця платформа розроблена з фокусом на централізоване диспетчерське управління, дозволяючи координувати роботу десятків водіїв одночасно з урахуванням поточної інтенсивності дорожнього руху. Унікальною технічною особливістю

архітектури RoadWarrior є використання спеціалізованих генетичних алгоритмів для вирішення класичної задачі комівояжера та її варіацій. На відміну від детермінованих методів перебору, цей еволюційний підхід дозволяє знаходити субоптимальні, тобто достатньо якісні для практичного застосування рішення для маршрутів із сотнями точок зупинки значно швидше, ніж це роблять класичні точні алгоритми. Це робить систему незамінною в ситуаціях, коли необхідно оперативно перепланувати логістику для великої кількості транспортних засобів в умовах дефіциту часу.

Тим не менш, орієнтація на професійний сектор формує і специфічні недоліки системи. Намагання розробників надати користувачеві максимальний контроль над параметрами маршрутизації призвело до значного ускладнення інтерфейсу управління, що створює високий поріг входження для нових співробітників та вимагає часу на освоєння всіх інструментів. Крім того, незважаючи на сильні сторони математичного апарату, автоматично згенеровані маршрути не завжди є ідеальними з точки зору водія-практика. Алгоритм, працюючи з цифровою картою, може не враховувати специфічні локальні нюанси, такі як зручність під'їзду до рампи, якість дорожнього покриття у дворах або неформальні правила проїзду територією підприємств. Через це робота з RoadWarrior часто потребує експертної валідації та ручного коригування маршрутів досвідченим логістом, що унеможлиблює повну автоматизацію процесу без людського нагляду.

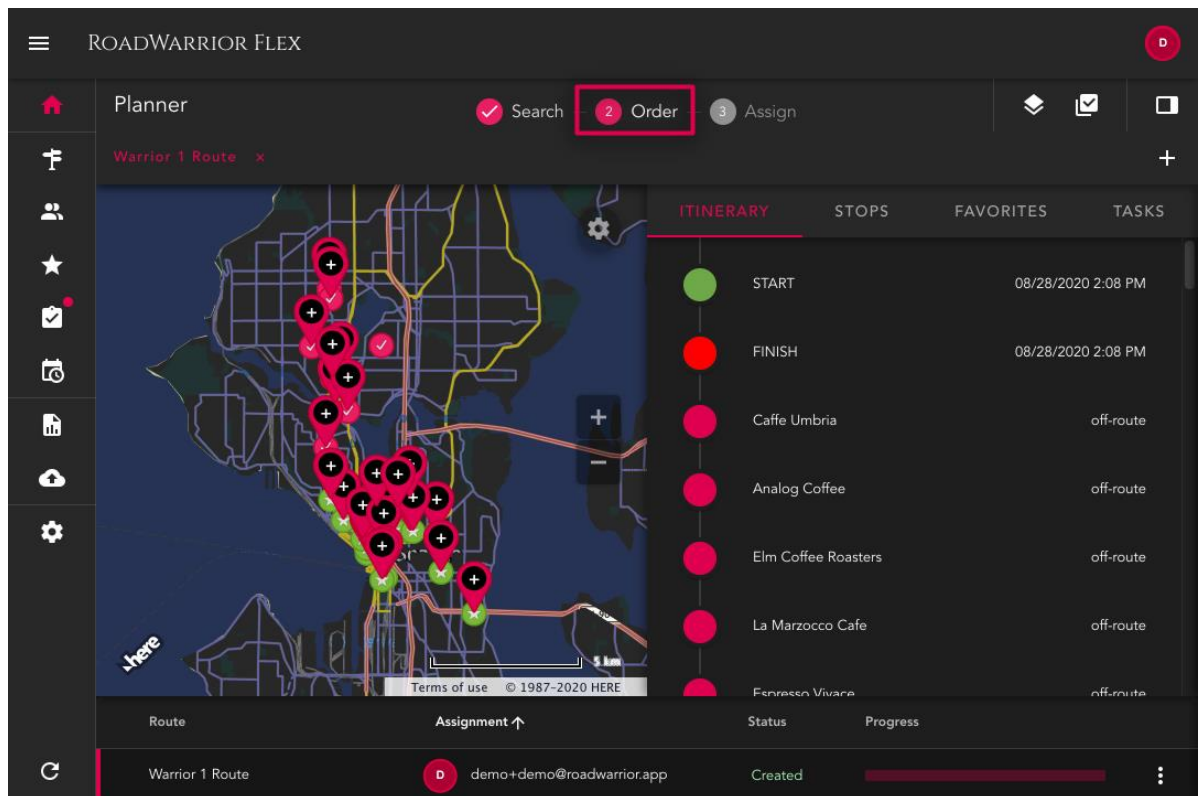


Рис.1.3. Інтерфейс застосунку RoadWarrior.

Для великих дистриб'юторських компаній та підприємств зі значним товарообігом одним із найбільш ефективних інструментів на ринку є комплексна хмарна платформа ANT-Logistics. Як свідчать назва системи та пов'язані з нею наукові публікації, її математичним ядром виступає алгоритм мурашиної колонії. Цей підхід імітує природну поведінку соціальних комах, які використовують феромонні сліди для колективного пошуку найкоротших шляхів від мурашника до джерела їжі. У контексті програмного забезпечення це дозволяє віртуальним агентам знаходити глобальні оптимуми у складних графах транспортних мереж. Часто цей метод комбінується з іншими метаевристиками локального пошуку, що забезпечує високу точність результатів при врахуванні множини жорстких обмежень, таких як вагогабаритні характеристики вантажу, сумісність товарів та вузькі часові вікна доставки.

Водночас, орієнтація на великий бізнес визначає специфіку архітектури та впровадження системи. ANT-Logistics позиціонується не просто як маршрутизатор, а як частина корпоративної екосистеми, що вимагає

двосторонньої інтеграції з обліковими ERP-системами та сервісами GPS-моніторингу. Такий підхід забезпечує повну автоматизацію циклу доставки, проте створює високий поріг входження. Високий рівень складності налаштування бізнес-правил та параметризації алгоритму, необхідність тривалого етапу впровадження та потреба у спеціальному навчанні персоналу роблять це рішення фінансово та організаційно важким. Для представників малого та середнього бізнесу, які потребують швидкого старту та інтуїтивної зрозумілості, така надлишкова функціональність часто стає нераціональною інвестицією.

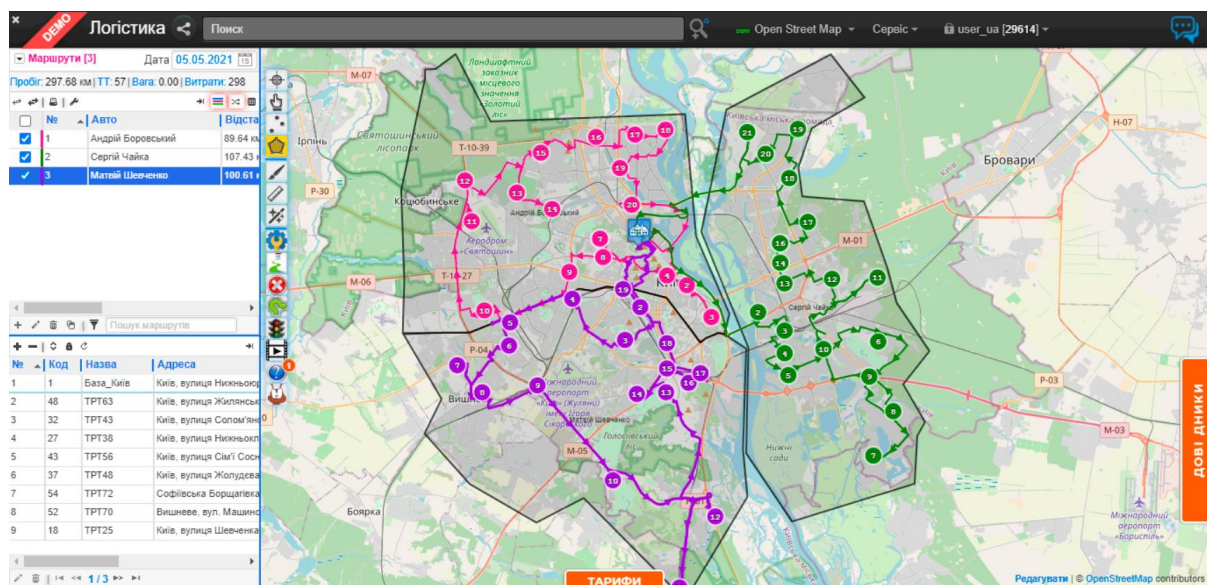


Рис.1.4. Інтерфейс застосунку ANT-Logistics.

Узагальнюючи результати проведеного аналізу ринку програмного забезпечення, можна сказати, що провідні комерційні платформи досягли високого рівня технологічної зрілості, активно імплементуючи складні математичні апарати, такі як генетичні алгоритми, метод мурашиної колонії та гібридні моделі кластеризації. Ці системи орієнтовані насамперед на мінімізацію логістичних витрат у масштабах великих дистриб'юторських мереж, пропонуючи користувачам різноманітний, але часто надлишковий функціонал. Висока вартість ліцензій, складність інтеграції з існуючими бізнес-процесами та необхідність тривалого навчання персоналу створюють високий поріг входження, роблячи такі рішення економічно недоцільними для сегмента малого та середнього бізнесу.

Водночас, на ринку спостерігається виразний дефіцит доступних веб-сервісів, які б змогли заповнити нішу між ручним плануванням та дорогими корпоративними системами. Існує потреба у створенні адаптивних інструментів, які б поєднували ефективність сучасних алгоритмів оптимізації із простотою користувацького інтерфейсу. Особливий інтерес у цьому контексті представляє використання методу табу-пошуку, який, на відміну від ресурсомістких еволюційних алгоритмів, здатен забезпечити оптимальний баланс між швидкістю розрахунків та якістю маршрутів для локальних перевізників.

Виявлений дисбаланс між пропозицією ринку та реальними потребами невеликих логістичних операторів слугує вагомим обґрунтуванням доцільності розробки власної інформаційної системи. Створення спеціалізованого веб-сервісу дозволить реалізувати гнучку архітектуру, адаптовану до конкретних технічних вимог замовника, та впровадити модифіковані алгоритми маршрутизації, що враховують специфіку міської логістики без зайвого ускладнення процесу експлуатації.

1.4. Постановка задачі дослідження

Узагальнюючи результати аналізу, проведеного у попередніх підрозділах, можна стверджувати, що на ринку логістичного програмного забезпечення сформувалася технологічна прогалина. Існуючі комерційні платформи часто є функціонально надлишковими або фінансово недоступними для сегмента малого та середнього бізнесу, вимагаючи значних інвестицій у впровадження та підтримку. З іншого боку, класичні методи ручного планування, на які досі покладаються невеликі перевізники, не забезпечують необхідної ефективності в умовах високої динаміки міського трафіку та зростаючих вимог клієнтів до швидкості доставки. Враховуючи виявлений дисбаланс, метою роботи визначено підвищення ефективності транспортних перевезень шляхом розробки алгоритмічного забезпечення та

програмної реалізації спеціалізованого веб-сервісу для автоматизованої побудови та оптимізації маршрутів доставки. Об'єктом дослідження виступає процес управління вантажоперевезеннями в логістичних системах, а предметом дослідження є математичні моделі, методи кластеризації та метаевристичні алгоритми вирішення задачі маршрутизації транспортних засобів.

З наукової точки зору, задача дослідження не обмежується простою автоматизацією рутинних дій диспетчера, а полягає у знаходженні оптимальної конфігурації маршрутів у межах формалізованої транспортної мережі. Ключовим критерієм оптимізації виступає мінімізація цільової функції, що виражає сумарну вартість перевезень, яка може інтерпретуватися як загальна дистанція пробігу або сумарний час перебування в дорозі, за умови суворого дотримання системи жорстких обмежень. До таких критичних обмежень відносяться: неперевищення номінальної вантажопідйомності транспортних засобів, забезпечення одноразового відвідування кожної точки доставки без дублювання та обов'язкова умова замкненості маршрутів, що передбачає повернення автомобіля до вихідного складу після завершення рейсу [16].

Для досягнення поставленої мети передбачається послідовне вирішення комплексу взаємопов'язаних науково-практичних завдань, що охоплюють усі етапи проєктування системи. На початковому етапі необхідно провести порівняльний аналіз ефективності алгоритмів пошуку найкоротшого шляху на графах, зокрема, класичного алгоритму Дейкстри та модифікованого алгоритму A^* з використанням евристики Гаверсина. Це потрібно для обґрунтування вибору методу швидкої побудови матриці відстаней, яка є фундаментом для подальших розрахунків. Також важливим завданням є дослідження впливу використання реальних топологічних геоданих з сервісу OpenRouteService на точність моделювання, що дозволить наблизити розрахункову модель до умов реального міського середовища.

Ключовим етапом роботи є теоретичне обґрунтування та практична розробка комбінованого методу маршрутизації, який базуватиметься на стратегії декомпозиції складної задачі. Цей метод поєднуватиме стратегію попередньої кластеризації точок доставки із застосуванням модифікованого методу Fixed K-Means для їх жорсткої прив'язки до складів та подальшу евристичну оптимізацію маршрутів всередині кожного сформованого кластера [17]. Для вибору найбільш ефективного інструменту локальної оптимізації необхідно виконати експериментальне порівняння сучасних метаевристичних підходів – генетичного алгоритму, алгоритму мурашиної колонії та методу табу-пошуку. Порівняння проводитиметься за комплексними критеріями швидкодії, точності знайденого рішення та стабільності результатів при масштабуванні вхідних даних, що дозволить обрати алгоритм, найбільш придатний для роботи в режимі реального часу.

На основі обраних алгоритмічних рішень та результатів моделювання необхідно спроектувати архітектуру програмного комплексу, що відповідатиме сучасним вимогам до веб-застосунків. Вона включатиме серверну частину для обробки бізнес-логіки та зберігання даних, окремий модуль математичних розрахунків та клієнтський веб-інтерфейс для інтерактивної взаємодії з користувачем і візуалізації побудованих маршрутів на електронній карті.

Висновки до розділу 1

У першому розділі виконано аналіз теоретичних засад транспортної логістики та встановлено, що ефективне вирішення проблеми «останньої милі» вимагає переходу від інтуїтивного планування до використання автоматизованих систем на базі алгоритмів маршрутизації транспортних засобів. Оскільки ця задача належить до класу NP-важких, обґрунтовано неефективність точних методів для реальних бізнес-сценаріїв та доведено доцільність застосування гібридних підходів, що поєднують кластеризацію з

метаевристичними алгоритмами (генетичні, мурашині, табу-пошук). Такий підхід дозволяє знаходити субоптимальні рішення з високою точністю та прийнятною швидкістю обчислень.

Аналіз ринку програмного забезпечення виявив дефіцит доступних рішень для малого та середнього бізнесу, що обумовило необхідність розробки власного адаптивного веб-сервісу. На основі цього сформульовано постановку задачі дослідження, що передбачає створення алгоритмічного забезпечення та програмну реалізацію інформаційної системи, здатної враховувати вантажопідйомність транспорту та геолокацію. Реалізація поставлених завдань дозволить автоматизувати процес формування маршрутів, мінімізувати експлуатаційні витрати автопарку та підвищити якість логістичного обслуговування.

РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА РОЗРОБКА КОМПЛЕКСНОГО АЛГОРИТМУ МАРШРУТИЗАЦІЇ

2.1. Концептуальна модель системи та декомпозиція задачі

Основою розроблюваної системи є інформаційна модель, що відображає структуру транспортної мережі та логіку перевезення вантажів. У рамках даного дослідження предметна область розглядається як сукупність географічних об'єктів, об'єднаних дорожніми зв'язками [9, 10].

Ключовими елементами моделі є два типи вузлів: склади, які виступають початковими та кінцевими точками маршрутів, та точки доставки, що формують потребу у перевезеннях. Кожен клієнт характеризується певною величиною попиту – вагою товару, яку необхідно доставити. Важливим параметром моделі є вартість переміщення між будь-якими двома об'єктами мережі. У даній роботі ця вартість визначається не як проста геометрична відстань по прямій, а як реальна довжина шляху або час проїзду дорогами загального користування, що отримується з використанням геоінформаційних сервісів та спеціалізованих алгоритмів пошуку шляху [12].

Постановка задачі полягає у побудові такого набору маршрутів, який забезпечує обслуговування всіх замовлень із мінімальними сумарними витратами (найменшим загальним пробігом або часом). При цьому процес формування маршрутів суворо регламентується низкою логістичних обмежень, які визначають допустимість розв'язку [1, 18]:

1. Обмеження ресурсів автопарку – для кожного транспортного засобу визначено параметр максимальної вантажопідйомності. Сумарна вага всіх вантажів, запланованих для перевезення одним рейсом, не повинна перевищувати цей ліміт. Це є критичною умовою фізичної здійсненності маршруту.
2. Умова одноразового обслуговування – кожна точка доставки має бути включена лише в один маршрут і відвідана транспортним засобом рівно один раз. Це запобігає дублюванню замовлень та забезпечує повноту виконання плану.

