Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол № __ від __ грудня 2025р.

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)

«___»_____ 2025 р.

Кваліфікаційна робота
здобувача другого (магістерського) рівня вищої освіти

«МОДЕЛЬ ІНФОРМАЦІЙНОЇ СИСТЕМИ КОНТРОЛЮ ЛОГІСТИКИ ТРАНСПОРТНИХ ЗАСОБІВ»

Спеціальність ***123 – Комп'ютерна інженерія.***
Освітня програма ***Комп'ютерна інженерія***

Виконавець _____ **Давид МУКІЄНКО**
(підпис)

Науковий керівник _____ **Вікторія СТРІЛЕЦЬ**
(підпис)

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до магістерської кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел і трьох додатків. Загальний обсяг роботи складає 104 сторінки, із яких 70 сторінок основної частини з 5 рисунками, 2 таблицями, 38 найменувань списку використаних джерел та чотирма додатками.

Метою дослідження є забезпечення оптимізації маршрутів доставки з урахуванням дорожніх умов та мінімізацію часу і витрат під час доставки товарів через розробку та впровадження інтелектуальної інформаційної системи управління логістикою.

Об'єкт дослідження – об'єктом дослідження є процес управління логістичними маршрутами транспортних засобів у міському середовищі, зокрема у системах доставки товарів від складів до клієнтів.

Предмет дослідження – підходи, методи й алгоритми побудови маршрутів, а також інформаційні моделі логістичних систем.

Проблема, яка вирішується в кваліфікаційній роботі, полягає в тому, щоб скориставшись існуючими алгоритмами маршрутизації та підходами до створення програмного коду мінімізувати час доставки в логістичних системах, його оптимізацію та адаптацію до динамічних умов, запобігти неефективним маршрутам та зменшити витрати на простої транспортних засобів.

Область застосування – розробка інформаційних систем транспортної логістики. Розроблений програмний продукт може широко використовуватися в сфері логістичних компаній та служб доставки.

Ключові слова: *алгоритм Дейкстри, пошук A*, логістика транспортних засобів, задача маршрутизації, OSRM API, Leaflet.js.*

ABSTRACT

The explanatory note to the master's thesis consists of an introduction, three chapters, conclusions, a list of references, and three appendices. The total volume of the work is 104 pages, of which 70 pages are the main part with 5 figures, 2 tables, 38 reference titles, and four appendices.

The aim of the study is to ensure the optimization of delivery routes taking into account road conditions and to minimize time and costs during the delivery of goods through the development and implementation of an intelligent logistics management information system.

The object of the research is the process of managing logistics routes of vehicles in an urban environment, particularly in systems that deliver goods from warehouses to customers.

The subject of the research is the approaches, methods, and algorithms for route construction, as well as information models of logistics systems.

The problem addressed in the qualification work consists in the use of existing routing algorithms and approaches to software code development to minimize delivery time in logistics systems, optimize and adapt it to dynamic conditions, prevent inefficient routes, and reduce idle costs of vehicles.

The area of application is the development of information systems for transport logistics. The developed software product can be widely used in the field of logistics companies and delivery services.

Keywords: *Dijkstra's algorithm, A* search, transport logistics problem, transport routing problem, OSRM API, Leaflet.js.*

ЗМІСТ

ВСТУП

Актуальність роботи. У зв’язку з інтенсивним розвитком міської інфраструктури та постійним зростанням обсягів вантажоперевезень дедалі більш актуальною стає проблема раціональної організації логістичних процесів. Традиційні системи маршрутизації, що використовуються на практиці, не завжди здатні адекватно враховувати динамічні фактори транспортного середовища – такі як робота світлофорів, затори у пікові години, необхідність оперативного об’їзду перешкод чи врахування вартості простою транспортних засобів. У цьому контексті розробка інформаційної системи інтелектуального управління логістикою набуває особливої значущості, оскільки забезпечує підвищення ефективності процесу доставки за рахунок адаптивного вибору оптимальних маршрутів з урахуванням реальних дорожніх умов.

Метою дослідження забезпечення оптимізації маршрутів доставки з урахуванням дорожніх умов та мінімізацію часу і витрат під час доставки товарів через розробку та впровадження інтелектуальної інформаційної системи управління логістикою.

Об’єкт дослідження – процес управління логістичними маршрутами транспортних засобів у міському середовищі, зокрема у системах доставки товарів від складів до клієнтів.

Предмет дослідження – підходи, методи й алгоритми побудови маршрутів, а також інформаційні моделі логістичних систем.

Методи дослідження. Щоб осягнути окреслену ціль, були застосовані наступні методи дослідження::

1. Алгоритмічне моделювання – розробка та впровадження алгоритмів маршрутизації A* та Дейкстри з урахуванням специфіки міської інфраструктури.

2. Об’єктно-орієнтоване програмування – створення модульної архітектури системи з чітким розподілом функціоналу між компонентами

3. Асинхронне програмування – використання Promise та async/await для паралельного виконання запитів до OSRM API та анімації руху машин.

4. Імітаційне моделювання – створення моделі поведінки транспортних засобів при зустрічі з перешкодами (світлофорами).

5. Ітеративна розробка – поетапне вдосконалення системи на основі аналізу результатів тестування.

6. Порівняльний аналіз – оцінка ефективності різних алгоритмів маршрутизації за критеріями часу доставки та уникнення затримок.

7. UI/UX проектування – розробка інтуїтивного інтерфейсу з можливістю управління відображенням інформаційних панелей.

Завдання дослідження:

– виконати аналіз існуючих інформаційних систем логістичного управління та методів маршрутизації транспортних засобів з метою визначення їхніх переваг і недоліків у міських умовах;

– дослідити математичні та алгоритмічні підходи до оптимізації маршрутів доставки, зокрема алгоритми A*, Дейкстри та їх модифікації з урахуванням динамічних дорожніх факторів;

– розробити концептуальну модель інформаційної системи контролю логістики транспортних засобів із можливістю інтеграції з сервісами типу OSRM API для отримання даних про дорожні умови в реальному часі;

– створити алгоритмічну модель інтелектуальної маршрутизації, що забезпечує мінімізацію часу доставки та скорочення простоїв транспортних засобів;

– реалізувати прототип інформаційної системи з використанням об’єктно-орієнтованих і асинхронних технологій програмування, забезпечивши модульність та масштабованість архітектури;

– розробити імітаційну модель поведінки транспортних засобів при зміні дорожніх умов (світлофори, затори, перешкоди) для оцінки ефективності запропонованого підходу;

– провести порівняльний аналіз ефективності реалізованих алгоритмів за критеріями часу доставки, стабільності маршруту та витрат ресурсів;

– розробити інтерфейс користувача з урахуванням принципів UI/UX-дизайну для наочного відображення руху транспортних засобів та інформаційних панелей контролю;

– здійснити тестування та оцінку функціональності розробленої системи в умовах, наближених до реальних, і сформулювати рекомендації щодо її практичного впровадження.

РОЗДІЛ 1
АНАЛІЗ ЗАДАЧ І ТЕХНОЛОГІЙ ТРАНСПОРТНОЇ ЛОГІСТИКИ

Логістика перевезень є важливою частиною сучасної дисципліни, що дозволяє інтегрувати ланцюги постачання через переміщення товару від постачальників до споживачів. Логістика перевезень та перевезення у світлі глобалізації та нові вимоги до швидкості та якості обслуговування досліджуються як важлива економічна проблема стійкості й конкурентоспроможності бізнесу. Підприємства досягають ефективності через нові речі в економіці, в першу чергу в математичній оптимізації, технології, що взаємодіють в обслуговуванні, та сфері гнучких алгоритмів. Тому сучасні способи перевезень, які обслуговують транспортну логістику, дозволяють досягати значної економії в підприємствах та в обслуговуванні підвищувати якість.

1.1 Задачі транспортної логістики

Транспортна логістика включає в себе безліч підходів до оптимізації з унікальними характеристиками, обмеженнями, критеріями оптимальності для кожної з таких задач. Управління транспортними процесами оптимально, коли є знання з теорії й практики математичних моделей зазначених задач і способів їх реалізації.

Завдання вітчизняної транспортної логістики і у світлі з погляду зарубіжних країн має різну класифікацію: за складом, за типом обмежень, за характером цілей, за формою, топологією транспортної мережі. Основними з таких задач є:

Завданнями маршрутизації транспортних засобів, що обслуговують клієнтську базу, є визначення й обчислення оптимального маршруту для усіх

одиниць складу легкового каравану, що є узагальненням класичної задачі комівояжера й має велику

Завданнями оптимізації складу та розподілу є обраний максимально вигідного розташування логістичних об'єктів у просторі, враховуючи транспортні витрати, час доставки та вимоги клієнтів.

Завданнями управління запасами із транспортуванням логістики є об'єднання визначення рівня запасів на складах та модуля планування транспортних переміщень вантажу, що в кінцевому результаті дає можливість досягти глобальної оптимізації Supply Chain Management (SCM) [1, 2].

До завдань Supply Chain - розподіл, планування та управління у сфері логістики, оптимального розподілу одиниць товару визначеної у наявності, постачальника на одиницях споживача, мінімум транспортних витрат, та дотримання заданих обмежень для величини пропускної спроможності пустої логістичної мережі.

Задача комівояжера (Traveling Salesman Problem) це одна з найвідоміших та основних задач з комбінаторної оптимізації в логістиці засобів переміщення. Сформульована в 1930 р., має велику актуальність у створенні маршрутів, та створенні інших задач що стосуються оптимальності в рішенні.

Математична постановка задачі

В нас є наступний граф $G(V, E)$, де $V = (v_1, v_2, \dots, v_n)$ – наші вершини, E – задані ребра, задіяні для того, аби вершини були з'єднанні, та c_{ij} – відстань переміщення з міста i до міста j. Задача потребує знаходження гамільтонів цикл найменшої вартості, а саме маршрут, який відвідує кожне місто рівно один раз і повертається до початкового міста [20, 34].

Задачу можна сформулювати як задачу цілочисельного лінійного програмування. Введемо бінарну змінну:

$$x_{ij} = \begin{cases} 1, \text{якщо ребро } (i,j) \text{ входить в оптимальний маршрут,} \\ 0, \text{в іншому випадку.} \end{cases}$$

Тоді задача комівояжера формулюється наступним чином:

$$\min Z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij},$$

Дано умови:

– кожне місто відвідується рівно один раз $\sum_{j=1}^{n} x_{ij} = 1, \forall i \in V$;

– з кожного міста виходить рівно одне ребро $\sum_{i=1}^{m} x_{ij} = 1, \forall j \in V$;

– відсутні підцикли $\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \forall S \subset V, 2 \leq |S| \leq n - 1$;

– $x_{ij} \in \{0,1\}, \forall i, j \in V$.

Умова відсутності підциклів, вигадана Джонсоном, має дуже незамінне значення в забезпеченні розв'язку що утворює моноцикл, а не кілька окремих циклів.

Одна з формулювань задачі комівояжера обумовлена необхідністю обслуговування множини клієнтів, використовуючи флот транспортних засобів з обмеженою місткістю. Цю задачу називають проблемою маршрутизації транспортних засобів (VRP, Vehicle Routing Problem) [13].

Математичне формулювання

Розглянемо транспортну мережу, представлену графом $G = (V, E)$, де $V = \{v_0, v_1, \dots, v_n\}$ – множина вершин, причому v_0 – депо, а $v_1, \dots, v_n$ – клієнти. Кожен клієнт i має попит q_i, а кожен транспортний засіб має вантажопідйомність Q. Вартість переміщення між вершинами i та j позначається як c_{ij}.

Введемо бінарні змінні:

$$x_{ijk} = \begin{cases} 1, \text{якщо траспортний засіб } k \text{ переміщується з вершини } i \text{ до вершини } j, \\ 0, \text{в іншому випадку.} \end{cases}$$

Базова формулювання задачі VRP має вигляд:

$$\min Z = \sum_{k=1}^{K} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ijk},$$

за умов:

– кожен клієнт обслуговується рівно один раз $\sum_{k=1}^{K} \sum_{j=0}^{n} x_{ijk} = 1, \forall i \in V\{0\}$;

- кожен транспортний засіб виїжджає з депо $\sum_{j=0}^{n} x_{0jk} = 1, \forall k = 1, \dots, K$;
- збереження потоку $\sum_{i=0}^{n} x_{ijk} - \sum_{j=0}^{n} x_{jik} = 0, \forall i \in V, \forall k$;
- обмеження вантажопідйомності $\sum_{i=1}^{n} q_i \sum_{j=0}^{n} x_{ijk} \leq Q, \forall k$;
- $x_{ijk} \in \{0,1\}, \forall i, j \in V, k$.

Існує кілька типів проблем маршрутизації, враховуючи специфіку реальних логістичних систем.

У випадку VRP з часовими вікнами (VRPTW) існують обмеження на час обслуговування клієнтів. Є часовий проміжок, в межах якого кожен клієнт повинен бути обслугований.

Є випадки у VRP з підбором і доставкою (VRPSPD), коли транспортні засоби повинні доставляти товари до клієнтів і також забирати товари від них.

У випадку VRP з кількома депо (MDVRP) існує кілька розподільчих центрів, з яких можуть починатися маршрути.

У задачі VRP з перевантаженнями (VRP з перевантаженнями) дозволено передавати товари між транспортними засобами в призначених точках перевантаження. Це дуже актуально для мереж дорогоцінних роздрібних продавців.

Транспортне завдання - класичне лінійне завдання програмування, яке полягає в тому, щоб визначити оптимальний план транспортування для одного вантажу від товарів до споживання при мінімізовані загальні витрати на перевезення.

Математична постановка задачі

Нехай маємо m пунктів постачання $S_1, S_2, \dots, S_m$ з обсягами продукції $a_1, a_2, \dots, a_m$ відповідно, та n пунктів споживання $D_1, D_2, \dots, D_n$ з потребами $b_1, b_2, \dots, b_n$. Вартість транспортування одиниці вантажу з пункту i до пункту j дорівнює c_{ij}.

Позначимо через x_{ij} кількість одиниць вантажу, що транспортується з пункту постачання i до пункту споживання j. Тоді математична модель транспортної задачі має вигляд:

$$\min Z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij},$$

за умов:

– обмеження постачання $\sum_{j=1}^{n} x_{ij} = a_i, i = 1,2, \dots, m$;

– обмеження попиту $\sum_{i=1}^{m} x_{ij} = b_j, j = 1,2, \dots, n$;

– $x_{ij} \geq 0, \forall i, j$.