3. Умова замкненості маршруту – логістика підприємства передбачає, що кожен транспортний засіб розпочинає рух зі закріпленого за ним складу і, після об'їзду всіх клієнтів, обов'язково повертається назад у те ж саме депо.

Описана задача належить до класу NP-важких задач комбінаторної оптимізації. Це означає, що зі збільшенням кількості клієнтів кількість можливих варіантів маршрутів зростає експоненціально, що унеможливорює перебір усіх комбінацій за прийнятний час та вимагає застосування евристичних методів [13].

Основна проблема масштабування полягає у факторіальному зростанні простору пошуку рішень. При лінійному збільшенні кількості точок доставки кількість можливих варіантів побудови маршрутів зростає експоненціально. Якщо для малого набору вхідних даних застосування точних математичних методів є технічно можливим, то при кількості вузлів, що перевищує 50, час, необхідний для перебору варіантів та знаходження глобального оптимуму, стає неприпустимо великим для інтерактивних систем. Така специфіка задачі робить неможливим використання точних алгоритмів у режимі реального часу та обґрунтовує необхідність застосування наближених евристичних методів та стратегій декомпозиції, які дозволяють отримати субоптимальне, але прийнятне для бізнесу рішення за обмежений проміжок часу [19].

Для подолання обчислювальної складності та забезпечення ефективної роботи системи з великими масивами даних доцільним є використання стратегії декомпозиції, відомої як «Cluster-First, Route-Second» (спочатку кластеризація, потім маршрутизація). Ця концепція виступає основним методом вирішення проблеми масштабування, дозволяючи трансформувати складну комбінаторну задачу NP-рівня в структуру, придатну для оперативної обробки [20].

Сутність даного підходу полягає у розбитті глобальної задачі оптимізації на набір незалежних локальних підзадач меншої розмірності. Процес реалізується шляхом попереднього групування точок доставки за

територіальною ознакою або прив'язкою до логістичних центрів, після чого для кожного сформованого кластера окремо вирішується задача побудови оптимального маршруту [21]. Така декомпозиція дозволяє суттєво зменшити простір пошуку рішень, забезпечуючи значний приріст швидкодії алгоритму без критичної втрати якості кінцевого результату.

Структурна схема запропонованого підходу базується на послідовному виконанні трьох взаємопов'язаних етапів, що забезпечують логічну трансформацію вхідних даних у готовий план перевезень. На початковій стадії відбувається геометричне групування замовлень навколо наявних складів, що реалізується методами кластеризації. Цей процес дозволяє локалізувати задачу та прив'язати клієнтів до конкретних центрів обслуговування. Другий етап передбачає розрахунок реальних дорожніх відстаней між об'єктами в межах сформованих груп, результатом якого є побудова матриці вартостей. Завершальним кроком є безпосередній евристичний пошук маршрутів усередині кожного кластера, що забезпечує оптимізацію шляху для конкретних транспортних засобів з урахуванням усіх обмежень [22].

Застосування такої гібридної моделі дозволяє досягти необхідного балансу між точністю математичних розрахунків та швидкістю обробки запитів. Цей аспект є критично важливим для функціонування веб-сервісу, який повинен забезпечувати інтерактивну взаємодію з користувачем та надавати результати планування у прийнятний час без надмірних затримок, властивих точним методам оптимізації [23].

2.2. Дослідження методів кластеризації та евристичної оптимізації

Для практичної верифікації запропонованої моделі та обґрунтування вибору конкретних алгоритмічних рішень було проведено серію експериментальних досліджень. Основною метою цього етапу є визначення найбільш ефективної комбінації методів для реалізації кожного кроку декомпозиції задачі: геометричного розбиття множини замовлень, розрахунку вартості переміщення та фінальної маршрутизації. Критеріями вибору визначено точність отриманих результатів, тобто мінімізація сумарної вартості маршруту, швидкодію алгоритмів та їхню здатність до масштабування при збільшенні навантаження [1, 20].

Експерименти проводилися на синтетичних наборах даних, що імітують реальну логістичну мережу в межах міста Києва. Генерація координат точок доставки здійснювалася у визначеному діапазоні широт (50.35–50.52) та довгот (30.35–30.65), що охоплює центральні та прилеглі райони столиці. Кожна згенерована точка характеризувалася географічною прив'язкою до вузлів дорожнього графа OpenStreetMap та випадковим значенням попиту в діапазоні від 1 до 5 одиниць вантажу. Для оцінки поведінки алгоритмів в умовах різного навантаження було сформовано тестові вибірки різної розмірності, що включали $N=10, 50, 100$ та 150 точок доставки, що узгоджується з діапазонами розмірностей, які розглядаються в сучасних дослідженнях [24].

Моделювання виконувалося у хмарному обчислювальному середовищі Google Colab з використанням мови програмування Python та спеціалізованих бібліотек для роботи з геоданими. Для забезпечення достовірності розрахунків відстаней та часу проїзду використовувалися дані сервісу OpenRouteService, який дозволяє враховувати реальні обмеження дорожнього руху. Такий підхід забезпечив баланс між керованістю параметрів експерименту та наближеністю умов тестування до реального середовища функціонування логістичних систем, де точність геопросторових даних є критичною [12, 25].

На першому етапі експериментального дослідження вирішувалося ключове завдання формування зон обслуговування. Для цього було програмно реалізовано та порівняно три підходи до кластеризації точок доставки: класичний алгоритм К-Means, модифікацію з фіксованими центрами та метод збалансованої кластеризації, доцільність використання яких для задач маршрутизації підтверджується кількома дослідженнями [15, 21].

Як базовий метод використано класичний алгоритм К-Means, який ітеративно визначає центри кластерів шляхом мінімізації сумарної евклідової відстані від точок доставки до центроїдів. Його програмна реалізація, що базується на стандартних бібліотеках машинного навчання, дозволяє швидко обробляти великі масиви даних, проте не враховує специфічних логістичних обмежень [26].

```
def kmeans(points, k, max_iters=100, tol=1e-4):
    points = np.array(points)
    n_points = points.shape[0]
    centroids = points[np.random.choice(n_points, k, replace=False)]

    for _ in range(max_iters):
        distances = np.linalg.norm(points[:, None] - centroids[None, :], axis=2)
        labels = np.argmin(distances, axis=1)
        new_centroids = np.array([
            points[labels == i].mean(axis=0) if np.any(labels == i) else
            centroids[i]
            for i in range(k)
        ])
        if np.linalg.norm(new_centroids - centroids) < tol:
            break
        centroids = new_centroids
    return labels, centroids
```

Результати тестування засвідчили, що класичний алгоритм К-Means демонструє найвищу швидкодію з показником часу виконання 0,0094 с. Однак, як видно з візуалізації результатів (рис. 2.1), суттєвим недоліком цього методу є відсутність жорсткої прив'язки центроїдів до реальних складів. Це призвело до диспропорційного розподілу: один кластер виявився перевантаженим (29 точок), тоді як інші залишалися мінімально задіяними. Така поведінка є типовою для стандартних алгоритмів кластеризації без обмежень і потребує впровадження модифікацій для практичного застосування у логістиці [27].

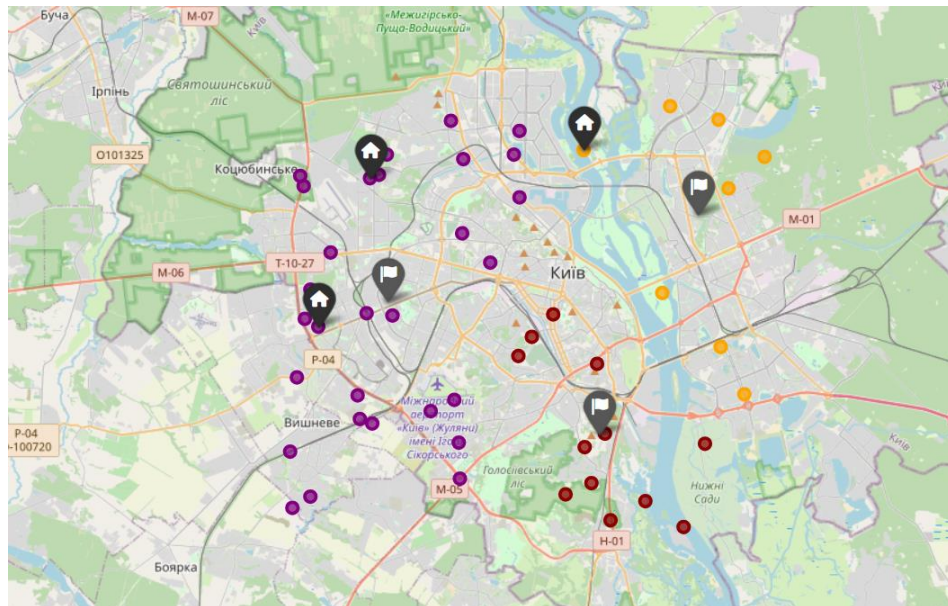


Рис.2.1. Кластеризація за допомогою KMeans.

Для адаптації моделі до реальних інфраструктурних вимог було реалізовано метод «Fixed K-Means». Його особливість полягає у фіксації початкових центроїдів виключно в координатах існуючих складів, що забезпечує прив'язку кожної точки замовлення до найближчого логістичного вузла.

```
def fixed_kmeans(points, warehouses, max_iters=10):
    warehouses = [np.array(w) for w in warehouses]
    clusters = {tuple(wh): [] for wh in warehouses}
    for _ in range(max_iters):
        clusters = {tuple(wh): [] for wh in warehouses}
        for point in points:
            distances = [np.linalg.norm(point - wh) for wh in warehouses]
            idx = np.argmin(distances)
            clusters[tuple(warehouses[idx])].append(point)
    return clusters
```

Аналіз результатів показав, що хоча метод Fixed K-Means потребував дещо більшого часу обчислень (0,0118 с), він забезпечив формування значно логічніших зон обслуговування (рис. 2.2). Кластери групуються безпосередньо навколо фіксованих координат складів, що повністю відповідає практичним вимогам транспортної логістики, як це було зазначено в результатах попередніх досліджень [1].

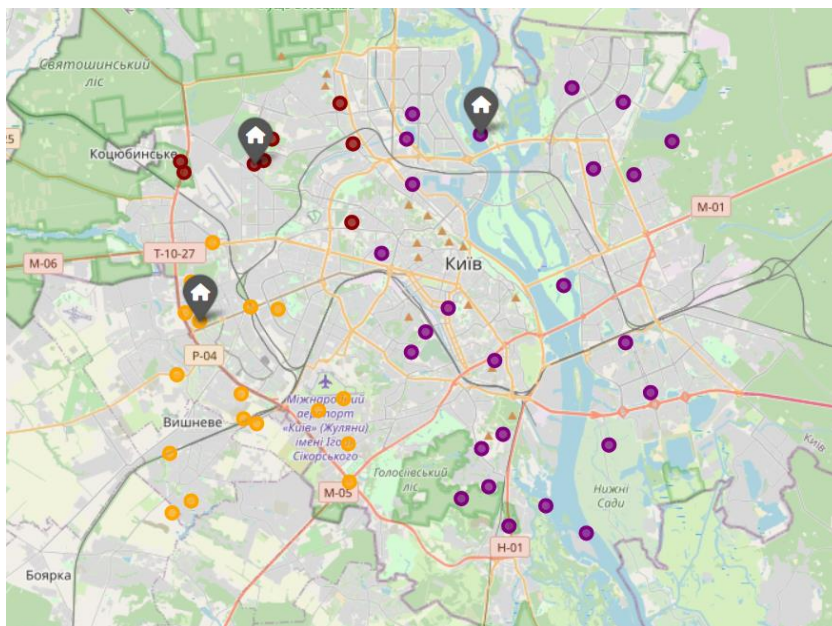


Рис.2.2. Кластеризація за допомогою Fixed KMeans.

Третій реалізований підхід – збалансована кластеризація (Balanced Clustering) – додатково вводить обмеження на максимальну кількість точок у кластері. Це дозволяє примусово перерозподіляти замовлення між складами для уникнення ситуацій, коли один вузол перевантажений, а інші простоюють. Цей метод базується на модифікаціях алгоритму K-Means для задач з обмеженнями ємності [28].

```
def balanced_clustering(points, warehouses, max_points=13):
    clusters = {tuple(wh): [] for wh in warehouses}
    closest_idx = pairwise_distances_argmin_min(points, warehouses)[0]
    for i, wh_idx in enumerate(closest_idx):
        wh = tuple(warehouses[wh_idx])
        if len(clusters[wh]) < max_points:
            clusters[wh].append(points[i])
        else:
            alt_wh = min(clusters, key=lambda k: len(clusters[k]))
            clusters[alt_wh].append(points[i])
    return clusters
```

Цей метод гарантував найбільш рівномірне навантаження, розподіляючи по 16–17 замовлень на кожен склад (рис. 2.3). Однак такий баланс досягався ціною зниження географічної ефективності: алгоритм примусово прикріплював точки до віддалених терміналів, ігноруючи їхню близькість до іншого складу. Крім того, цей підхід супроводжувався найвищими витратами часу – 0,0193 с [1]. Це підтверджує відомий у теорії оптимізації факт, що введення жорстких обмежень на рівномірність кластерів

часто призводить до збільшення сумарної внутрішньокластерної відстані та погіршення компактності зон [29].

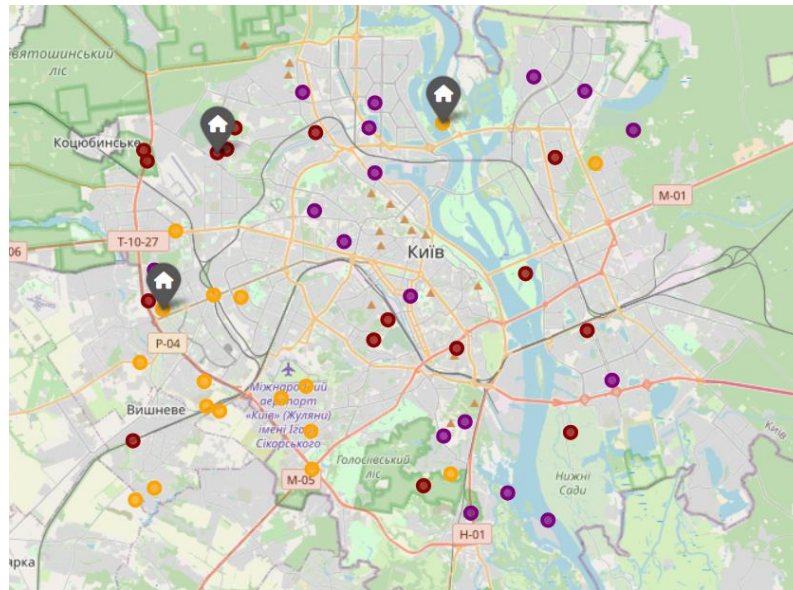


Рис.2.3. Кластеризація за допомогою Balanced Clustering.

Враховуючи, що пріоритетом системи є коректна логістика, саме метод Fixed K-Means було обрано як базовий, оскільки він пропонує необхідний компроміс: забезпечує реалістичну прив'язку до депо, якої не має звичайний K-Means, та працює достатньо швидко для задач реального часу.

На наступному етапі дослідження було реалізовано та порівняно два класичні методи пошуку найкоротших маршрутів у дорожньому графі – алгоритм Дейкстри та алгоритм A*. Обидва підходи застосовувалися для генерації матриці відстаней на основі вуличної мережі Києва, імпортованої через бібліотеку OSMnx, що дозволяє працювати з реальними топологічними даними міста [25].

Алгоритм Дейкстри реалізовано як класичний жадібний метод для знаходження найкоротшого шляху у зваженому графі. Механізм його роботи базується на використанні пріоритетної черги, яка забезпечує послідовний вибір вершин із мінімальною поточною відстанню від стартової точки [10].

```
def dijkstra_osmnx(G, start, goal):  
    dist = {node: float('inf') for node in G.nodes}  
    dist[start] = 0  
    visited = set()  
    queue = [(0, start)]  
  
    while queue:
```

```

current_dist, current_node = heapq.heappop(queue)
if current_node == goal:
    return dist[goal]
if current_node in visited:
    continue
visited.add(current_node)

for neighbor in G.neighbors(current_node):
    edge_weight = G[current_node][neighbor].get('length', 1)
    alt = current_dist + edge_weight
    if alt < dist[neighbor]:
        dist[neighbor] = alt
        heapq.heappush(queue, (alt, neighbor))
return float('inf')

```

Головною перевагою методу є його універсальність та гарантія оптимальності результату, оскільки він не залежить від просторової структури даних. Проте, через відсутність інформації про напрямки цілі, алгоритм змушений перевіряти значну кількість вузлів, що суттєво знижує його швидкодію на масштабних графах вуличних мереж [11].