Транспортна задача називається *збалансованою*, якщо загальна пропозиція дорівнює загальному попиту: $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$.

Якщо речі вийдуть з удару, просто підкинуть якісь зроблені джерела або цілі, коли попит перемог у товарних автоматах, ми побачимо деякі сумнівні джерела, що з'являються з чайовими та трюками, або навіть безкоштовні квитки на автомобілі в іншому випадку, це не справжня угода.

Функціональне місце призначення об'єкту ((Facility Location Problem) існує для вибору місць з можливість їхнього оптимального віддалення складам, на роздільних центрах або в інших логістичних об'єктах з метою мінімізації транспортних витрат та забезпечення ефективного обслуговування клієнтів)

Формулювання p-медіанної задачі

Нехай задано множину потенційних місць розташування об'єктів та множину клієнтів. Знайдіть місця для об'єктів, які являлося би оптимальними (тобто відстань від клієнтів до об'єктів була мінімальною), з метою мінімізації сумарної вагової відстані від клієнтів до найближчих об'єктів.

Введемо бінарні змінні: $y_i = \begin{cases} 1, \text{якщо об}'\text{єкт розміщується в точці } i, \\ 0, \text{в іншому випадку} \end{cases}$,

$x_i = \begin{cases} 1, \text{якщо клієнт } j \text{ обслуговується об}'\text{єктом в точці } i, \\ 0, \text{в іншому випадку} \end{cases}$.

Математична модель p-медіанної задачі:

$$\min Z = \sum_{i \in I} \sum_{j \in J} d_{ij} h_j x_{ij},$$

за умов:

кожен клієнт обслуговується одним об’єктом $\sum_{i \in I} x_{ij} = 1, \forall j \in J$;

клієнт може обслуговуватися тільки існуючим об’єктом $x_{ij} \leq y_i, \forall i \in I, \forall j \in J$;

розміщується рівно p об’єктів $\sum_{i \in I} y_i = p$;

$x_{ij}, y_i \in \{0,1\}, \forall i, j,$

де d_{ij} – відстань між точкою та клієнтом, h_j – вага (попит) клієнта.

Існує кілька варіантів проводити місце великої трафік порушень.

Задача p-центру (minimax)

є прямою до мінімізації максимальної відстані від клієнтів до найближчого об'єкта, що особливо важливо для екстрених служб.

Задача максимального покриття

визначає розміщення об'єктів таким чином, щоб максимізувати кількість клієнтів, які знаходяться в межах заданого радіусу обслуговування.

Задача з обмеженнями місткості

показує максимальну ємність кожного об'єкта, який відображає реальні обмеження складських приміщень.

Завдання з управління запасами

(Inventory Routing Problem) з маршрутизацією є способом змішаного вирішення питання про рівень запасу на складах клієнтів та плануванням маршрутів доставки в горизонті планування. Інтегрована ця задача тільки і дозволяє досягти суттєвої економічної вигоди у зіставленні з окремою реалізацією задач управління запасами та маршрутизації.

Математична постановка задачі

Розглянемо систему з одним постачальником та n клієнтів протягом горизонту планування T періодів. Нехай I_j^t – рівень запасів клієнта j на початок

періоду t, U_j – максимальна місткість складу клієнта j, d_j^t – попит клієнта j в періоді t, Q – місткість транспортного засобу.

Змінні рішення включають бінарну змінну x_{ij}^t, що дорівнює 1, якщо транспортний засіб переміщується від клієнта i до клієнта j в періоді t, та змінну q_j^t, що визначає обсяг доставки клієнту j в періоді t.

Цільова функція включає витрати на транспортування та зберігання:

$$\min Z = \sum_{t=1}^{T}\sum_{i=0}^{n}\sum_{j=0}^{n} c_{ij}x_{ij}^t + \sum_{t=1}^{T}\sum_{j=1}^{n} h_j I_j^t,$$

де c_{ij} – вартість переміщення між клієнтами i та j, h_j – вартість зберігання одиниці продукції у клієнта j.

Основні обмеження включають динаміку запасів, місткість транспортних засобів та складів, а також умови маршрутизації. Рівень запасів визначається рекурентно:

$$I_j^{t+1} = I_j^t + q_j^t - d_j^t, \forall j, t.$$

1.2 Класичні методи розв'язання задач логістики

Оптимізація транспортної логістики потребує залучення широкого спектру математичних методів, починаючи від чітких алгоритмічних рішень і закінчуючи евристичними підходами.

Обрання відповідного методу обумовлюється обсягом поставленого завдання, її внутрішньою будовою та очікуваною точністю кінцевого рішення.

1.2.1 Методи лінійного програмування

Лінійне програмування слугує наріжним каменем у процесі розв'язання численних проблем транспортної логістики, зокрема, у класичній транспортній задачі та при оптимізації розподілу ресурсів.

У всесвіті математичної оптимізації одним з основоположників і визнаним гуру є Джордж Данціг, який у 1947 році запропонував нову структуру,

математичний алгоритм і методику. Йому властива концепція безперервної, ітераційної, повільної, крок за кроком, подорожі крізь з пожиткова багатогранна призма всіх можливих кутових рішень з величезними просторами, прагнучи зрештою, до ідеального - оптимізації цільової функції.

Шлях Симплекс-методу розгортається у таких фундаментальних етапах:

1. Пошук відправної точки: Насамперед необхідно знайти початковий базисний допустимий розв'язок. Для цього нерідко вдаються до винахідливих технік, таких як введення штучних змінних або ж використання двофазного симплекс-методу.

2. Діагностика досконалості: На цьому етапі відбувається ретельна перевірка, чи є поточне рішення оптимальним. Обчислюються спеціальні оцінки для небазисних змінних. Якщо всі ці оцінки виявляються позитивними або нульовими (для задачі мінімізації), то ми досягли вершини – поточний розв'язок є ідеальним.

3. Вибір нового гравця: Якщо оптимальності ще не досягнуто, обирається "кандидат" на входження в базис. Для задачі мінімізації це буде та небазисна змінна, чия оцінка є найбільш від'ємною, вказуючи на найбільший потенціал для покращення.

4. Визначення "відступаючого": Наступний крок – вирішити, яка базисна змінна має поступитися місцем. Тут застосовується критерій мінімального відношення, що допомагає ідентифікувати ту змінну, яка першою досягне нульового значення.

5. Трансформація базису: Коли "вхідний" і "вихідний" гравці визначені, виконується симплекс-перетворення. Це, по суті, перебудова системи рівнянь, що веде нас до абсолютно нового базисного розв'язку.

Цей циклічний танець триває, ітерації продовжуються, доки система не досягне свого фінального аккорду – оптимального рішення, або ж доки не буде

виявлено, що цільова функція є необмеженою, тобто її можна покращувати до безкінечності.

На основі цього геніального підходу виникла його філігранна адаптація – **Транспортний симплекс-метод**. Це спеціалізована, удосконалена версія класичної методики, тонко налаштована для швидкого та ефективного вирішення транспортних задач. Завдяки унікальній архітектурі самої задачі, цей метод використовує зручний табличний формат представлення та оптимізовані розрахунки, що значно прискорює процес.