Більш досконалим варіантом є алгоритм A*, імплементований як оптимізована версія пошуку, що використовує евристичну функцію для спрямування обчислювального процесу. В якості евристики застосовано формулу Haversine, яка розраховує географічну відстань «по прямій» між поточним вузлом та цільовою точкою.

```

def haversine(coord1, coord2):
    R = 6371.0 # km
    lat1, lon1 = radians(coord1[0]), radians(coord1[1])
    lat2, lon2 = radians(coord2[0]), radians(coord2[1])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = sin(dlat/2)**2 + cos(lat1)*cos(lat2)*sin(dlon/2)**2
    return R * 2 * atan2(sqrt(a), sqrt(1 - a)) * 1000

def a_star_osmnx(G, start, goal):
    def heuristic(n1, n2):
        coord1 = (G.nodes[n1]['y'], G.nodes[n1]['x'])
        coord2 = (G.nodes[n2]['y'], G.nodes[n2]['x'])
        return haversine(coord1, coord2)

    g = {node: float('inf') for node in G.nodes}
    f = {node: float('inf') for node in G.nodes}
    g[start] = 0
    f[start] = heuristic(start, goal)

    open_set = [(f[start], start)]
    visited = set()

    while open_set:
        _, current = heapq.heappop(open_set)

```

```

    if current == goal:
        return g[goal]
    if current in visited:
        continue
    visited.add(current)

    for neighbor in G.neighbors(current):
        edge_weight = G[current][neighbor].get('length', 1)
        tentative_g = g[current] + edge_weight
        if tentative_g < g[neighbor]:
            g[neighbor] = tentative_g
            f[neighbor] = g[neighbor] + heuristic(neighbor, goal)
            heapq.heappush(open_set, (f[neighbor], neighbor))

return float('inf')

```

Ця функція дозволяє алгоритму пріоритизувати вершини, що знаходяться ближче до фінішу, та відсікати неперспективні напрямки. Завдяки такому підходу A* зберігає точність алгоритму Дейкстри (за умови коректності евристики), але демонструє значно вищу швидкість виконання завдяки зменшенню простору пошуку [11].

У ході експерименту було згенеровано матриці відстаней для тестового набору з 15 точок. Зіставлення результатів засвідчило повну ідентичність довжин шляхів, знайдених обома методами, що підтверджує коректність програмної реалізації та адекватність обраної евристики для алгоритму A*. Водночас аналіз часових витрат виявив суттєву перевагу евристичного підходу: алгоритм A* виконав обчислення за 8,847 с, тоді як метод Дейкстри потребував 21,415 с. Це демонструє більш ніж двократне підвищення продуктивності системи при збереженні точності розрахунків, що корелює з результатами аналогічних досліджень ефективності пошукових алгоритмів [1, 11].

Для оцінки практичної придатності розробленого рішення було проведено додаткове порівняння матриці відстаней, отриманої алгоритмом A*, з даними сервісу OpenRouteService (ORS). Метою цього етапу було визначення відхилень локальної моделі від результатів роботи спеціалізованого онлайн-інструменту, який базується на екосистемі OpenStreetMap та вважається еталоном у задачах маршрутизації [25].

Аналіз отриманих числових даних виявив закономірність, за якої дистанції, розраховані сервісом ORS (рис. 2.4), стабільно перевищують результати локального алгоритму A* (рис. 2.5).

distance_matrix_ors.csv

distance_matrix_astar.csv

1 to 15 of 15 entries

Filter

	Start	WP1	WP2	WP3	WP4	WP5	WP6	WP7	WP8	WP9	WP10	WP11	WP12	WP13	WP14
Start	0.0	25272.72	11429.92	11683.29	19781.81	15432.83	24080.83	12965.8	17302.68	22167.81	5942.46	25993.17	12077.8	21489.6	26084.72
WP1	25281.22	0.0	21457.26	21425.81	16569.46	27598.15	10497.72	21013.25	10853.65	13923.21	22006.58	8167.64	13775.59	18810.71	15167.07
WP2	10936.43	21499.62	0.0	12782.46	11982.17	21407.84	18473.54	14587.96	12374.44	14368.17	10983.61	22220.07	9935.43	23808.01	23606.2
WP3	10371.07	20014.73	13595.92	0.0	21947.81	11335.28	19082.27	3827.07	14233.45	24333.81	7016.32	20735.18	9582.08	14168.34	20680.6
WP4	17093.4	16337.79	8987.96	18939.43	0.0	27564.81	21561.0	20744.93	15461.9	8213.7	17140.58	22176.76	13022.89	26801.93	26693.66
WP5	17158.12	28178.92	22421.22	11524.26	30773.11	0.0	26533.58	8149.18	23058.74	33159.11	15200.98	28899.37	18407.38	15328.17	21840.42
WP6	23101.73	8785.83	18162.41	20244.0	19903.17	27719.47	0.0	21134.57	7558.81	17256.92	19827.09	9506.28	11596.09	19109.53	11959.9
WP7	14244.7	23425.44	17204.41	3608.49	25556.3	8644.45	21780.1	0.0	17841.93	27942.3	10624.8	24145.89	13190.57	10574.68	17086.94
WP8	16596.21	11026.78	12413.42	13738.48	15655.18	22842.44	8000.7	16257.54	0.0	13866.94	13321.57	11747.23	5090.58	14505.02	13133.36
WP9	22224.16	13882.07	14118.72	24070.19	8256.17	32695.57	18152.37	27229.5	13653.74	0.0	22271.34	19721.03	16768.97	25476.99	23285.04
WP10	7499.3	20495.19	11194.14	6732.22	19546.03	14378.35	17869.21	9082.09	12097.08	21932.03	0.0	21215.64	8285.28	18336.05	21307.2
WP11	27975.05	6405.6	24518.44	24119.64	21558.76	30291.98	13191.56	23707.09	13914.83	18912.51	24700.41	0.0	16469.42	21504.54	17860.9
WP12	13006.15	14142.55	9604.37	10148.42	14318.43	18523.33	11116.46	11953.92	5405.01	15746.54	9731.51	14863.0	0.0	15524.05	16249.13
WP13	22343.63	18624.95	23361.58	12581.88	27708.59	15791.7	16555.59	9206.8	14689.44	25062.34	19167.1	19345.4	14942.27	0.0	10400.05
WP14	23219.33	11633.01	21009.59	20034.99	22750.35	23244.81	7267.13	16659.91	10405.99	20104.1	20128.5	12353.46	12993.12	14550.74	0.0

distance_matrix_ors.csv

distance_matrix_astar.csv

1 to 15 of 15 entries

Filter

	Start	WP1	WP2	WP3	WP4	WP5	WP6	WP7	WP8	WP9	WP10	WP11	WP12	WP13	WP14
Start	0.0	24084.0	10262.0	9808.0	17247.0	10998.0	20500.0	12369.0	15197.0	21610.0	6413.0	24745.0	11128.0	19309.0	22689.0
WP1	24084.0	0.0	20693.0	19418.0	15989.0	20636.0	8115.0	19028.0	10003.0	12672.0	19426.0	6211.0	13338.0	16269.0	11688.0
WP2	10262.0	20693.0	0.0	12289.0	8548.0	15181.0	16935.0	14199.0	11415.0	12924.0	8894.0	21969.0	8672.0	20246.0	20648.0
WP3	9808.0	19418.0	12289.0	0.0	18614.0	4512.0	15833.0	3463.0	12186.0	22907.0	3395.0	20079.0	8925.0	10506.0	17495.0
WP4	17247.0	15989.0	8548.0	18614.0	0.0	21505.0	18492.0	20523.0	14449.0	7526.0	15219.0	20948.0	12157.0	24233.0	22205.0
WP5	10998.0	20636.0	15181.0	4512.0	21505.0	0.0	17052.0	2720.0	15074.0	25795.0	6588.0	21297.0	11812.0	8697.0	16995.0
WP6	20500.0	8115.0	16935.0	15833.0	18492.0	17052.0	0.0	15444.0	6575.0	15568.0	15842.0	8776.0	9753.0	12685.0	6802.0
WP7	12369.0	19028.0	14199.0	3463.0	20523.0	2720.0	15444.0	0.0	14342.0	24877.0	6858.0	19689.0	11081.0	7180.0	15399.0
WP8	15197.0	10003.0	11415.0	12186.0	14449.0	15074.0	6575.0	14342.0	0.0	12267.0	10927.0	11384.0	4308.0	12162.0	10102.0
WP9	21610.0	12672.0	12924.0	22907.0	7526.0	25795.0	15568.0	24877.0	12267.0	0.0	19573.0	17674.0	14353.0	22366.0	19281.0
WP10	6413.0	19426.0	8894.0	3395.0	15219.0	6588.0	15842.0	6858.0	10927.0	19573.0	0.0	20087.0	7630.0	13901.0	17504.0
WP11	24745.0	6211.0	21969.0	20079.0	20948.0	21297.0	8776.0	19689.0	11384.0	17674.0	20087.0	0.0	13999.0	16930.0	12349.0
WP12	11128.0	13338.0	8672.0	8925.0	12157.0	11812.0	9753.0	11081.0	4308.0	14353.0	7630.0	13999.0	0.0	12701.0	12560.0
WP13	19309.0	16269.0	20246.0	10506.0	24233.0	8697.0	12685.0	7180.0	12162.0	22366.0	13901.0	16930.0	12701.0	0.0	11184.0
WP14	22689.0	11688.0	20648.0	17495.0	22205.0	16995.0	6802.0	15399.0	10102.0	19281.0	17504.0	12349.0	12560.0	11184.0	0.0

Рис.2.4 – 2.5. Матриця, сформована за допомогою ORS та A*.

Така розбіжність є закономірною та пояснюється деталізацією моделі ORS, яка враховує реальні правила дорожнього руху, заборони поворотів та специфіку проїзду складних перехресть. Натомість реалізація A* функціонує в межах абстрактної топології дорожньої мережі, що дозволяє знаходити математично оптимальний шлях, але часто призводить до недооцінки його фактичної протяжності в умовах реального трафіку [10].

Попри виявлені кількісні відмінності, обидві матриці демонструють високу структурну кореляцію у розподілі відстаней між контрольними точками. Це свідчить про те, що розроблений алгоритм A* забезпечує достатню точність для задач попереднього планування. Хоча дані ORS є точнішими, їх використання у фінальній версії системи обмежене лімітами на

кількість API-запитів. Тому, як компромісне рішення, для обчислювального ядра системи обрано алгоритм A^* , який забезпечує необхідну автономність, високу швидкість (в 2 рази швидше за Дейкстру) та достатню для цілей планування точність [1].

На завершальному етапі експериментального дослідження було проведено порівняльний аналіз трьох метаевристичних методів маршрутизації: генетичного алгоритму (GA), алгоритму мурашиної колонії (ACO) та методу табу-пошуку (Tabu Search). Для забезпечення чистоти експерименту всі алгоритми тестувалися на ідентичних вхідних даних із залученням трьох транспортних засобів вантажопідйомністю 20 одиниць. Результати моделювання представлені на рисунку 2.6.

```
Алгоритм: Генетичний
Найкраща вартість: 224517.01
Маршрут 1: Start → WP2 → WP9 → WP1 → WP11 → WP13 → WP14 → WP6 → Start | Попит: 20.0
Маршрут 2: Start → WP3 → WP5 → WP8 → WP4 → WP12 → WP7 → Start | Попит: 16.0
Маршрут 3: Start → WP10 → Start | Попит: 5.0
Час виконання: 0.10 сек

Алгоритм: Мурашиний
Найкраща вартість: 177282.10
Маршрут 1: Start → WP10 → WP3 → WP7 → WP5 → WP13 → WP14 → WP6 → WP8 → Start | Попит: 20.0
Маршрут 2: Start → WP12 → WP11 → WP1 → WP9 → WP4 → Start | Попит: 16.0
Маршрут 3: Start → WP2 → Start | Попит: 5.0
Час виконання: 1.41 сек

Алгоритм: Табу пошук
Найкраща вартість: 215543.84
Маршрут 1: Start → WP2 → WP8 → WP6 → WP11 → WP1 → WP14 → WP13 → WP5 → Start | Попит: 19.0
Маршрут 2: Start → WP9 → WP4 → WP12 → WP7 → WP3 → Start | Попит: 17.0
Маршрут 3: Start → WP10 → Start | Попит: 5.0
Час виконання: 0.12 сек
```

Рис.2.6. Результати використання метаевристичних алгоритмів.

Генетичний алгоритм продемонстрував найвищу швидкість, виконавши розрахунок усього за 0,10 с. Отриманий розв'язок складається з трьох маршрутів із загальною вартістю 224 517,01 одиниць. Попри високу швидкість обробки даних, цей метод показав найнижчу якість оптимізації серед порівнюваних підходів (рис. 2.7–2.8), що свідчить про недостатню глибину пошуку при обраних параметрах мутації та кросинговеру і схильність до передчасної збіжності, що є відомим недоліком класичних реалізацій GA [22].

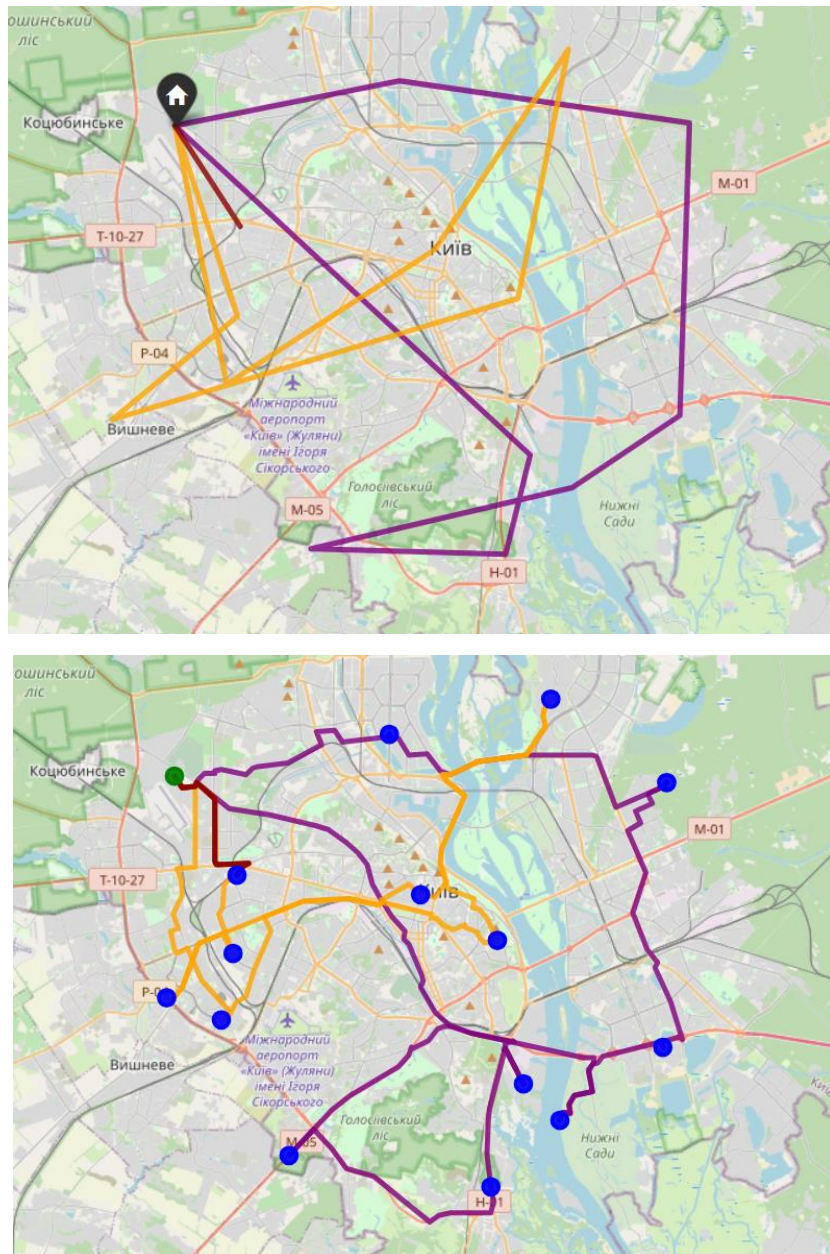


Рис.2.7 – 2.8. Маршрут, побудований генетичним алгоритмом.

Алгоритм мурашиної колонії забезпечив найкращий показник ефективності, досягнувши мінімальної сумарної вартості маршруту – 177 282,10 одиниць. Висока точність розв’язку обумовлена вдалим балансуванням навантаження та використанням феромонних моделей, які дозволяють агентам знаходити глобально оптимальні шляхи. Проте висока якість рішення була досягнута ціною суттєвого збільшення часу виконання до 1,41 с. Таким чином, метод виявився найточнішим, але й найбільш ресурсомістким, що робить його менш придатним для систем, які вимагають миттєвої реакції (рис. 2.9–2.10) [1].

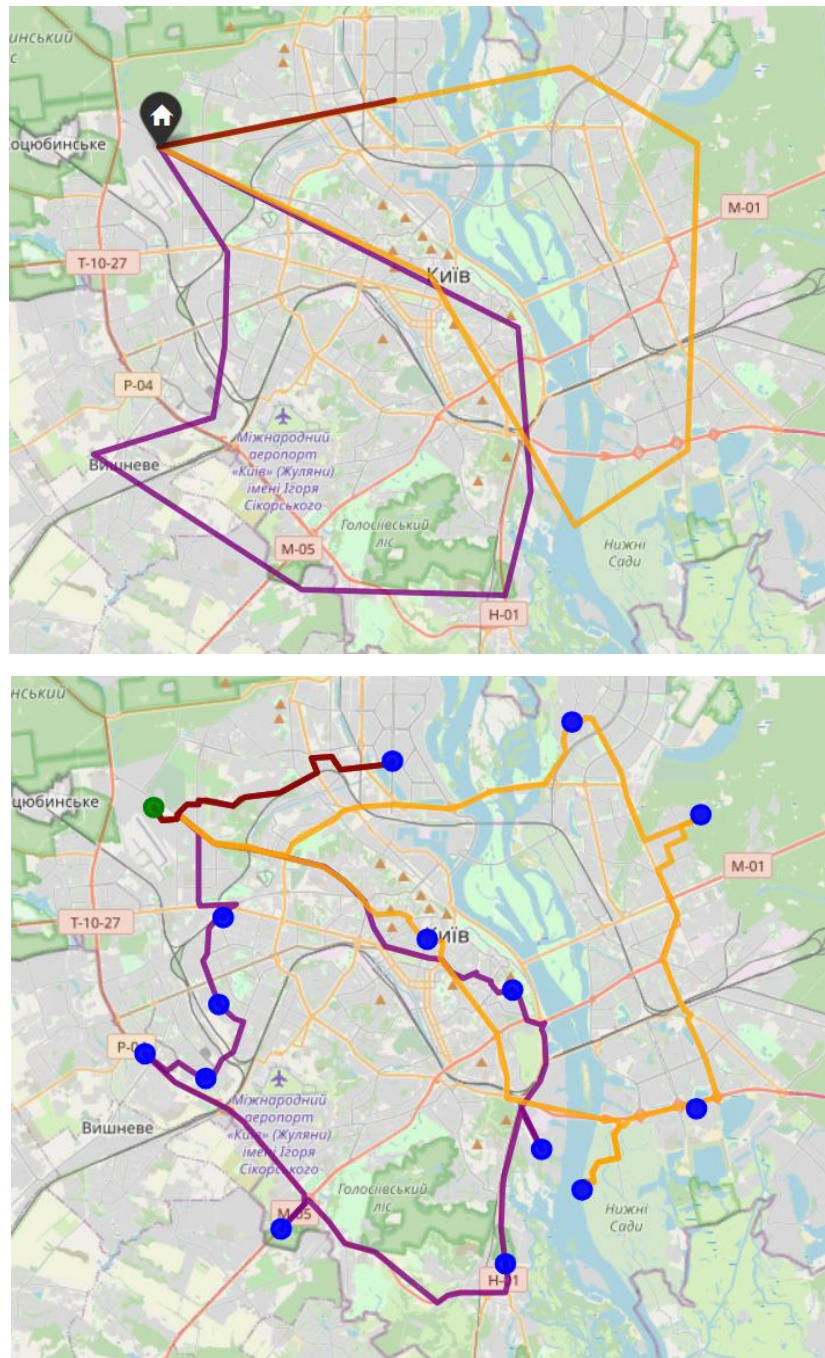


Рис.2.9 – 2.10. Маршрут, побудований мурашиним алгоритмом.

Метод табу-пошуку зайняв проміжну позицію, продемонструвавши оптимальний баланс між швидкістю та якістю. При часі виконання 0,12 с (що лише на 0,02 с повільніше за генетичний алгоритм) вартість маршруту склала 215 543,84 одиниць. Завдяки механізму короткострокової пам'яті («табу-листів») алгоритм успішно уникнув пасток локальних мінімумів і перевершив результати генетичного підходу, хоча й дещо поступився точності мурашиного алгоритму (рис. 2.11–2.12) [1]. Це узгоджується з даними інших

порівняльних досліджень, де Tabu Search часто визначається як найбільш стабільний метод для задач VRP з часовими обмеженнями [30].

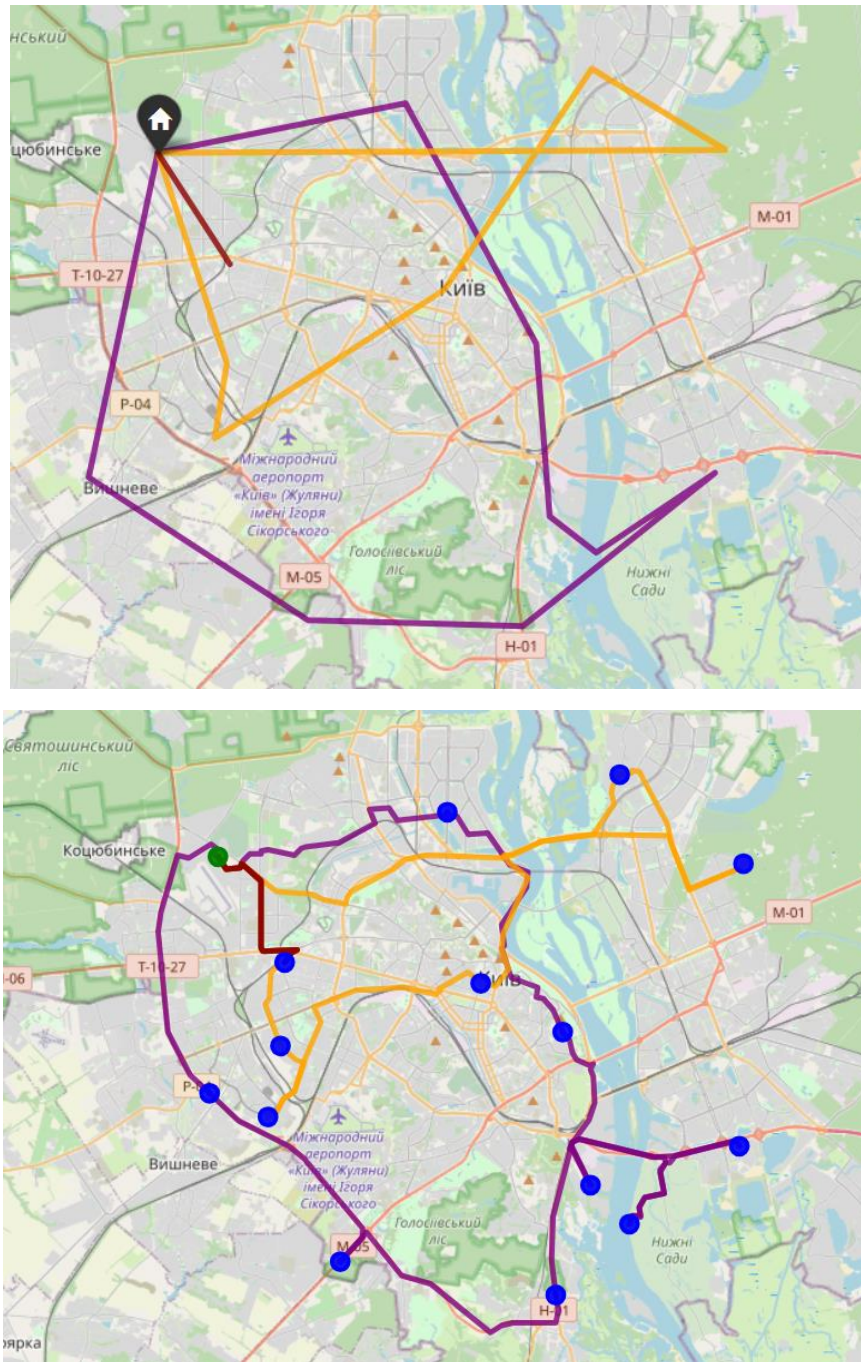


Рис.2.11 – 2.12. Маршрут, побудований табу-пошуком.

Підсумовуючи результати порівняльного аналізу, встановлено, що за критерієм якості оптимізації беззаперечним лідером є алгоритм мурашиної колонії. Однак його використання пов'язане зі значними часовими витратами, які зростають при збільшенні розмірності задачі. У розрізі швидкодії лідером є генетичний алгоритм, але він не забезпечує достатньої якості маршрутизації [1].

Метод табу-пошуку продемонстрував найкраще співвідношення «ціна/якість»: він працює в 10 разів швидше за мурашиний алгоритм, забезпечуючи при цьому прийнятну точність рішень. Враховуючи вимоги до веб-сервісу щодо роботи в режимі реального часу, для програмної реалізації системи було обрано саме метод табу-пошуку як базовий алгоритм оптимізації всередині кластерів.

На основі отриманих даних, було сформовано фінальну конфігурацію алгоритмічного ядра системи. Вибір методу Fixed K-Means забезпечує коректну прив'язку замовлень до складської інфраструктури, алгоритм A* гарантує високу швидкість розрахунку дистанцій без залежності від зовнішніх сервісів, а метод табу-пошуку надає найякісніші маршрути в умовах обмеженого часу. Така комбінація методів є обґрунтованою основою для розробки комплексного алгоритму роботи веб-сервісу, структура та реалізація якого будуть детально розглянуті в наступному підрозділі.

2.3. Побудова узагальненого алгоритму оптимізації маршрутів

На основі результатів порівняльного аналізу та обраних технічних рішень було спроектовано узагальнений алгоритм роботи системи, який реалізує архітектурний патерн «обчислювальний конвеєр» (Pipeline). Такий підхід забезпечує послідовну та контрольовану трансформацію вхідних даних про замовлення у фінальний оптимізований план маршрутів. Процес виконання розпочинається з отримання та валідації вхідного масиву даних у форматі JSON, який містить структуровану інформацію про геолокацію складів, координати точок доставки, вагові параметри вантажів та характеристики наявного автопарку.

На етапі ініціалізації відбувається створення допоміжних компонентів, необхідних для роботи алгоритму. Ключовими серед них є модуль DistanceProvider, який відповідає за швидкий розрахунок відстаней, та TrafficMonitor, що дозволяє моделювати дорожнє навантаження. Після

підготовки середовища виконується геометрична декомпозиція глобальної задачі за допомогою методу Fixed K-Means. Цей модуль розподіляє множину замовлень на незалежні групи, жорстко прив'язані до фізичних координат складів. Одночасно з кластеризацією відбувається процедура розподілу транспорту, яка закріплює доступні автомобілі за відповідними складами базування, формуючи таким чином ізольовані набори даних для автономної обробки.

Ядро оптимізації застосовується незалежно до кожного сформованого кластера. Критичним архітектурним рішенням на цьому етапі став вибір методу розрахунку вартості переміщення. Хоча попередні дослідження підтвердили високу точність алгоритму A^* на дорожньому графі, його використання в ітеративному процесі оптимізації виявилось надлишковим та ресурсоемним. Тому для забезпечення роботи сервісу в режимі реального часу було впроваджено оптимізований підхід на основі формули гаверсинуса. Це дозволяє миттєво оцінювати геодезичну відстань між будь-якими точками, що значно прискорює перевірку гіпотез під час пошуку рішення.

Процес безпосередньої побудови маршруту розпочинається з генерації початкового рішення за допомогою жадібного алгоритму, який сортує замовлення за пріоритетом. Отриманий базовий маршрут передається до циклу табу-пошуку, де відбувається його ітеративне покращення. На кожній ітерації алгоритм досліджує окіл рішення через оператори переміщення, перевіряє дотримання обмежень вантажопідйомності та правил табу-листа, а також оновлює стан монітора трафіку для врахування потенційних заторів.

На завершальному етапі роботи конвеєра результати локальної оптимізації з усіх кластерів агрегуються в єдину структуру. Система розраховує фінальні статистичні показники, включаючи загальний час, дистанцію та рівень завантаження транспорту, після чого здійснює серіалізацію результату у формат JSON для передачі клієнтському додатку. Графічне представлення розробленого алгоритму наведено на рис. 2.13.

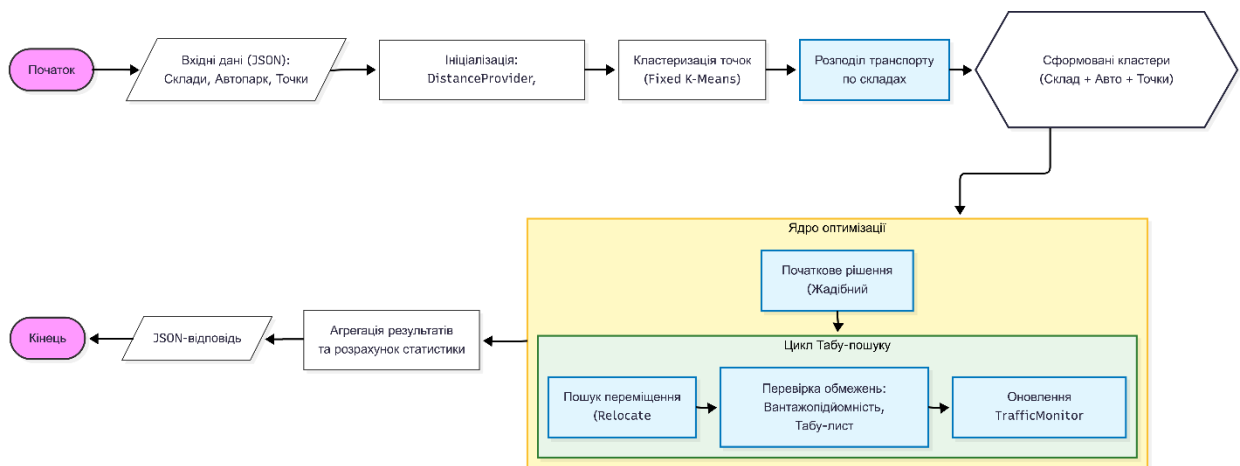


Рис.2.13. Послідовність виконання алгоритму.

Коректна робота алгоритму маршрутизації безпосередньо залежить від чіткої структуризації та валідності вхідної інформації. Взаємодія між основним сервером застосунку та обчислювальним сервісом реалізована через REST API, який приймає дані у форматі JSON. Цей формат є результатом серіалізації об'єктної моделі запиту, що забезпечує типізацію та цілісність даних на етапі їх передачі.

Вхідний пакет даних складається з трьох ключових масивів, що описують стан логістичної мережі. Перший масив містить інформацію про склади, де для кожного об'єкта вказується унікальний ідентифікатор та географічні координати (широта і довгота), що є точками прив'язки для транспортних засобів. Другий компонент описує автопарк. Критично важливими параметрами тут є вантажопідйомність транспортного засобу, яка виступає жорстким обмеженням при формуванні маршрутів, та ідентифікатор складу базування, що дозволяє алгоритму коректно розподіляти машини по кластерах на етапі декомпозиції. Третій масив представляє точки доставки, кожна з яких характеризується географічним розташуванням та вагою вантажу, що визначає попит клієнта.

Перед передачею цих даних на вхід алгоритмічного ядра виконується етап препроцесингу. Його важливість обумовлена необхідністю приведення різномірних даних до уніфікованого вигляду, придатного для математичних розрахунків. На цьому етапі здійснюється валідація географічних координат

(перевірка на належність до допустимого діапазону) та нормалізація числових типів даних. Також виконується попередня перевірка на фізичну здійсненність задачі: аналізується співвідношення сумарної ваги замовлень до загальної вантажопідйомності доступного флоту. Така фільтрація дозволяє виявити потенційні колізії ще до початку ресурсоємних ітерацій оптимізації, забезпечуючи стабільність роботи сервісу та запобігаючи помилкам під час виконання розрахунків.

Першим функціональним блоком обчислювального конвеєра є модуль геометричної декомпозиції. Його завдання полягає у розподілі глобального масиву замовлень між доступними логістичними центрами. Програмно цей етап реалізовано у функції `fixed_kmeans`, яка приймає на вхід два списки словників: `points` (точки доставки з координатами) та `warehouses` (склади).

Алгоритм функціонує за принципом мінімізації відстані. Для кожної точки доставки обчислюється дистанція до всіх доступних складів, і точка призначається до того депо, відстань до якого є найменшою. Важливою особливістю реалізації є використання метрики Haversine (відстань по великому колу) замість евклідової відстані, що забезпечує коректність розрахунків на сферичній поверхні Землі.

Лістинг реалізації алгоритму наведено нижче:

```
def fixed_kmeans(points, warehouses):
    clusters = {w["id"]: [] for w in warehouses}

    if not warehouses:
        return clusters

    warehouse_coords = {
        w["id"]: (w["latitude"], w["longitude"])
        for w in warehouses
    }

    for point in points:
        point_coord = (point["latitude"], point["longitude"])
        nearest_wh_id = min(
            warehouses,
            key=lambda wh: haversine(point_coord, warehouse_coords[wh["id"]]),
        )["id"]
        clusters[nearest_wh_id].append(point["id"])

    return clusters
```

Функція повертає словник, де ключем є унікальний ідентифікатор складу (`warehouse_id`), а значенням – список ідентифікаторів точок (`point_ids`), закріплених за ним.