Ключові фази транспортного симплекс-методу починаються з:

1. Формування первинного, базового плану поставок: для визначення стартового базисного допустимого розв'язку тут використовуються різноманітні стратегії:

Метод північно-західного кута: Він, наче дитяча забавка, починає заповнювати таблицю з лівого верхнього кута, послідовно задовольняючи всі вимоги щодо постачання та попиту. Хоча його простота очевидна, він часто приводить до маршрутів, які далекі від ідеалу, адже не враховує витрати.

Стратегія найменшої вартості (Метод мінімального елемента):

Цей підхід робить крок уперед. На кожній ітерації він обирає клітинку з найнижчою ціною транспортування. Ця деталізація, в свою чергу, якісно переосмислює з попереднього методу, що зумовлював ринкові стратегії та пошук алгоритмів.

Метод Фогеля: серед стандартних методів пошукових стратегій стартового плану, що активно використовується в оптимізації, без усякого сумніву, беззаперечним. Він дозволяє визначити потенційні "штрафи" - різницю між двома, в кожному прямокутному масиві, з двох, найменшими. Потім, з найвищим є "штраф" рядків, зосереджується на прямокутному масиві, та вартісному з

максимально меншиш, що, безумовно, забезпечує найрезультативніший і безумовно найвражаючі стартові позиції.

2. Перевірка оптимальності за допомогою методу потенціалів. Для кожного постачальника i визначається потенціал u_i, а для кожного споживача j – потенціал v_j. Для базисних змінних виконується умова $u_i + v_j = c_{ij}$.

Обчислення оцінок для небазисних змінних: $w_{ij} = u_i + v_j - c_{ij}$. Якщо всі оцінки невід'ємні, розв'язок є оптимальним.

3. Поліпшення розв'язання за допомогою методу ітерацій. Якщо є негативні оцінки, в основу вводиться небазовий змінний з найнижчою оцінкою. Будується замкнене коло, і виконується циклічне перерозподілення вантажів.

Угорський алгоритм є високоефективним методом розв'язання задач призначення і був створений Гарольдом Куном у 1955 році, ґрунтуючись на працях угорських математиків Деніса Кьоніга та Яноша Егерварі.

Задача призначення є підмножиною транспортної задачі, яка виникає, коли кількість робіт дорівнює кількості агентів, і кожен агент може бути призначений лише на одну роботу.

Математичне формулювання:

$$\min Z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij},$$

за умов:

1) кожен виконавець призначається лише на одну роботу $\sum_{j=1}^{n} x_{ij} = 1, \forall i$;

2) кожна робота виконується лише одним виконавцем $\sum_{i=1}^{n} x_{ij} = 1, \forall j$;

3) $x_{ij} \in \{0,1\}, \forall i, j$.

Алгоритм угорського методу:

1. Редукція рядків матриці вартостей. Від кожного елемента рядка віднімається мінімальний елемент цього рядка.

2. Редукція стовпців. Від кожного елемента стовпця віднімається мінімальний елемент цього стовпця.
3. Покриття нулів мінімальною кількістю ліній. Використовуються горизонтальні та вертикальні лінії для покриття всіх нулів у матриці.
4. Перевірка оптимальності. Якщо кількість ліній дорівнює розмірності матриці n то можна зробити оптимальне призначення. В іншому випадку переходимо до наступного кроку.
5. Коригування матриці. Визначають найменший елемент, який не перекритий жодною лінією. Цей елемент віднімають від усіх інших непокритих значень, а до тих елементів, які перетинаються двома лініями, додають його.
6. Повернення до попередніх дій. Потім знову повторюють з третього кроку, і так доки не буде знайдено найкраще можливе рішення.
7. Формування кінцевого призначення. З усіх отриманих нулів обирають такі, щоб у кожному рядку та кожному стовпці залишався лише один потрібний нуль.

Складність угорського алгоритму становить $O(n^3)$, що робить його підходящим для розв'язання задач середнього розміру.

1.2.2 Методи точного розв'язання для комбінаторних задач

Складні комбінаторні задачі, такі як задача комівояжера та задача маршрутизації транспортних засобів, протягом багатьох років вирішувалися за допомогою спеціальних методів, що дозволяють швидко знаходити достатні рішення.

Метод гілок і меж – це метод розв'язання задач цілочисельного програмування та комбінаторної оптимізації, створений Айлзою Ленд та Елісон Дойг у 1960 році. Це найпотужніший інструмент, який коли-небудь використовувався для розв'язання NP-складних задач оптимізації протягом багатьох років.

Основна мета цього методу полягає в систематичному розділенні всіх можливих розв'язків задачі на менші підгрупи (гілки) та оцінці (межі) для виключення тих підгруп, які не можуть дати оптимального розв'язку.

Існує три фундаментальні кроки, які передбачає робота алгоритму гілок та меж.

1. Розгалуження - це процес, який розділяє всі потенційні розв'язки задачі на менші набори розв'язків. У задачах маршрутизації це може бути вибір сегмента маршруту для використання або втрати.
2. Обмеження створює верхню та нижню межі вартості рішення для кожної групи. Нижні межі зазвичай визначаються шляхом послаблення інших аспектів задачі та лінійної релаксації у випадку задач цілочисельного програмування.
3. Правило обрізання дозволяє нам видаляти групи, де нижня межа вища за найкраще рішення на даний момент. Це дуже швидко зменшує обсяг обчислень.

Обхід рішення полягає між дослідженням груп у певному порядку.

Поширеними методами є метод «спочатку в глибину», метод «спочатку в ширину» та метод «спочатку найкращого» [5, 6, 16].

Метод гілок та меж для вирішення задачі комівояжера.

Релаксація за пунктом призначення дозволяє швидко оцінити нижню межу з досить точними оцінками.

У кожному вузлі дерева рішень розробляється задача призначення, рішення якої може включати підцикли. Метод гілок та меж застосовується для зменшення кількості цілей розташування для визначення мінімальних меж задачі комівояжера.

Коли підцикли доступні, операції розгалуження виконуються шляхом включення або виключення певної частини шляху.

Метод гілок та меж може бути використаний для дуже швидкого розв'язання задачі комівояжера з десятками тисяч міст та для розв'язання задач з мільйонами міст з відносно високою точністю.

1.2.3 Евристичні та метаевристичні методи

Евристичні та метаевристичні методи зазвичай застосовуються у випадках, коли проблема має багатовимірність, а точні методи непрактичні через обчислювальну складність, і вони забезпечують приблизні рішення за прийнятний час. Генетичні алгоритми, мурашині алгоритми, алгоритми рою частинок та пошук із забороною є такими методами.

Генетичні алгоритми відносяться до стохастичних методів оптимізації популяції, які імітують природний відбір та еволюцію. Проблеми транспортної логістики вирішувалися за допомогою генетичних алгоритмів, концепції, вперше запропонованої Джоном Голландом у 1970-х роках.

Основні операції генетичного алгоритму:

1. Розв'язки представлені у вигляді хромосом. Хромосома може бути заснована на порядку відвідувань клієнтів у задачі маршрутизації.

2. Створюється початкова популяція. Створюється випадковий або евристичний набір розв'язків.

3. Пристосованість кожної особини встановлюється на основі значення цільової функції.

4. Обираються батьки потомства. Схрещування між методами відбору, наприклад, турнір або рулетка.

5. Кросовер – це злиття двох геномів, яке породжує потомство. У разі проблем маршрутизації використовуються спеціальні оператори перехресного шунта, один з яких – перехресний шунт.

6. Генетичний алгоритм проводить кросовер генетичним матеріалом двох батьків, щоб створювати нащадок. Для задач з маршрутизацією сітки

використовують оптимізовані оператори, такі як порядковий кросовер або частково відображений кросовер. Мутація вносить випадкові зміни у хромосоми, щоб зберегти генетичне різноманіття. Наприклад, при задачі маршрутузації, це може лягати у двох містах або в інверсії частин шляху.

7. Створення нової популяції з нащадків, або нещодавніх, а також можливо часткової попередньої популяції

8. Повторення процесу до задовільної кількості поколінь або доти, доки не буде знайдено точку зупинки

Генетичні алгоритми ефективні для більш складних завдань з найбільшим пошуковим простором але можуть потребувати налаштування операційних параметрів для виготовлення.

Алгоритм мурашиних колоній (ACO) вперше був запропонований Марко Доріго в 1992 році, є метаевристикой, наслідуючи поведінку мурашок при пошуку шляху від мурашника до джерела їжі. Ця техніка є дуже ефективною для рішення задач маршрутизації, таких як комівояжер і VRP.

Етапи роботи алгоритму:

1. Ініціалізація феромонного рівня на ребрах графа і побудова розв’язків штучними мурашками. На початку роботи рівні феромонів рівномірно розподіляються. Потім кожна мурашка обирає випадково вершину, з якої вона почне побудовує власний маршрут, вибираючи наступну вершину ймовірнісно відносно рівня феромонів із-за неї та евристичної інформації, наприклад, відстані.

Ймовірність переходу мурашки k з вершини i до вершини j визначається формулою:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}},$$

де τ_{ij} – рівень феромону на ребрі (i, j), $\eta_{ij} = 1/d_{ij}$ – евристична інформація (зазвичай обернена відстань), α та β – параметри, що визначають відносну важливість феромону та евристичної інформації, N_i^k – множина допустимих вершин для мурашки k.

Оновлення феромону. Після того, як всі мурашки побудували свої маршрути, рівні феромону оновлюються. Феромон випаровується на всіх ребрах:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij},$$

де ρ – коефіцієнт випаровування ($0 < \rho < 1$).

2. Потім додається феромон на ребра, які входять у знайдені маршрути:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_k \Delta\tau_{ij}^{k},$$

де $\Delta\tau_{ij}^{k}$ – кількість феромону, що відкладається мурашкою *k* на ребрі (*i*,*j*), зазвичай пропорційна якості знайденого розв'язку.

3. Ітеративний процес триває до моменту виконання заданих критеріїв припинення обчислень.

Існують численні модифікації алгоритму мурашиних колоній (ACO), зокрема Ant System (AS), Max-Min Ant System (MMAS) та Ant Colony System (ACS). Ці варіанти диференціюються головним чином стратегіями оновлення феромону та унікальними механізмами, що забезпечують оптимальне співвідношення між дослідженням (експлорацією) та використанням накопичених знань (експлуатацією) пошукового простору [11].

Алгоритм рою частинок (Particle Swarm Optimization, PSO), запропонований Кеннеді та Еберхартом у 1995 році, є еволюційним алгоритмом, який моделює колективну динаміку соціальної поведінки зграй птахів або косяків

риб. Цей метод демонструє значну ефективність для вирішення задач неперервної оптимізації і успішно адаптований для дискретних оптимізаційних задач, зокрема у сфері логістики.

У парадигмі PSO кожна частинка інтерпретується як гіпотетичне рішення, що переміщується в багатовимірному просторі пошуку з певною векторною швидкістю. Координати та вектор швидкості кожної частинки динамічно коригуються з урахуванням її індивідуального найкращого знайденого положення (що репрезентує особистий досвід) та глобального оптимуму, виявленого усією сукупністю частинок (що відображає колективний досвід).

Оновлення швидкості частинки j відбувається за формулою:

$$v_j^{t+1} = wv_j^t + c_1 rand_1 \odot \left(x_{PB,j} - x_j^t\right) + c_2 rand_2 \odot \left(x_{GB} - x_j^t\right),$$

де w – коефіцієнт інерції, c_1 та c_2 – константи прискорення (особистий та соціальний компоненти), $rand_1$ та $rand_2$ – вектори випадкових чисел в інтервалі, $x_{PB,j}$ – найкраще положення частинки j, x_{GB} – глобально найкраще положення.

Положення частинки оновлюється як: $x_j^{t+1} = x_j^t + v_j^{t+1}$.

PSO відомий швидкою збіжністю, але може потрапляти в локальні оптимуми. Для подолання цього недоліку розроблено гібридні алгоритми, що поєднують PSO з генетичними алгоритмами або іншими методами.

1.3 Інформаційні технології транспортної логістики

Інформаційна технологія є потрібною для сучасного управління трафіком для автоматизованих процесів, підвищення прозорості ланцюга постачання та прийняття свідомого рішення на основі даних про реальний час.

1.3.1 Системи управління транспортуванням

Система управління транспортуванням (Transportation Management System, TMS) — це програмне рішення, яке призначене для планування, виконання та вдосконалення процесу фізичного переміщення товарів. Вона є важливим

елементом сучасної логістичної інфраструктури, надаючи видимість і контроль над усіма аспектами транспортних операцій.

Ключовою функцією TMS є динамічне планування маршрутів. Завдяки інтеграції з ERP-системами (Enterprise Resource Planning), CRM (Customer Relationship Management) та WMS (Warehouse Management Systems), система пропонує всебічний огляд всіх ланок постачання в режимі реального часу. Це дозволяє підприємствам обирати найбільш ефективні способи доставки, найкращих перевізників і оптимальні маршрути на основі актуальних даних про тип продукції, витрати, відстань і трафік.

Контроль за виконанням транспортних операцій здійснюється шляхом моніторингу вантажів у реальному часі та взаємодії між перевізниками, складами, дистриб'юторами й клієнтами. Сучасні TMS-системи забезпечують можливість управлінні складними міжнародними перевезеннями включно з імпортною документацією.

Аналітичні функції й звітність також відіграють важливу роль у TMS. Система накопичує всі логістичні дані й аналізує їх для збільшення прозорості маршрутів, оцінки роботи перевізників і контролю витрат. Крім того, вона може прогнозувати тенденції на майбутнє для підвищення бізнес-продуктивності та оперативності.

Управління вибором й оцінкою перевізників полягає в моніторингу контрактів та забезпеченні відповідності нормативним вимогам. Аудит вантажоперевезень й автоматизація платіжних процесів спрощують процедуру переговору рахунків перевозок.

Впровадження системи TMS приносить безліч вигод:

- Збільшення оперативної ефективності завдяки автоматизації рутинних завдань скорочує часовий ресурс на виконання щоденних задач і знижує ризик помилок.

- Скорочення транспортних витрат на 5-30% засобами оптимізації маршрутів і кращого вибору операторів значно економлять кошти компаній.

- Підвищена видимість ланцюга постачань через миттєве відстежування вантажу дозволяє швидко реагувати на виникаючі проблеми.

- Клієнтський сервіс поліпшується з точними прогнозами термінів доставлення та регулярним оновленням статусу замовлень.

- Гнучкість систем TMC відповідає зміненим потребам компанії по мірці збільшеного обсягу навантажень чи більш складних логістичних сценаріїв.