Окрему увагу в реалізації приділено обробці граничних випадків, зокрема ситуації з порожніми кластерами. Словник `clusters` ініціалізується ключами для всіх переданих складів. Якщо в результаті роботи алгоритму до певного складу не було віднесено жодної точки (наприклад, через його значну віддаленість від районів замовлень), відповідний список у словнику залишається порожнім. На наступних етапах конвеєра (в функції-оркестраторі) такі кластери автоматично детектуються та виключаються з процесу маршрутизації, що дозволяє уникнути запуску зайвих ітерацій оптимізації та економить обчислювальні ресурси. Також алгоритм є стійким до викидів: навіть віддалені точки будуть гарантовано прив'язані до найближчого (хоча і далекого) складу, що забезпечує повне покриття всіх замовлень.

Після завершення етапу геометричної декомпозиції подальший обчислювальний процес виконується незалежно для кожного сформованого кластера. Такий архітектурний підхід дозволяє уникнути побудови глобальної матриці відстаней розмірністю $N \times N$, що було б надлишковим та ресурсоємним для системи реального часу. Натомість для кожного кластера формується локальний контекст даних, який включає лише закріплені за ним точки доставки та відповідний склад.

Для забезпечення вартості переміщення між вузлами в межах цих локальних груп було розроблено спеціалізований модуль `DistanceProvider`. Хоча попередні дослідження підтвердили високу точність алгоритму A^* на дорожньому графі, його використання в ітеративному циклі оптимізації виявилось надто повільним. Тому у фінальній реалізації застосовано стратегію «лінивих» обчислень (*lazy evaluation*) з використанням формули гаверсінуса. Відстань між двома точками розраховується аналітично як ортодромія

(найкоротша відстань на сфері) безпосередньо у момент звернення до неї алгоритму.

```
EARTH_RADIUS_M = 6_371_000.0

def haversine(coord_a, coord_b):
    lat1, lon1 = map(radians, coord_a)
    lat2, lon2 = map(radians, coord_b)

    dlat = lat2 - lat1
    dlon = lon2 - lon1

    a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ** 2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    return EARTH_RADIUS_M * c

class DistanceProvider:
    def __init__(self, points, warehouses):
        self.index_to_coord = {}
        # ... ініціалізація індексів ...
        self._cache = {}

    def distance(self, idx_a, idx_b):
        if idx_a == idx_b: return 0.0
        key = (min(idx_a, idx_b), max(idx_a, idx_b))

        if key not in self._cache:
            coord_a = self.index_to_coord[idx_a]
            coord_b = self.index_to_coord[idx_b]
            # Миттєвий розрахунок замість пошуку на графі
            self._cache[key] = haversine(coord_a, coord_b)

        return self._cache[key]
```

Ключовою перевагою такої реалізації є неявна сегментація розрахунків. Оскільки алгоритм маршрутизації запускається ізольовано для кожного кластера, система запитує відстані виключно між об'єктами, що входять до цієї групи. Відстані між точками з різних кластерів ніколи не обчислюються і не зберігаються в оперативній пам'яті. Така оптимізація забезпечує високу масштабованість сервісу та ефективне використання ресурсів сервера при збільшенні кількості замовлень.

Центральним елементом системи маршрутизації є модуль оптимізації, побудований на базі метаевристичного алгоритму табу-пошуку. Його завданням є знаходження субоптимального розв'язку задачі VRP у межах окремого кластера за обмежений час. Головна функція маршрутизації

tabu_search виступає оркестратором цього процесу, керуючи ініціалізацією, ітеративним пошуком та управлінням пам'яттю алгоритму.

Процес оптимізації розпочинається з побудови базового допустимого розв'язку за допомогою функції initial_solution. Для цього використовується жадібна стратегія, згідно з якою список клієнтів попередньо сортується за спаданням попиту (ваги вантажу). Алгоритм послідовно намагається розмістити найбільш «важкі» замовлення у першому доступному транспортному засобі, що має достатній запас вантажопідйомності. У випадках, коли клієнт не вміщується в жоден автомобіль без порушення обмежень, він примусово додається до маршруту з найменшим відносним завантаженням, що забезпечує створення технічно правильного рішення для подальшого покращення.

```
def initial_solution(customer_ids, demand_dict, vehicle_list):
    routes = [
        {
            "vehicle_id": vehicle["id"],
            "capacity": vehicle["capacity"],
            "customers": [],
            "load": 0,
        }
        for vehicle in vehicle_list
    ]

    if not customer_ids:
        return routes

    sorted_ids = sorted(customer_ids, key=lambda cid: demand_dict[cid],
                        reverse=True)

    for customer_id in sorted_ids:
        placed = False
        demand = demand_dict[customer_id]
        for route in routes:
            if route["load"] + demand <= route["capacity"]:
                route["customers"].append(customer_id)
                route["load"] += demand
                placed = True
                break

        if not placed:
            overloaded_route = min(routes, key=lambda r: r["load"] /
                                   r["capacity"])
            overloaded_route["customers"].append(customer_id)
            overloaded_route["load"] += demand

    return routes
```

Оцінка якості знайдених рішень виконується ієрархічно через функції `route_metrics` та `evaluate_solution`. Функція метрик обчислює вартість конкретного маршруту шляхом підсумовування відстаней між точками, отриманих від провайдера даних, з урахуванням коефіцієнта завантаженості доріг від монітора трафіку. Якщо під час розрахунку виявляється перевищення ліміту вантажопідйомності транспортного засобу, до вартості додається значне штрафне значення, що дозволяє алгоритму тимчасово розглядати недопустимі варіанти для виходу з локальних мінімумів, але стимулює повернення до допустимої області рішень.

Покращення поточного маршруту відбувається шляхом дослідження його околу через оператор переміщення (`Relocate`), реалізований у функції `find_best_relocation`. Алгоритм перебирає клієнтів у всіх маршрутах і намагається перенести їх в інші доступні послідовності, перевіряючи при цьому дотримання жорстких умов щодо місткості цільового автомобіля. Для запобігання зацикленню процесу оптимізації використовується механізм табу-листа, який зберігає історію останніх виконаних переміщень і блокує зворотні дії протягом певної кількості ітерацій, забезпечуючи ефективну диверсифікацію пошуку. Повний код пошуку маршруту знаходиться у додатку А.

Після завершення роботи ядра оптимізації для всіх сформованих кластерів система переходить до етапу агрегації. Цей процес виконується функцією-оркестратором, яка послідовно збирає результати локальних розрахунків у єдину глобальну структуру даних. Оскільки кожен кластер оброблявся ізольовано, на цьому етапі відбувається об'єднання маршрутних листів, прив'язаних до різних складів, у загальний план перевезень.

Під час агрегації виконується розрахунок підсумкових метаданих, необхідних для оцінки ефективності логістики. Система підсумовує ключові показники всіх маршрутів, обчислюючи загальну дистанцію пробігу автопарку (`total_distance`) та сумарний час роботи водіїв. Окрім інтегральних показників, визначається критичний параметр `makespan` – тривалість

найдовшого маршруту, що визначає час повного завершення всіх доставки. Також для кожної точки в маршруті генерується порядковий номер (order), що фіксує точну послідовність об'їзду, необхідну для коректної візуалізації шляху на карті клієнтського додатку.

Сформований результат серіалізується у формат JSON та передається у відповіді основному серверу застосунку. Така структура вихідних даних дозволяє бекенду автоматично десеріалізувати отриману інформацію в типізований об'єкт відповіді. Це забезпечує зручне вилучення ключових метрик для їх окремого збереження в полях бази даних, а також дозволяє зберегти повний зліпок розрахунку. Такий підхід гарантує цілісність даних та спрощує подальшу роботу з ними – від зміни статусу заявки до передачі готового маршруту на клієнтський інтерфейс.

Висновки до розділу 2

У другому розділі здійснено математичне моделювання та алгоритмічне проєктування системи маршрутизації. Обґрунтовано доцільність застосування стратегії декомпозиції «Cluster-First, Route-Second», яка дозволила трансформувати NP-важку задачу VRP у послідовність локальних оптимізаційних підзадач. На основі проведеного експериментального дослідження визначено оптимальну комбінацію методів: Fixed K-Means для логічного розподілу замовлень між складами та алгоритм табу-пошуку для побудови маршрутів, який продемонстрував найкращий баланс між якістю рішень та швидкодією порівняно з генетичним та мурашиним алгоритмами.

Розроблено узагальнений алгоритм роботи системи, реалізований у вигляді обчислювального конвеєра. Для забезпечення високої продуктивності в режимі реального часу було оптимізовано модуль розрахунку метрик: ресурсоємний пошук шляху на графі замінено на швидкий аналітичний розрахунок дистанцій за формулою гаверсінуса. Спроектowana архітектура, що включає валідацію вхідних даних, динамічне врахування трафіку та

ітеративну оптимізацію, створює надійну основу для програмної реалізації веб-сервісу в наступному розділі.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВІСУ ДЛЯ ОПТИМІЗАЦІЇ ЛОГІСТИЧНИХ МАРШРУТІВ

3.1. Архітектура програмного комплексу та вибір засобів розробки

При проєктуванні програмного комплексу було обрано багаторівневу клієнт-серверну архітектуру з елементами мікросервісного підходу. Такий вибір обумовлений необхідністю чіткого розмежування зон відповідальності між компонентами системи, що забезпечує гнучкість розробки, можливість незалежного масштабування окремих модулів та спрощує подальшу підтримку коду. Структурно система складається з чотирьох основних логічних рівнів, кожен з яких виконує специфічний набір функцій та реалізований за допомогою оптимального для своїх задач технологічного стека.

Рівень клієнта відповідає за безпосередню взаємодію з кінцевим користувачем. Він реалізований як односторінковий веб-застосунок (SPA) на базі фреймворку Angular. Вибір цієї технології у поєднанні з бібліотекою графічних компонентів Material UI дозволив створити адаптивний та реактивний інтерфейс, що забезпечує відображення графічних елементів, прийом вхідних даних та візуалізацію результатів розрахунку на інтерактивній карті. Головною задачею цього рівня є забезпечення зручного користувацького досвіду та первинна валідація даних на стороні браузера, що розвантажує сервер від обробки некоректних запитів.

Центральною ланкою системи виступає основний сервер застосунку, реалізований на платформі .NET з використанням мови програмування C# та технології ASP.NET Core. Цей вибір аргументується високою продуктивністю платформи, строгою типізацією та надійністю, необхідною для реалізації бізнес-логіки, автентифікації та авторизації користувачів. Для взаємодії з даними використано ORM-систему Entity Framework Core, що значно спрощує управління довідниками складів та автопарку. Основний сервер діє як оркестратор: він приймає завдання на розрахунок від клієнта, зберігає їхній

статус та делегує складні обчислювальні задачі спеціалізованому сервісу, забезпечуючи узгодженість роботи всієї системи.

Для виконання ресурсоемних математичних операцій виділено окремий сервіс обчислень, розроблений мовою Python із використанням мікрофреймворку Flask. Таке рішення дозволило розгорнути алгоритмічне ядро системи у вигляді легковагого REST-сервісу, який відповідає за кластеризацію точок, розрахунок матриць відстаней і евристичну оптимізацію маршрутів методом табу-пошуку. Винесення цієї логіки в окремий сервіс дозволяє ефективно використовувати потужні бібліотеки для наукових обчислень, такі як NumPy, не перевантажуючи основний веб-сервер та забезпечуючи ізоляцію обчислювальних процесів.

Важливим елементом архітектури є рівень зберігання даних. Для цього обрано систему керування базами даних MS SQL Server. Вона характеризується високою надійністю транзакцій та відмінною інтеграцією з платформою .NET. Взаємодія з базою даних здійснюється виключно через основний бекенд, що гарантує цілісність та безпеку інформації. Така архітектура дозволяє досягти високої продуктивності системи, оскільки важкі математичні обчислення виконуються асинхронно і не блокують обробку HTTP-запитів користувачів.

Основою інформаційної взаємодії між компонентами системи є архітектурний стиль REST, який визначає єдиний інтерфейс для обміну даними між клієнтом, сервером та сервісами. Обмін інформацією здійснюється за протоколом HTTP з використанням формату JSON, що забезпечує універсальність інтеграції різних платформ та легкість масштабування. Практичну реалізацію цього підходу відображено на діаграмі послідовності (рис. 3.1), яка ілюструє динаміку обробки даних на етапі їх підготовки.

Життєвий цикл роботи розпочинається на рівні клієнтського інтерфейсу Angular, де користувач через інтерактивні форми вводить параметри логістичної мережі: координати складів, характеристики автопарку та

специфікації точок доставки. Після завершення введення та проходження первинної валідації на стороні клієнта, веб-застосунок формує та відправляє асинхронний POST-запит до відповідного контролера API основного сервера. Отримавши дані, бекенд на платформі .NET виконує перевірку бізнес-правил і, у разі успіху, ініціює транзакцію збереження нової сутності в реляційній базі даних. Процедура завершується отриманням підтвердження від бази даних, після чого сервер повертає клієнту статус успішного виконання запиту, а інтерфейс автоматично перенаправляє користувача на сторінку створеного об'єкта.

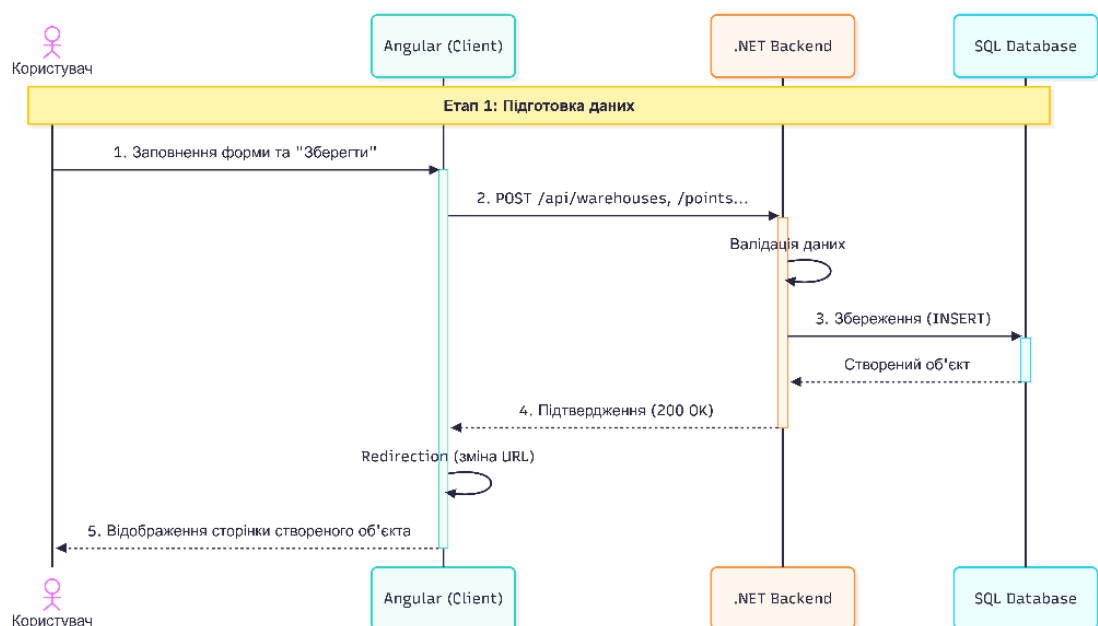


Рис.3.1. Створення сутностей.

Наступним, найбільш технологічно складним етапом роботи системи, є процес безпосереднього розрахунку оптимальних траєкторій, логіку якого візуалізовано на діаграмі послідовності (рис. 3.2). Ініціація сценарію відбувається шляхом створення користувачем нового екземпляра маршруту, який початково зберігається в базі даних зі статусом «чернетка». На цьому кроці система лише фіксує намір користувача та прив'язує до маршруту обрані склади, ще не виконуючи жодних математичних операцій.

Ключова фаза обробки розпочинається після натискання кнопки «Розрахувати» в інтерфейсі клієнтського застосунку. генерує запит до

контролера калькуляції на основному сервері, який починає діяти як оркестратор бізнес-процесу. Перед тим як делегувати завдання, бекенд звертається до бази даних SQL для консолідації повного контексту задачі: завантажує параметри доступних транспортних засобів та географічні координати точок доставки. Цей етап агрегації даних є критичним для забезпечення коректності вхідних параметрів для алгоритму

Сформований масив даних серіалізується у JSON та передається через внутрішній HTTP-запит на спеціалізований сервіс Python. Саме на цьому рівні, ізольовано від основного потоку обробки запитів, виконується ресурсоємне ядро алгоритму: геометрична кластеризація замовлень та оптимізація маршрутів методом табу-пошуку. Після завершення обчислень сервіс повертає структуровану відповідь, що містить геометрію шляху та розраховані метрики ефективності. Основний сервер обробляє отриманий результат, оновлює запис у базі даних, змінюючи статус маршруту на «готовий», та передає готовий об'єкт на клієнтську сторону. Завершується цикл графічною візуалізацією побудованих маршрутів на інтерактивній карті, що дозволяє логісту миттєво оцінити якість планування.

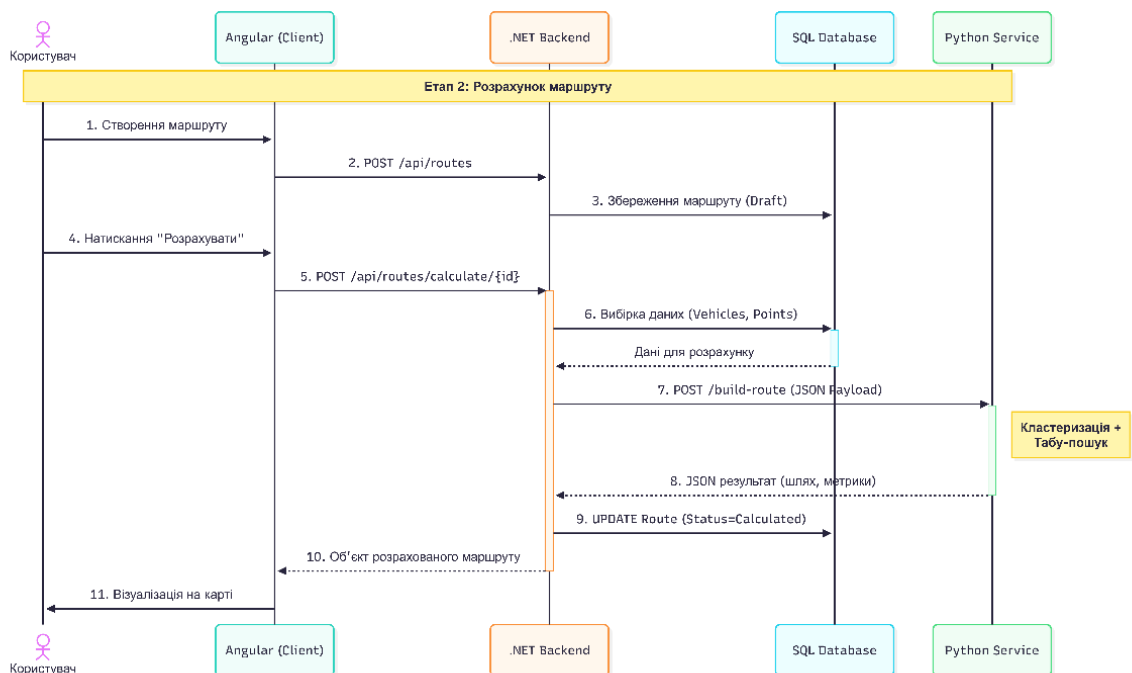


Рис.3.2. Створення та обрахунок маршруту.

Завершальним етапом життєвого циклу маршруту в системі є фаза його безпосереднього виконання та моніторингу, логіку якої відображено на діаграмі послідовності (рис. 3.3). Цей процес ініціюється оператором в інтерфейсі веб-застосунку шляхом натискання кнопки «Активувати» для обраного маршруту. Клієнтський застосунок формує запит PUT до відповідного API-ендпоінту, передаючи команду на зміну статусу. Основний сервер обробляє запит та оновлює відповідний запис у базі даних SQL, встановлюючи статус «активний». Це є сигналом для системи, що транспортний засіб вийшов на лінію, що відповідає стандартам управління подіями в цифрових логістичних ланцюгах.

Оскільки система на даному етапі функціонує в режимі прототипу без підключення до реальних GPS-трекерів, процес фізичного переміщення моделюється програмно. На клієнтській стороні запускається таймер симуляції, тривалість якого пропорційна розрахунковому часу рейсу. Успішне завершення роботи таймера імітує прибуття водія в кінцеву точку, після чого фронтенд автоматично надсилає запит на завершення маршруту. Сервер фіксує цей факт у базі даних, змінюючи статус на «завершено», і повертає код відповіді 204 No Content, що підтверджує успішну фіналізацію процесу.

Архітектурою також передбачено альтернативний сценарій (блок alt на діаграмі), який обробляє форс-мажорні обставини. Якщо під час виконання виникає необхідність скасувати рейс, оператор використовує команду «Відмінити». У цьому випадку система надсилає запит на ендпоінт, який перериває симуляцію та оновлює статус маршруту в базі даних на «скасовано». Така логіка забезпечує гнучкість управління та повну простежуваність усіх станів замовлення в реальному часі.

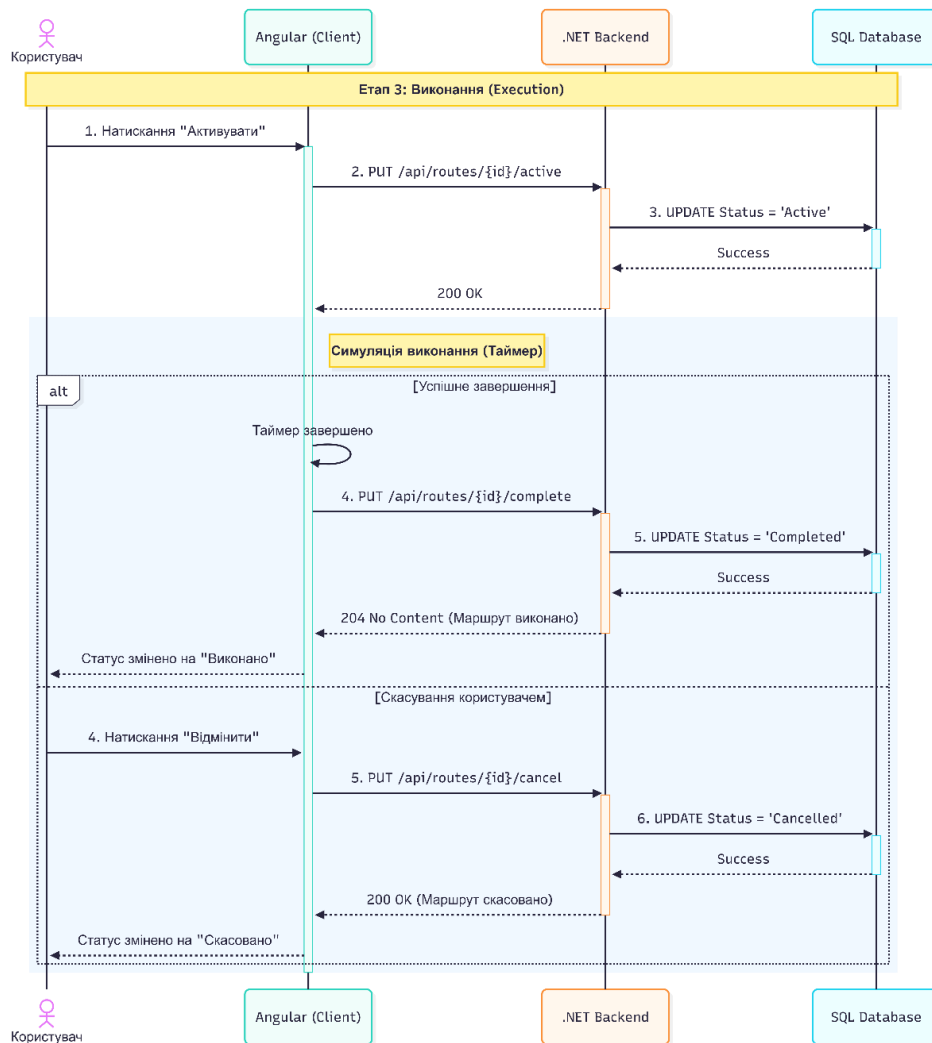


Рис.3.3. Зміна статусу маршруту.

Підсумовуючи вищевикладене, можна стверджувати, що спроектована модель взаємодії компонентів забезпечує надійність та узгодженість даних у системі. Чітке розмежування зон відповідальності між основним сервером, що керує бізнес-процесами, та спеціалізованим обчислювальним модулем дозволяє ефективно виконувати ресурсоємні задачі маршрутизації, не знижуючи загальної продуктивності веб-застосунку. Така децентралізація обчислювальних потужностей відповідає сучасним архітектурним вимогам до побудови інтелектуальних транспортних систем, де критично важливим є забезпечення мінімального часу відгуку при обробці складних комбінаторних задач, навіть в умовах пікового навантаження.

Окремо варто відзначити потенціал розробленої архітектури до горизонтального масштабування. Завдяки ізоляції математичного ядра в

окремий сервіс, система зберігає здатність до ефективної обробки зростаючого потоку заявок без необхідності втручання в логіку основного бекенду. Реалізований механізм управління життєвим циклом обрахунку маршруту – від його створення та калькуляції до фінального завершення – гарантує прозорість та контрольованість усіх логістичних операцій. Це створює надійний технологічний фундамент для реалізації інтуїтивно зрозумілого інтерфейсу користувача, детальний опис та сценарії роботи з яким наведено у наступному підрозділі.

3.2. Інтерфейс користувача та сценарії роботи з системою

Інтерфейс користувача розроблено з дотриманням принципів Material Design, що забезпечує інтуїтивну зрозумілість, візуальну цілісність та сучасний вигляд системи. Для реалізації графічної складової використано бібліотеку Angular Material, яка надає набір стандартизованих UI-компонентів, таких як картки, поля введення, таблиці даних та діалогові вікна. Використання єдиної бібліотеки стилів дозволило уніфікувати оформлення всіх модулів додатку, забезпечуючи передбачувану поведінку інтерфейсу та високу ергономіку взаємодії.

Організація робочого простору базується на фіксованій структурі макета. Основним навігаційним елементом є бічна панель, розташована ліворуч. Вона містить головне меню з посиланнями на ключові функціональні розділи системи: склади, транспорт, точки доставки та маршрути. Така архітектура навігації забезпечує користувачеві швидкий доступ до будь-якого модуля системи в один клік, незалежно від того, на якій сторінці він знаходиться в даний момент. Центральна частина екрана відведена під динамічне відображення контенту активного модуля.

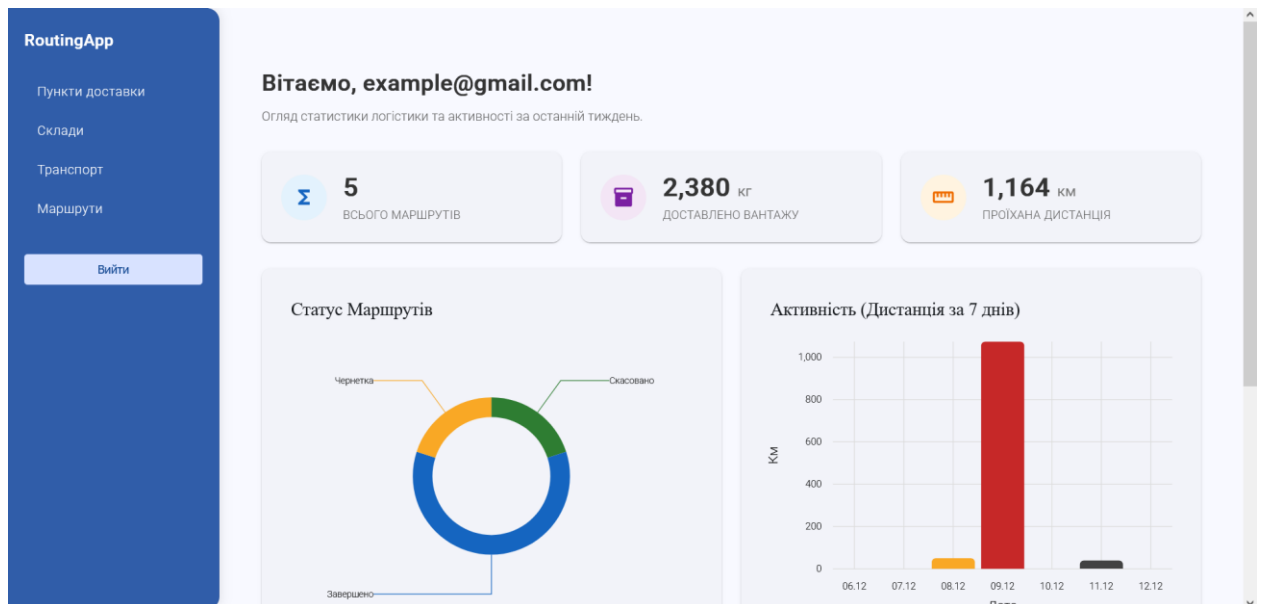


Рис.3.4. Головна сторінка.

Для забезпечення надійного захисту облікових записів та безпеки доступу до логістичних даних в системі інтегровано хмарну платформу ідентифікації Auth0. Це рішення дозволяє винести процеси автентифікації за межі основного коду додатку, використовуючи індустріальні стандарти протоколів OAuth 2.0 та OpenID Connect.

При спробі входу в систему користувач автоматично перенаправляється на захищену універсальну сторінку авторизації, що надається провайдером. Інтерфейс форми є лаконічним та інтуїтивно зрозумілим, містить поля для введення облікових даних (електронної пошти та пароля). Після успішної верифікації користувача Auth0 генерує захищений токен доступу у форматі JWT, який передається назад у веб-застосунок. Тільки за наявності валідного токена система відкриває доступ до робочого простору та функціонального меню, гарантуючи, що сторонні особи не зможуть переглядати або змінювати конфіденційні дані про маршрути та клієнтів.

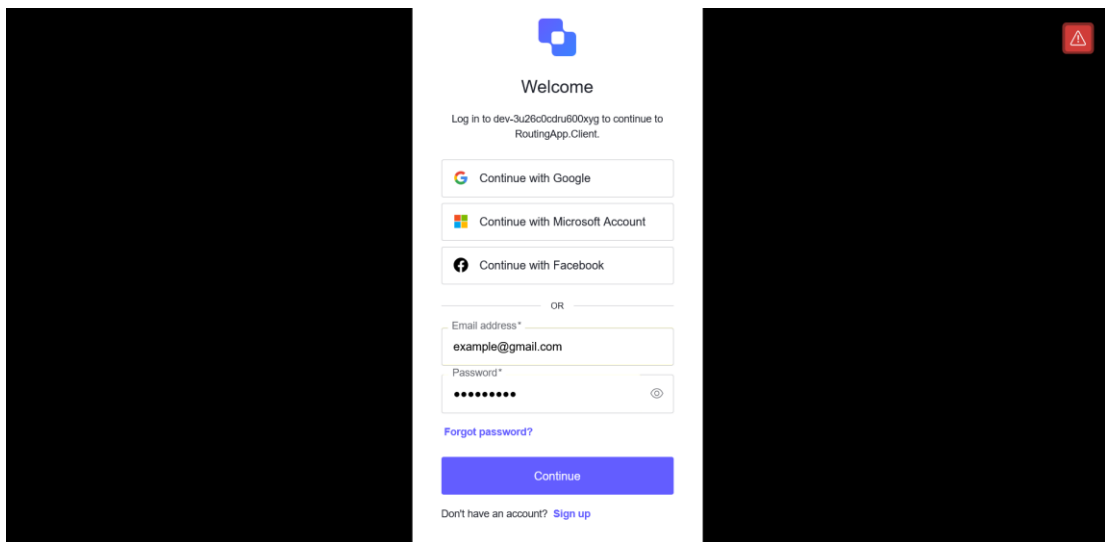


Рис.3.5. Сторінка авторизації користувача через сервіс Auth0.

Наступним кроком після успішної авторизації є наповнення системи вхідною інформацією, необхідною для роботи алгоритмів маршрутизації. Цей етап реалізовано через групу модулів управління довідниками, які дозволяють користувачеві формувати цифрову модель логістичної мережі. Робота розпочинається з розділу «Склади», де оператор має змогу переглядати реєстр наявних розподільчих центрів у вигляді таблиці та додавати нові об'єкти.

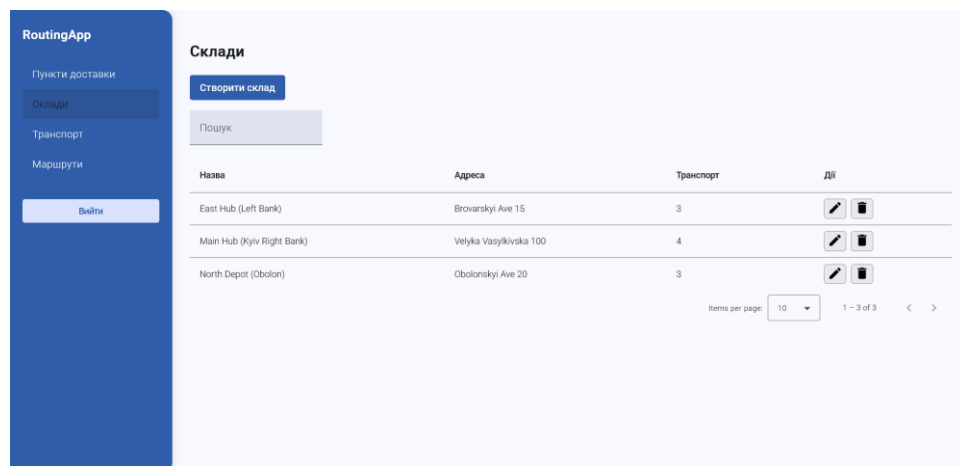


Рис.3.6. Сторінка «Склади».