Проте впровадження TMC супроводжується деякими викликами:

- Інтеграція існуючих платформ може бути комплексною задачею і вимагати творчого підходу до співпраці із IT-відділами для гарантування плавного обміну даними.

- Опір технічному прогресу серед працівників зі звичайною моделлю можуть затримати перехід до нової системи; необхідна серйозна робота з навчання персоналу щодо переваг нової технології.

- Первісні інвестиційні витрати могуть бути суттєвими особливо для менших підприємств; все ж таки довгострокові результати часто перекривають стартові вкладення.

- Економічна нестабільність через COVID-19 може стати причиною невизначеності при проникненню даних рішень у різноманіття виробничих успіхтів організацій;

- Зауваги пов’язані зі створеним дистанційним форматом роботи – питання доступності ресурсів і обладнання можуть виявитися критично важливими протягом впровадження нового ПО;

Компанія Techno Softwares служить чудовим прикладом успішного застосування TMS: стикнувшись зі стрімким ростом витрат на транспортування та неефективностю використовуючи правильну маршрутизацію вони адаптували свою систему керуючи своїми специфікаціями результат сфер великих

покращених показники були очевидни вже за кілька місяців після реалізації проектуванні здорово зміцнивши позиціонування фахових методик декларуючи сили тотальної опуклості ТТС діяльністю приростили абсолютно задругоївани питання досвіду щоб прямолінійно вирішити якісь затриманні неполадки що виходять під час доставки оскільки почало вище здаватись падаючо просто забавляючи заможным онлян сайт тощо [9 ,16 ,38].

1.3.2 Системи управління складом

Система управління складом (Warehouse Management System, WMS) — це програмне забезпечення, що покликане оптимізувати щоденні діяльності на складі (див. рис. 1). Вона координує різноманітні складські процеси, в тому числі моніторинг запасів, обробку замовлень і відвантаження.

Рисунок 1 – Основні операції системи управління складом (WMS)

Контроль запасів є основною функцією WMS. Система постійно відстежує обсяги товарів у режимі реального часу, пропонуючи точну інформацію про кількість та місцезнаходження продуктів. Крім того, WMS спрощує проведення циклічного підрахунку та поповнення асортименту.

Обробка замовлень також вдосконалюється за допомогою WMS, яка робить більш зручними етапи комплектації, упаковки і відправлення. Це сприяє

підвищенню якості й швидкості виконання замовлень. Оптимізація використання склада є ще однією важливою перевагою WMS.

Завдяки раціональному розміщенню ремесел скорочується час переміщень усередині складу.

Управління трудозатратами дає змогу точно контролювати продуктивність працівників і ефективно призначати задачі залежно від їхньої продуктивності та навантаження. Звітування й аналітика формують звіти про операційні процеси на складі й забезпечують інформацією для постійного поліпшення.

TMS (Transportation Management System) та WMS є спеціалізованими програмними рішеннями в області управління ланцюгами постачання, кожне з яких виконує свої власні кола завдань.

TMS орієнтована тільки на контроль транспортних операцій із акцентуванням на зовнішній логістикою поза межами складу; тоді як WMS фокусується на внутрішніх warehouse-операціях, займаючись такими аспектами як ведення обліку запасів и виконання замовлень поряд із просторовою оптимізацією всередині приміщення.

Діяльність системи WMS фінально завершується в межах самого складу: вона регулює внутрішні процедури та ресурси у той час як TMS охоплює сфери зовнішньої логістики та мереж транспортування.

Також ключові показники цілком діаметрально протилежні: для оцінки роботи використовуються точність ведення обліку товарів й якість комплектування запитів у випадку з WMS; натомість TMS вимірює своєчасність доставок коштів і конкурентоспроможність перевізників .

Взаємодія між двома системами з іншими корпоративними платформами такими як ERP створює комплексний інструмент для координації ланцюга поставок.

1.3.3 Системи корпоративного планування ресурсів

Системи ERP (планування ресурсів підприємства) є комплексними бізнес-інструментами, які об'єднують кілька основних систем в одне рішення. У сфері транспортної логістики ERP-системи виконують критично важливу функцію щодо координації фінансових потоків, ланцюгів постачання, управління кадрами та процесів закупок.

SAP ERP для логістики має модульну структуру з адаптованими під галузь конфігураціями для виробництва, роздрібної торгівлі, сфери охорони здоров'я та логістичних послуг. Система сприяє міжфункціональній співпраці шляхом централізації всіх операцій на єдиній платформі.

Основні переваги SAP у логістиці включають:

- Специфічні для сектору модулі, що відповідають потребам виробництва, охорони здоров'я й роздрібного бізнесу.

- Інтеграція можливостей управління ланцюгами постачання і людськими ресурсами об'єднує задачі з управління запасами і планування робочої сили в єдиній системі.

- Поєднані рішення ERP та HR забезпечують цілісність процесів через автоматизацію заробітної плати, врахувань робочого часу та менеджмент пільг.

Недоліки SAP полягають у:

- Високих витратах на впровадження та тривалому періоді налаштування через його складність;

- Крутій кривій навчання, яка передбачає організований тренінг персоналу задля ефективного використання системи;

- Неспішній клієнтській підтримці з довгим часом очікування на вирішення технічних проблем.

Oracle ERP для логістики - це хмарна платформа корпоративного планування ресурсів, яка інтегрує фінансовий монетарний контроль разом із закупками і управлінню проектами в одному місці.

Ключові характеристика Oracle у транспортній логістикою містять:

- Оптимізація маршрутного планування дозволяє знайти найбільш ефективні способи доставки вантажів від простих до складних мультимодальних рішень.

- Автоматизація управлінських циклів транспортування економить час при обробці замовлень і вантаженнях включно зі створенням рахунків за перевезеннями.

- Менеджмент автопарку передбачає спеціалізоване планування для максимальної продуктивності ресурсів поряд із залученням сторонніх перевізників.

- Управління перевізниками допомагає в оцінюванні їх роботи й вибору найкращих партнерів для досягнення економічних вигод і своєчасності доставлення.

Переваги Oracle ERP включають:

1. Зменшення ручної праці завдяки видимості даних.

2. Оптимізацію маршрутного графіка й контролю витрат на навантаження.

3. Потужний захист даних зі стабільною доступністю.

4. Ретельно організоване керування контрактами.

5. Легка інтеграція з провідними бухгалтерськими платформами.

Недоліки Oracle ERP:

1. Потребує специфічних знань для його оптимального застосування.

2. Маємо справу зі складним і ресурсоємким встановленням системи.

1.4 Сучасні технологічні тренди у транспортній логістиці

Сучасна транспортна логістика перебуває у фазі динамічного оновлення під впливом цифрових технологій, які дають змогу підвищувати точність планування, скорочувати витрати та покращувати керованість процесів. Провідну роль у цих змінах нині відіграють системи штучного інтелекту (ШІ) та

машинного навчання (МН), що поступово стають основним інструментом оптимізації логістичних операцій.

Використання аналітичних алгоритмів дає змогу перетворювати великі обсяги історичних і поточних даних на практичні рекомендації. Такі системи допомагають виявляти закономірності, аналізувати попередній досвід і будувати прогнозні сценарії для покращення прийняття управлінських рішень. Компанії впроваджують ці інструменти, щоб вдосконалити довгострокове стратегічне планування, мінімізувати ризики та підвищити ефективність ресурсного використання.

Один із найпоширеніших напрямів використання ШІ — прогнозування попиту. Аналітичні моделі враховують історію продажів, сезонність, тенденції споживання й зовнішні чинники (святкові періоди, стан економіки тощо), що дозволяє підприємствам заздалегідь коригувати обсяги перевезень та складські запаси. За аналітичними оцінками McKinsey, використання ШІ може зменшити похибку прогнозів до 50%, що істотно підвищує ефективність транспортних систем.

Ще одним проривним інструментом стають цифрові двійники — моделі, які відтворюють роботу реальних логістичних процесів у віртуальному середовищі. Їх застосовують для тестування нових стратегій, оцінювання потенційних ризиків або імітації змін у ланцюзі постачання без втручання в реальні операції. Так, компанія DB Schenker завдяки використанню таких рішень змогла підвищити результативність складських процесів майже на третину.

До найвідоміших практичних прикладів застосування ШІ в логістиці належать:

UPS ORION — система динамічної маршрутизації, яка оптимізує шлях водіїв та скорочує споживання пального.

DHL Resilience360 — платформа для прогнозування перебоїв у ланцюгах постачання й автоматичного коригування маршрутів.

Amazon AI Robotics — роботизована система керування складом, що прискорює обробку замовлень і підвищує пропускну здатність.

Паралельно набуває поширення технологія блокчейн, яка відкриває нові можливості для безпеки й прозорості обміну інформацією в логістичних мережах. За прогнозами Market Research Future (MRFR), ринок блокчейн-рішень у галузі постачання демонструватиме середньорічне зростання близько 39% у 2024–2032 роках.

Блокчейн забезпечує цифрову ідентифікацію кожного вантажу, фіксуючи дані щодо його походження, шляхів транспортування, відповідальних сторін та сертифікації. Такий підхід мінімізує ризики підробки записів і спрощує верифікацію даних між учасниками процесу. Крім того, смартконтракти автоматизують виконання домовленостей, знижуючи потребу в ручній перевірці або адміністративному втручанні.

Окремо варто відзначити потенціал блокчейну у сфері екологічної звітності. Завдяки розподіленому реєстру можна контролювати рівень викидів, витрати палива та реалізацію сталих ініціатив, спрямованих на зниження вуглецевого сліду.

Серед прикладів ефективного використання цієї технології:

Maersk спільно з IBM розробили систему TradeLens, що усуває надлишкову документацію та робить ланцюг поставок більш прозорим.

Walmart, використовуючи IBM Food Trust, запровадив блокчейн-контроль за рухом продуктів харчування, що гарантує простежуваність і безпечність продукції.

1.5 Переваги та обмеження логістичних інформаційних систем

Переваги логістичних інформаційних систем

Покращення рівня дієвості логістики переважно досягається шляхом автоматизації рутинних завдань та оптимізації внутрішніх процесів. Завдяки

застосуванню спеціалізованих програмних рішень, підприємства можуть зменшити вплив людського чинника, посилити точність контролю запасів, оптимізувати транспортні шляхи та координувати діяльність складів і логістичних вузлів у реальному часі.

Цифрове спостереження дозволяє досягти прозорості ланцюга постачання та повного нагляду за переміщенням матеріальних потоків. Такий підхід забезпечує компаніям швидку реакцію на зміни чи перешкоди у процесах, а також сприяє ухваленню проактивних керівних рішень. Видимість усіх етапів постачання суттєво підвищує стійкість логістичних систем та зменшує небезпеку неузгодженостей.

Особлива увага приділяється оптимізації рівнів запасів, яка базується на прогнозуванні попиту й аналізі даних про товарообіг. Це дає змогу утримувати запаси в межах економічно обґрунтованих показників, що скорочує витрати на зберігання і запобігає виникненню дефіциту чи надлишку.

Зменшення витрат досягається через більш точне планування маршрутів, раціональне використання транспортних засобів, зменшення кількості помилок і автоматизацію багатьох процесів. Незважаючи на затрати на впровадження таких систем, переваги — збільшена продуктивність, економія ресурсів і задоволеність клієнтів — значно перевищують фінансове навантаження.

Якість обслуговування споживачів суттєво покращується: своєчасна доставка товару, доступність асортименту й точність виконання замовлень формують довір’я з боку споживачів та підвищують їх лояльність. Покращене управління логістичною мережею безпосередньо позначається на репутації компанії у галузі.

Сучасні системи логістики характеризуються гнучкістю й масштабованістю — вони швидко адаптуються до збільшених обсягів роботи

або змін у бізнес-процесах. Це особливо важливо для компаній які розширюють свій бізнес чи виходять на новому ринку.

Однак реалізація логістичного програмного забезпечення стикається із певними викликами:

1. Висока вартість впровадження - це особливо відзначається при великих корпоративних рішеннях (як-от SAP або Oracle).
2. Складності інтеграції із вже існуючими платформами ведуть до необхідності додаткових знань і ресурсозатрат.
3. Тривала навчальна програма для персоналу є вимогою через потребу освоїти новітні компетенції.
4. Залежність від постачальника ПЗ може обмежити можливості щодо його налаштування тут.
5. Питання кібербезпеки тягнуть за собою дотримання норм захисту інформації разом із регулярними аудитами.
6. Необхідна безперервна технічна підтримка , оновлення та адаптація до нових задач бізнесу.

Для малих підприємств зі скромним бюджетом найбільш прийнятними будуть хмарні рішення - вони пропонують нижчий стартовий поріг затрат й простоту в реалізації (наприклад: LogiNext Mile або хмарні TMS-системи).

На противагу цьому великі корпорації зі складною структурою частково вибирають потужні інтегровані платформи такі як Oracle OTM або SAP TM – їх широке поле можливостей конструювання й аналітики містить важливий аспект для кожного випадку деталізованого поперечного аналізуючи специфіку індустрії масштаби підприємництва , бюджет і чинну IT-структуру .

Висновок за розділом 1

У цьому розділі здійснено всебічний аналіз задач, методів і технологій у сфері транспортної логістики. Завдання транспортування характеризуються

значною обчислювальною складністю та відносяться до категорії NP-важких комбінаторних задач оптимізації. Математичні формулювання таких класичних проблем, як задача комівояжера, маршрутизація транспорту, транспортна задача та завдання з розміщення об'єктів, слугують теоретичною основою для створення методів їхнього вирішення.

Розглянуті традиційні методи оптимізації включають симплекс-метод, транспортний симплекс-метод, угорський алгоритм і метод гілок і меж – ці підходи забезпечують точні результати для малих і середніх за розміром задач. Також вивчені евристичні та метаевристичні алгоритми, які здатні знаходити високоякісні наближені рішення за прийнятний часовий проміжок при вирішенні великомасштабних завдань.

Сучасні інформаційні технології — такі як системи управління перевезеннями (TMS), системи керування запасами (WMS), Інтернет речей (IoT), штучний інтелект та блокчейн — змінюють ландшафт транспортної логістики через надання реального часу видимості операцій, автоматизацію процесу й ухвалення рішень на основі даних.

На основі проведеного дослідження визначені ключові **напрямки для подальшого дослідження** у рамках магістерської роботи:

- Проектування архітектури інтегрованої інформаційної системи для транспортної логістики з поєднанням функціональності TMS, WMS і IoT-платформ із метою забезпечення загальної видимості постачання.