Процес створення складу передбачає заповнення веб-форми. Тут вказуються унікальна назва та точні географічні координати (широта і довгота), що слугуватимуть стартовими точками для майбутніх рейсів. Також, можна відразу прив'язати авто, яке ще не прив'язане до жодного складу. Адресу та координати можна ввести як вручну, так і за допомогою карти чи пошуку.

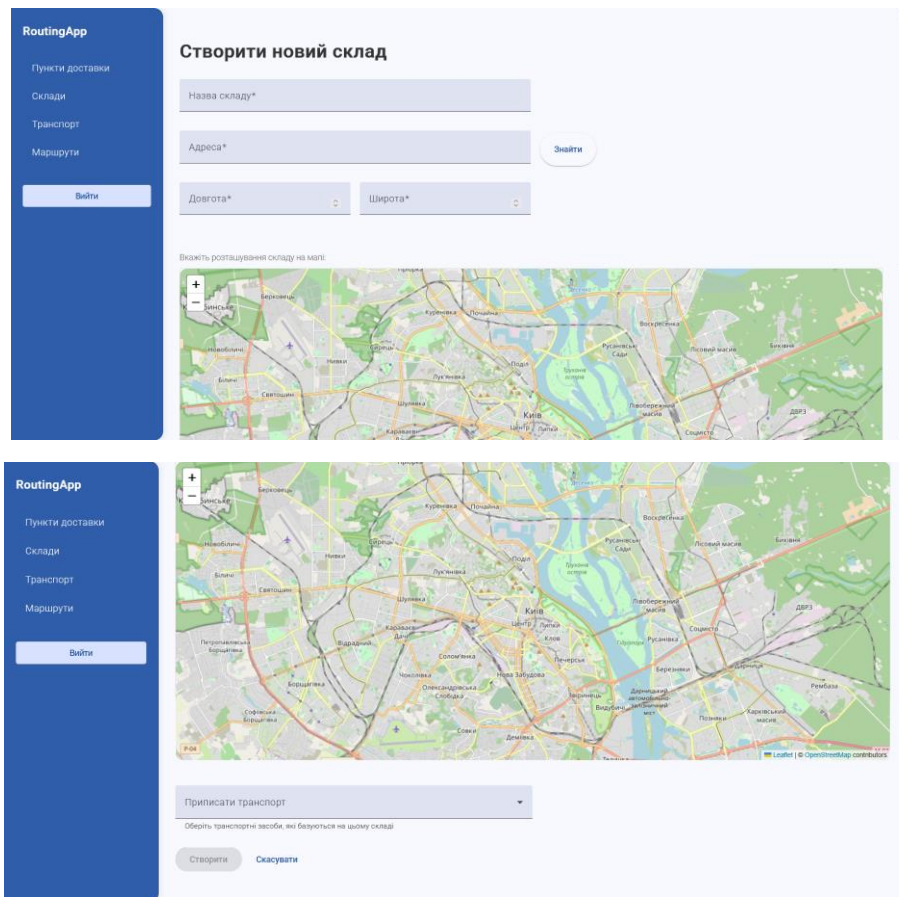


Рис.3.7 – 3.8. Форма створення складу.

Тісно пов'язаним із попереднім є модуль управління автопарком «Транспорт». Інтерфейс цього розділу дозволяє реєструвати транспортні засоби, обов'язково вказуючи їхню вантажопідйомність – критичний параметр, який використовується алгоритмом для контролю перевантажень. Важливою особливістю реалізації є логічна прив'язка кожного автомобіля до конкретного складу, що формує структуру парку для кожного депо окремо.

RoutingApp

Пункти доставки

Склади

Транспорт

Маршрути

Вийти

Транспорт

Створити транспортний засіб

Пошук

Назва	Вантажопідйомність	Статус	Warehouse Name	Адреса складу	Дії
Eco-Mover #99	800		North Depot (Obolon)	Obolonskyi Ave 20	
Express #77	800		East Hub (Left Bank)	Brovarskyi Ave 15	
Heavy Truck #500	5000		North Depot (Obolon)	Obolonskyi Ave 20	
Reserve #00	800		Main Hub (Kyiv Right Bank)	Velyka Vasykivska 100	
Sprinter #101	1500		Main Hub (Kyiv Right Bank)	Velyka Vasykivska 100	
Sprinter #102	1500		North Depot (Obolon)	Obolonskyi Ave 20	
Transit #201	1500		East Hub (Left Bank)	Brovarskyi Ave 15	
Transit #202	1500		Main Hub (Kyiv Right Bank)	Velyka Vasykivska 100	
Van #001	800		East Hub (Left Bank)	Brovarskyi Ave 15	

Рис.3.9. . Сторінка «Транспорт».

Рис.3.10. Форма додавання авто.

База клієнтських замовлень формується у розділі «Пункти доставки». Тут користувач вручну створює точки доставки, вводячи інформацію про їхнє розташування, або редагує існуючі записи, забезпечуючи актуальність даних перед початком планування. Аналогічно зі складами, адресу та координати можна ввести як вручну, так і за допомогою карти чи пошуку.

Назва	Адреса	Дії
Client Point #1	Street 1, Kyiv	
Client Point #10	Street 10, Kyiv	
Client Point #11	Street 11, Kyiv	
Client Point #12	Street 12, Kyiv	
Client Point #13	Street 13, Kyiv	
Client Point #14	Street 14, Kyiv	
Client Point #15	Street 15, Kyiv	
Client Point #16	Street 16, Kyiv	
Client Point #17	Street 17, Kyiv	

Рис.3.11 – 3.12. Сторінка «Пункти доставки» та форма створення.

Центральним функціональним блоком веб-сервісу є модуль планування маршрутів, який консолідує попередньо введені дані та запускає процеси оптимізації. Процедура розпочинається зі створення нового об'єкта маршруту. Інтерфейс пропонує користувачеві форму, де першочергово обирається базовий склад. На основі цього вибору система автоматично фільтрує списки доступних ресурсів, відображаючи лише ті транспортні засоби, що територіально або логічно прив'язані до обраного депо. Такий підхід унеможливорює виникнення логічних колізій ще на етапі підготовки даних. Після вибору необхідних замовлень та автопарку маршрут зберігається у системі зі статусом чернетки.

The image displays two screenshots of the 'RoutingApp' web interface, specifically the 'Create new route' form. Both screenshots show a sidebar on the left with navigation links: 'Пункти доставки', 'Склади', 'Транспорт', 'Маршрути', and a 'Вийти' button. The main content area is titled 'Створити новий маршрут'.

The top screenshot shows the initial state of the form with empty dropdowns for 'Назва маршруту*', 'Склади*', and 'Транспорт*'. Below these is a section for 'Точки доставки та вантаж' with a 'Точка доставки*' input field and a 'Вага (кг)*' input field with a clear button. A '+ Додати точку' button is also present. At the bottom are 'Створити' and 'Скасувати' buttons.

The bottom screenshot shows the form after some data has been entered. The 'Склади*' dropdown is now populated with 'Main Hub (Kyiv Right Bank), North Depot (Obolon)'. The 'Транспорт*' dropdown is open, showing a list of vehicles: 'Sprinter #101 (1500 kg)', 'Transit #202 (1500 kg)', 'Reserve #00 (800 kg)', and 'Sprinter #102 (1500 kg)'. The 'Вага (кг)*' input field is also visible. The '+ Додати точку' button and the 'Створити'/'Скасувати' buttons are still present.

Рис.3.13 – 3.14. Форма створення маршруту.

Ініціалізація обчислювального процесу відбувається шляхом натискання кнопки «Перерахувати маршрут» на панелі інструментів маршруту. Після отримання відповіді від алгоритмічного ядра інтерфейс переходить у режим перегляду результатів.

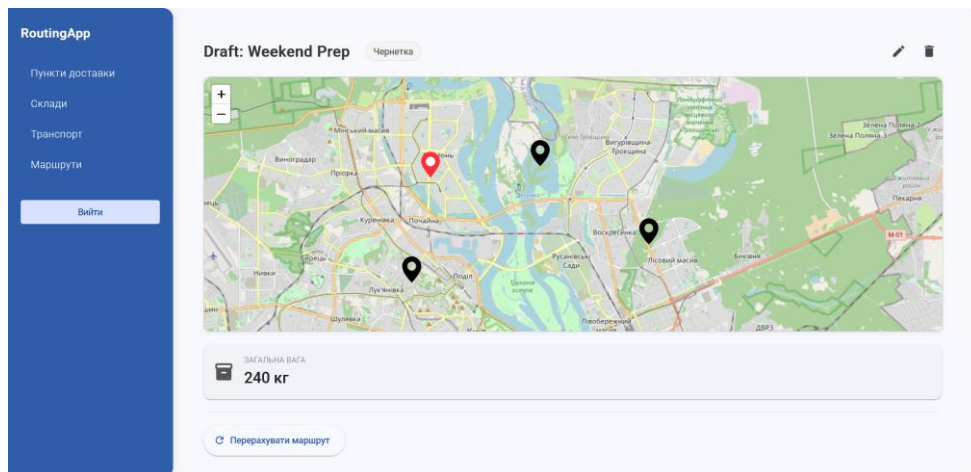


Рис.3.15. Сторінка інформації про маршрут.

Для візуалізації траєкторій руху використано картографічну бібліотеку Leaflet з підкладкою OpenStreetMap. На карті відображається геометрія побудованого шляху, що з'єднує склад та клієнтські точки в оптимальній послідовності. Окрім візуальної складової, система виводить блок ключових метрик ефективності: загальну розрахункову дистанцію пробігу, прогнозований час виконання рейсу.

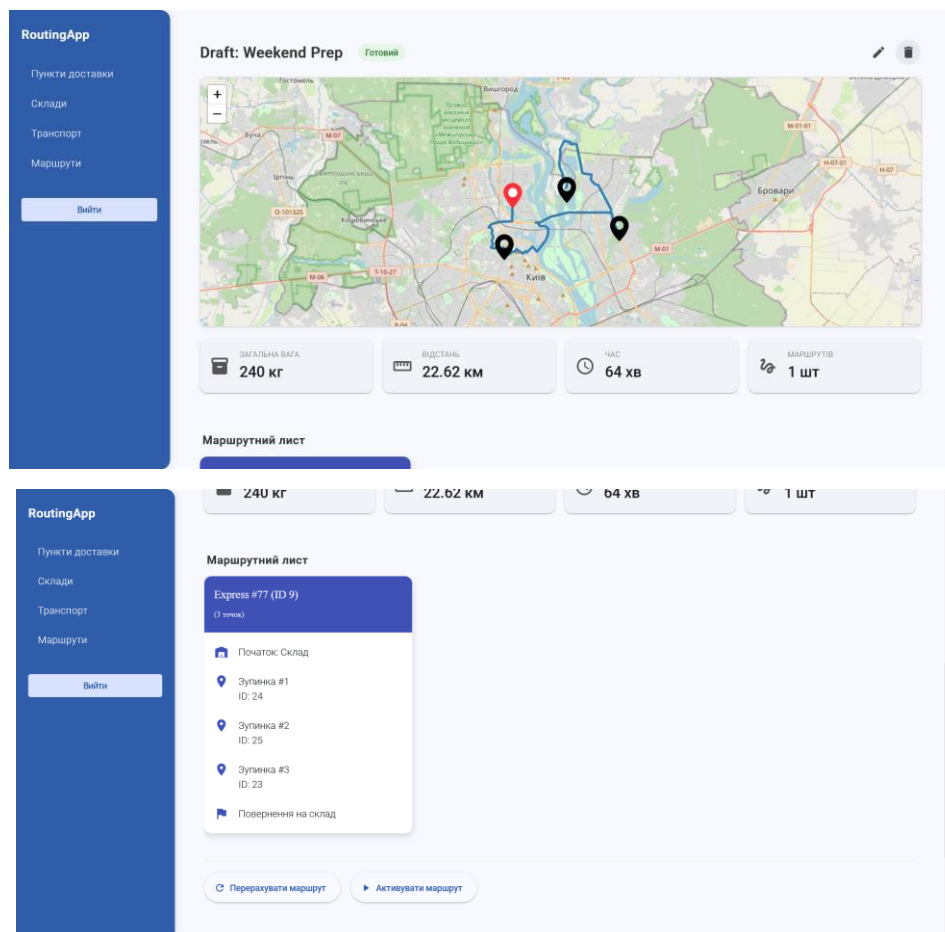


Рис.3.16 – 3.17. Результати обрахунку маршруту.

Важливою складовою функціональності системи є інструментарій для моніторингу та управління життєвим циклом транспортних операцій. Після перевірки та затвердження розрахованого плану диспетчер має можливість перевести його в режим виконання за допомогою функції активації. Ця дія ініціює початок логістичного процесу та фіксує параметри маршруту, унеможливаючи випадкове редагування його складу під час виконання рейсу.

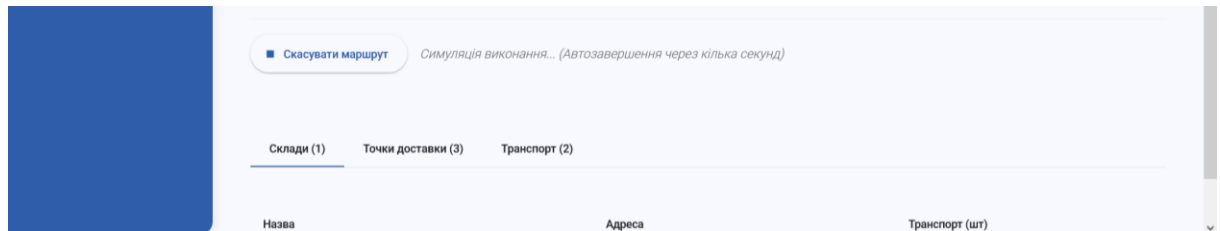


Рис.3.18. Активація маршруту.

Для забезпечення наочності та швидкої орієнтації в інтерфейсі реалізовано систему кольорової індикації станів. Кожен запис у списку маршрутів супроводжується відповідним візуальним маркером (бейджем): статус «Чернетка» позначає етап планування, яскравий індикатор «Активний» сигналізує про те, що транспортний засіб знаходиться на лінії, а статуси «Завершено» або «Скасовано» фіксують фінальний результат операції.

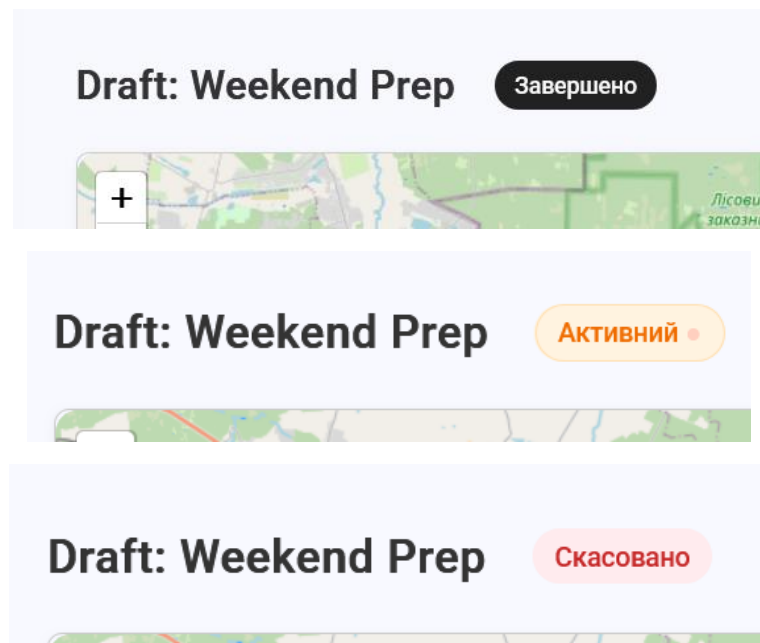


Рис.3.19 – 3.21. Відображення статусу маршруту.

Підсумовуючи огляд клієнтської частини, слід зазначити, що спроектований інтерфейс забезпечує високий рівень зручності використання завдяки дотриманню стандартів дизайну та логічному структуруванню навігації. Оптимізація сценаріїв взаємодії дозволила мінімізувати кількість необхідних дій оператора: повний цикл роботи, від вибору вхідних даних до отримання візуалізованого маршруту на карті, реалізується за декілька кліків. Такий підхід дозволяє ефективно використовувати систему персоналу без глибокої технічної підготовки, приховуючи складність математичних алгоритмів за інтуїтивно зрозумілою та ергономічною візуальною оболонкою.

Висновки до розділу 3

У третьому розділі здійснено опис веб-сервісу для автоматизованої побудови логістичних маршрутів. Розроблена гібридна архітектура, що поєднує основний сервер на ASP.NET Core та спеціалізований обчислювальний сервіс на Python, забезпечила ефективне розмежування бізнес-логіки та ресурсномістких математичних алгоритмів. Використання Angular для клієнтської частини та MS SQL Server для зберігання даних, дозволило створити гнучку, масштабовану систему, здатну виконувати розрахунки в асинхронному режимі без зниження швидкодії обробки користувацьких запитів.

Спроектований інтерфейс користувача надає інтуїтивно зрозумілі інструменти для повного циклу управління логістикою: від ведення довідників складів та автопарку до візуалізації побудованих маршрутів на інтерактивній карті. Реалізація сценаріїв захищеної авторизації, автоматичної валідації даних та моніторингу статусів виконання рейсів підтверджує практичну цінність створеного прототипу. Розроблений програмний комплекс успішно вирішує задачі автоматизації етапу «останньої милі», значно скорочуючи час на планування та мінімізуючи вплив людського фактора на якість логістичних рішень.

ВИСНОВКИ

У кваліфікаційній роботі вирішено науково-прикладну задачу підвищення ефективності управління вантажними перевезеннями на етапі «останньої милі». Головним результатом дослідження є розробка та програмна реалізація веб-орієнтованої системи підтримки прийняття рішень, в основу якої покладено удосконалений гібридний алгоритм маршрутизації.

У першому розділі проведено комплексний аналіз теоретичних основ транспортної логістики, який виявив ключові проблеми сучасного ринку, зокрема зростання витрат на етапі «останньої милі» в умовах урбанізації та розвитку електронної комерції. Здійснено класифікацію математичних моделей маршрутизації, що дозволило ідентифікувати задачу VRP як базову для подальшого моделювання. Критичний огляд існуючих програмних засобів автоматизації засвідчив, що більшість комерційних рішень є або надто коштовними, або складними для адаптації під специфічні потреби малого та середнього бізнесу, що обґрунтувало доцільність розробки власної інформаційної системи.

У другому розділі розроблено концептуальну модель системи та здійснено декомпозицію задачі маршрутизації за стратегією «Cluster-First, Route-Second». На основі дослідження ефективності різних підходів обґрунтовано вибір гібридного алгоритму, що поєднує метод кластеризації для розподілу точок між складами та метаевристичний алгоритм табу-пошуку для оптимізації маршрутів усередині кожного кластера. Така комбінація, доповнена оптимізованим механізмом розрахунку дистанцій, забезпечила високу швидкість обчислень та здатність знаходити субоптимальні рішення з мінімальною похибкою, що було підтверджено в ході моделювання.

Третій розділ присвячено програмній реалізації спроектованої системи у вигляді веб-сервісу. Розроблено архітектуру комплексу, яка базується на взаємодії основного сервера на платформі .NET для обробки бізнес-логіки та спеціалізованого сервісу на Python для виконання математичних алгоритмів. Реалізовано зручний інтерфейс користувача, що дозволяє виконувати повний

цикл логістичних операцій: від ведення довідників складів і автопарку до візуалізації побудованих маршрутів на карті та моніторингу їх виконання.

Загальним результатом роботи є створення ефективного інструментарію, який дозволяє транспортним підприємствам мінімізувати вплив людського фактора, скоротити експлуатаційні витрати та підвищити якість обслуговування клієнтів. Перспективи подальшого розвитку системи полягають в інтеграції з сервісами моніторингу дорожнього трафіку в реальному часі та розробці мобільного клієнта для водіїв.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Власенко О. В., Рябова Є. В. Порівняльний аналіз метаевристичних алгоритмів для оптимізації маршрутів доставки // Інформаційні технології і автоматизація – 2025 : матеріали XVIII Міжнар. наук.-практ. конф. (Одеса, 30–31 жовт. 2025 р.). Одеса : ОНТУ, 2025. С. 88–89.
2. Transforming Urban Logistics: Sustainable and Efficient Last-Mile Delivery in Cities. World Economic Forum. Geneva, 2024. 28 p. URL: https://www3.weforum.org/docs/WEF_Transforming_Urban_Logistics_2024.pdf
3. The Last-Mile Delivery Challenge: Giving retail and consumer product customers a superior delivery experience. Capgemini Research Institute. URL: <https://www.capgemini.com/insights/research-library/the-last-mile-delivery-challenge/>
4. Muñoz-Villamizar A., Solano-Charris E. L., Reyes-Rubiano L., Faulin J. Measuring Disruptions in Last-Mile Delivery Operations. Logistics. 2021. Vol. 5, No. 1. Art. 17. DOI: <https://doi.org/10.3390/logistics5010017>
5. Braekers K., Ramaekers K., Nieuwenhuysen I. The Vehicle Routing Problem: State of the Art Classification and Review. Computers & Industrial Engineering. 2015. Vol. 99. DOI: 10.1016/j.cie.2015.12.007.
6. Kuijpers J. Comparing Clustering Algorithms in a Time Window Constrained Vehicle Routing Problem: MSc Thesis : Thesis code ORL 88843. Wageningen University, Department of Social Sciences, Operations Research and Logistics. Wageningen, 2023.
7. Bulková Z., Gašparík J., Čamaj J. Implementation of automated systems in logistics: The key to efficiency and environmental sustainability. Transportation Research Procedia. 2025. Vol. 87. P. 1–10. DOI: <https://doi.org/10.1016/j.trpro.2025.04.100>.
8. Sarioguz O. Artificial Intelligence for sustainable logistics: Reducing carbon emissions and fuel consumption through route optimization. International

- Journal of Science and Research Archive. 2025. Vol. 15, No. 3. P. 1527–1537.
DOI: <https://doi.org/10.30574/ijsra.2025.15.3.1933>.
9. Kothimbire D. K., Shelke D. S., Gaikwad S. V., Yelpale A. P., Shinde R. N. A Comprehensive Review of Graph Theory Applications in Network Analysis. International Journal of Mathematics and Computer Research. 2025. Vol. 13, Iss. 03. P. 4956–4967. DOI: <https://doi.org/10.47191/ijmcr/v13i3.08>.
 10. Guze S. Review of methods and algorithms for modelling transportation networks based on graph theory. Scientific Journal of Gdynia Maritime University. 2018. No. 107. P. 25–39. DOI: 10.26408/107.02.
 11. Rachmawati D., Gustin L. Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem. Journal of Physics: Conference Series. 2020. Vol. 1566, No. 1. Art. 012061. DOI: 10.1088/1742-6596/1566/1/012061.
 12. Foead D., Ghifari A., Kusuma M. B., Hanafiah N., Gunawan E. A Systematic Literature Review of A* Pathfinding. Procedia Computer Science. 2021. Vol. 179. P. 507–514. DOI: <https://doi.org/10.1016/j.procs.2021.01.034>.
 13. Liu F., Lu C., Gui L., Zhang Q., Tong X. Heuristics for Vehicle Routing Problem: A Survey and Recent Advances. arXiv preprint arXiv:2303.04147. 2023. URL: <https://arxiv.org/pdf/2303.04147>.
 14. Mahmudy W. F., Widodo A. W., Haikal A. H. Challenges and Opportunities for Applying Meta-Heuristic Methods in Vehicle Routing Problems: A Review. Engineering Proceedings. 2024. Vol. 63. Art. 12. DOI: <https://doi.org/10.3390/engproc2024063012>.
 15. Malinen M. I., Fränti P. Balanced K-Means for Clustering // Structural, Syntactic, and Statistical Pattern Recognition / eds. P. Fränti et al. Berlin : Springer, 2014. P. 32–41. (Lecture Notes in Computer Science; Vol. 8621). DOI: https://doi.org/10.1007/978-3-662-44415-3_4.
 16. Kumar S. N., Panneerselvam R. A Survey on the Vehicle Routing Problem and Its Variants. Intelligent Information Management. 2012. Vol. 4, No. 3. P. 66–74. DOI: <https://doi.org/10.4236/iim.2012.43010>.

17. Thao N. L. P., Vinh L. N. H., Yu V. F., Phuc P. N. K. The Improved K-Means Algorithm for Clustering Optimization: A Comparative Study on VRP Dataset. // Proceedings of the 15th International Conference on Industrial Engineering and Operations Management. Singapore : IEOM Society International, 2025. DOI: <https://doi.org/10.46254/AN15.20250257>.
18. Abdoune S., Boulif M. Towards a better approach to the Vehicle Routing Problem. arXiv preprint arXiv:2506.23028. 2025. URL: <https://arxiv.org/abs/2506.23028>.
19. Notice D., Soleimani H., Pavlidis N. G., Kheiri A., Muñoz M. A. Instance Space Analysis of the Capacitated Vehicle Routing Problem with Mixture Discriminant Analysis // Genetic and Evolutionary Computation Conference (GECCO '25) (Málaga, Spain, July 14–18, 2025). New York : ACM, 2025. P. 1172–1180. DOI: <https://doi.org/10.1145/3712256.3726405>.
20. Garside A. K., Laili N. R. A Cluster-First Route-Second Heuristic Approach to Solve The Multi-Trip Periodic Vehicle Routing Problem. Jurnal Teknik Industri. 2019. Vol. 20, No. 2. P. 172–181. DOI: <https://doi.org/10.22219/JTIUMM.Vol20.No2.172-181>.
21. Alesiani F., Ermis G., Gkiotsalitis K. Constrained Clustering for the Capacitated Vehicle Routing Problem (CC-CVRP). Applied Artificial Intelligence. 2022. Vol. 36, No. 1. Art. e1995658. DOI: <https://doi.org/10.1080/08839514.2021.1995658>.
22. Moghdani R., Salimifard K., Demir E., Barak S., Aazami A., Hosseini Shekarabi S. A. A metaheuristic approach for the multi-objective sustainable vehicle routing problem. Annals of Operations Research. 2025. DOI: <https://doi.org/10.1007/s10479-025-06904-1>.
23. Kumari M., De P. K., Chaudhuri K., Narang P. Utilizing a hybrid metaheuristic algorithm to solve capacitated vehicle routing problem. Results in Control and Optimization. 2023. Vol. 13. Art. 100292. DOI: <https://doi.org/10.1016/j.rico.2023.100292>.

24. Le T. D. C., Nguyen D. D., Oláh J., Pakurár M. Clustering algorithm for a vehicle routing problem with time windows. *Transport*. 2022. Vol. 37, no. 1. P. 17–27. DOI: <https://doi.org/10.3846/transport.2022.16850>.
25. Vargas-Muñoz J. E., Srivastava S., Tuia D., Falcão A. X. OpenStreetMap: Challenges and Opportunities in Machine Learning and Remote Sensing. *IEEE Geoscience and Remote Sensing Magazine*. 2021. Vol. 9, no. 1. P. 184–199. DOI: <https://doi.org/10.1109/MGRS.2020.2994107>.
26. Xu D., Tian Y. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science*. 2015. Vol. 2, no. 2. P. 165–193. DOI: <https://doi.org/10.1007/s40745-015-0040-1>.
27. Akhand M. A. H., Sultana T., Shuvo M. I. R., Al-Mahmud. Constructive and Clustering Methods to Solve Capacitated Vehicle Routing Problem. *Oriental Journal of Computer Science and Technology*. 2017. Vol. 10, no. 3. P. 656–664. URL: <http://www.computerscijournal.org/?p=6623>.
28. Kalisvaart J. Constrained K-means Clustering and Vehicle Routing : Master Thesis / Tilburg University. Eindhoven, 2022. 72 p.
29. Lähderanta T., Lovén L., Ruha L., Leppänen T., Launonen I., Riekkilä J., Sillanpää M. J. Capacitated spatial clustering with multiple constraints and attributes. *Engineering Applications of Artificial Intelligence*. 2024. Vol. 127. Art. 107182. <https://doi.org/10.1016/j.engappai.2023.107182>
30. Tan S.-Y., Yeh W.-C. The Vehicle Routing Problem: State-of-the-Art Classification and Review. *Applied Sciences*. 2021. Vol. 11, Iss. 21. Art. 10295. DOI: <https://doi.org/10.3390/app112110295>

ДОДАТКИ

Лістинг реалізації табу-пошуку

```

from collections import deque
import numpy as np
from copy import deepcopy

TABU_PENALTY = 1_000_000

def initial_solution(customer_ids, demand_dict, vehicle_list):
    routes = [
        {
            "vehicle_id": vehicle["id"],
            "capacity": vehicle["capacity"],
            "customers": [],
            "load": 0,
        }
        for vehicle in vehicle_list
    ]

    if not customer_ids:
        return routes

    sorted_ids = sorted(customer_ids, key=lambda cid: demand_dict[cid],
reverse=True)

    for customer_id in sorted_ids:
        placed = False
        demand = demand_dict[customer_id]
        for route in routes:
            if route["load"] + demand <= route["capacity"]:
                route["customers"].append(customer_id)
                route["load"] += demand
                placed = True
                break

        if not placed:
            overloaded_route = min(routes, key=lambda r: r["load"] /
r["capacity"])
            overloaded_route["customers"].append(customer_id)
            overloaded_route["load"] += demand

    return routes

def route_metrics(
    route,
    demand_dict,
    depot_index,
    provider,
    id_to_index,
    traffic_monitor=None,
):
    customers = route["customers"]
    load = route["load"]
    cost = 0.0
    prev_index = depot_index
    prev_coord = provider.index_to_coord[prev_index]

    for customer_id in customers:
        current_index = id_to_index[customer_id]

```

```

        current_coord = provider.index_to_coord[current_index]
        multiplier = (
            traffic_monitor.multiplier(prev_coord, current_coord)
            if traffic_monitor
            else 1.0
        )
        cost += provider.distance(prev_index, current_index) * multiplier
        prev_index = current_index
        prev_coord = current_coord

    depot_coord = provider.index_to_coord[depot_index]
    if prev_index != depot_index:
        multiplier = (
            traffic_monitor.multiplier(prev_coord, depot_coord)
            if traffic_monitor
            else 1.0
        )
        cost += provider.distance(prev_index, depot_index) * multiplier

    if load > route["capacity"]:
        cost += TABU_PENALTY

    return cost, load

def evaluate_solution(
    routes,
    demand_dict,
    depot_index,
    provider,
    id_to_index,
    traffic_monitor=None,
):
    return sum(
        route_metrics(
            route,
            demand_dict,
            depot_index,
            provider,
            id_to_index,
            traffic_monitor=traffic_monitor,
        )[0]
        for route in routes
    )

def find_best_relocation(
    routes,
    demand_dict,
    depot_index,
    provider,
    id_to_index,
    tabu_set,
    traffic_monitor=None,
):
    best_candidate = None
    best_cost = float("inf")

    for from_idx, route in enumerate(routes):
        if not route["customers"]:
            continue

```

```

for cust_pos, customer_id in enumerate(route["customers"]):
    demand = demand_dict[customer_id]
    for to_idx, target in enumerate(routes):
        if from_idx == to_idx:
            continue

        if target["load"] + demand > target["capacity"]:
            continue

        move = (customer_id, from_idx, to_idx)
        if move in tabu_set:
            continue

        candidate_routes = [
            {
                "vehicle_id": r["vehicle_id"],
                "capacity": r["capacity"],
                "customers": r["customers"].copy(),
                "load": r["load"],
            }
            for r in routes
        ]
        candidate_routes[from_idx]["customers"].pop(cust_pos)
        candidate_routes[from_idx]["load"] -= demand
        candidate_routes[to_idx]["customers"].append(customer_id)
        candidate_routes[to_idx]["load"] += demand

        candidate_cost = evaluate_solution(
            candidate_routes,
            demand_dict,
            depot_index,
            provider,
            id_to_index,
            traffic_monitor=traffic_monitor,
        )

        if candidate_cost < best_cost:
            best_cost = candidate_cost
            best_candidate = (
                candidate_routes,
                move,
                candidate_cost,
            )

if best_candidate is None:
    return None

candidate_routes, move, candidate_cost = best_candidate
updated_loads = [r["load"] for r in candidate_routes]
return candidate_routes, updated_loads, move, candidate_cost

def tabu_search(
    customer_ids,
    demand_dict,
    depot_index,
    provider,
    id_to_index,
    vehicle_list,
    traffic_monitor=None,
    iterations=200,
    tabu_size=100,

```

```

):
    if not customer_ids:
        return [], 0.0
    if not vehicle_list:
        raise ValueError("No vehicles assigned to this warehouse.")

    sorted_vehicles = sorted(vehicle_list, key=lambda v: v["capacity"],
reverse=True)
    current = initial_solution(customer_ids, demand_dict, sorted_vehicles)
    current_cost = evaluate_solution(
        current,
        demand_dict,
        depot_index,
        provider,
        id_to_index,
        traffic_monitor=traffic_monitor,
    )

    best_solution = deepcopy(current)
    best_cost = current_cost

    tabu_queue = deque()
    tabu_set = set()
    current_loads = [r["load"] for r in current]

    for _ in range(iterations):
        neighbor = find_best_relocation(
            current,
            demand_dict,
            depot_index,
            provider,
            id_to_index,
            tabu_set,
            traffic_monitor=traffic_monitor,
        )

        if not neighbor:
            break

        current, current_loads, move, candidate_cost = neighbor

        if candidate_cost < best_cost:
            best_solution = deepcopy(current)
            best_cost = candidate_cost

        tabu_queue.append(move)
        tabu_set.add(move)
        if len(tabu_queue) > tabu_size:
            expired = tabu_queue.popleft()
            tabu_set.discard(expired)

    return best_solution, best_cost

```