- Розробка математичної моделі й алгоритму рішення багатокритеріальної задачі щодо оптимізації транспортних послуг з висвітленням економічних, екологічних та соціальних аспектів.

- Створення прототипу системи підтримки ухвалених рішень у галузі транспортування шляхом використання цифрових двійників і моделювання симуляції.

- Дослідження ефективності різноманітних стратегій інтеграції технології блокчейн для підвищення прозорості та безпеки в логістичних процедурах.

Цілком зазначені завдання окреслюють наступну дослідницьку діяльність у майбутніх частинах магістерської роботи з акцентом на вироблення ефективних рішень для оптимізації процесів в області transport logistics за допомогою сучасних методик та інновацій.

РОЗДІЛ 2
ПРОЄКТУВАННЯ МОДЕЛІ ІНФОРМАЦІЙНОЇ СИСТЕМИ КОНТРОЛЮ ЛОГІСТИКИ ТРАНСПОРТНИХ ЗАСОБІВ

2.1. Аналіз вимог до інформаційної системи

Сучасні логістичні системи потребують інтелектуальних рішень для оптимізації процесів доставки та контролю транспортних засобів. Аналіз вимог до інформаційної системи контролю логістики є критичним етапом проєктування, що визначає функціональні та нефункціональні характеристики майбутньої системи.

Функціональні вимоги визначають, що саме повинна робити система для досягнення поставлених цілей. На основі аналізу сучасних логістичних платформ та потреб користувачів було виділено наступні ключові функціональні вимоги.

Відстеження транспортних засобів у реальному часі. Система повинна забезпечувати безперервний моніторинг поточного місцезнаходження транспортних засобів з точністю до декількох метрів. Це досягається через інтеграцію з GPS-датчиками та технологіями Інтернету речей (IoT). Дані про координати, швидкість руху та напрямок мають оновлюватись з інтервалом не більше однієї секунди для забезпечення актуальності інформації [4, 33].

Розрахунок оптимальних маршрутів. Система має розраховувати найефективніші маршрути між пунктами відправлення та призначення з урахуванням множини факторів. Алгоритми маршрутизації повинні враховувати не лише відстань, але й поточну дорожню ситуацію, наявність світлофорів, обмеження швидкості та інші параметри. Розрахунок маршруту має завершуватись протягом 2-3 секунд для забезпечення оперативності прийняття рішень.

Управління множинними транспортними засобами. Система повинна підтримувати одночасний контроль та координацію роботи декількох

транспортних засобів. Це включає можливість призначення завдань, моніторингу виконання та динамічного перерозподілу ресурсів при виникненні непередбачених ситуацій.

Візуалізація даних на інтерактивній карті. Користувацький інтерфейс має забезпечувати наочне відображення всіх елементів системи на географічній карті. Це включає позиції транспортних засобів, складів, клієнтів, маршрутів та додаткових об'єктів інфраструктури. Інтерфейс має бути інтуїтивно зрозумілим та підтримувати можливість масштабування, переміщення та взаємодії з елементами карти.

Аналіз та уникнення перешкод. Система повинна автоматично ідентифікувати потенційні перешкоди на маршруті, такі як затори, дорожні роботи або світлофори з тривалими циклами очікування. На основі цієї інформації мають розраховуватись альтернативні шляхи для мінімізації часу доставки.

Управління даними про склади та клієнтів. Система має зберігати та обробляти інформацію про розташування складів, доступні товари, адреси клієнтів та історію доставок. Ці дані використовуються для планування маршрутів та оптимізації логістичних процесів.

Моніторинг стану вантажів. Функціональність системи має включати відстеження поточного стану доставок, включаючи інформацію про завантажений товар, етап виконання доставки та очікуваний час прибуття.

Формування звітності та аналітики. Система повинна генерувати статистичні звіти про ефективність роботи транспортних засобів, витрачений час, пройдену відстань, кількість виконаних доставок та інші ключові показники ефективності.

Нефункціональні вимоги визначають якісні характеристики системи та обмеження, за яких вона має функціонувати.

Продуктивність. Система має забезпечувати обробку запитів та оновлення даних з мінімальною затримкою. Час відгуку на користувацькі дії не повинен перевищувати 100 мілісекунд для створення ефекту миттєвої реакції. Розрахунок маршрутів для одного транспортного засобу має завершуватись протягом 2-3 секунд навіть при складній дорожній мережі.

Масштабованість. Архітектура системи повинна підтримувати можливість розширення функціональності та збільшення кількості обслуговуваних транспортних засобів без суттєвого зниження продуктивності. Система має ефективно працювати як з 10, так і з декількома сотнями одночасно активних транспортних засобів.

Надійність. Коефіцієнт доступності системи має становити не менше 99.5%, що відповідає максимальному часу простою близько 1.8 годин на місяць. Система повинна мати механізми резервного копіювання даних та відновлення після збоїв.

Безпека. Система має захищати конфіденційні дані про маршрути, клієнтів та операційну діяльність компанії. Доступ до функцій управління має бути обмежений системою авторизації з підтримкою різних рівнів прав користувачів [24].

Сумісність. Інформаційна система повинна коректно функціонувати в різних веб-браузерах (Chrome, Firefox, Safari, Edge) та на різних пристроях (комп'ютери, планшети, смартфони).

Зручність використання. Інтерфейс системи має бути інтуїтивно зрозумілим для користувачів з різним рівнем технічної підготовки. Ключові функції повинні бути доступні протягом не більше 2-3 кліків миші.

Підтримуваність. Код системи має бути структурованим, добре документованим та відповідати сучасним стандартам програмування для забезпечення можливості подальшого розвитку та внесення змін.

Вимоги до інтеграції з зовнішніми сервісами. Для забезпечення повноцінної функціональності система повинна інтегруватись з зовнішніми API та сервісами.

Інтеграція з картографічними сервісами. Система має використовувати сучасні картографічні API для отримання географічних даних, дорожньої мережі та розрахунку маршрутів. В розробленій системі використовується Open Source Routing Machine (OSRM) API, який забезпечує швидкий та надійний розрахунок маршрутів на основі даних OpenStreetMap [29, 30].

Підтримка різних профілів маршрутизації. OSRM API надає можливість розрахунку маршрутів для різних типів транспорту (автомобіль, велосипед, пішохід), що дозволяє адаптувати систему під різні логістичні потреби.

Протокол обміну даними. Взаємодія з зовнішніми сервісами має здійснюватись через стандартизовані протоколи (HTTP/HTTPS, REST API) для забезпечення сумісності та можливості заміни сервісів при необхідності.

Вимоги до візуалізації даних. Ефективна візуалізація є критичною для розуміння стану логістичної системи та прийняття оперативних рішень.

Інтерактивна карта. Система має використовувати бібліотеку Leaflet.js для створення інтерактивної карти з підтримкою масштабування, переміщення та відображення різних шарів інформації. Карта повинна підтримувати відображення маркерів для складів, клієнтів, транспортних засобів та інших об'єктів з можливістю налаштування їх зовнішнього вигляду.

Відображення маршрутів. Розраховані маршрути мають відображатись на карті у вигляді кольорових ліній (поліліній) з можливістю диференціації різних типів маршрутів. В розробленій системі використовуються різні кольори для маршрутів, розрахованих алгоритмами A* (синій) та Dijkstra (помаранчевий).

Анімація руху транспортних засобів. Система має візуалізувати рух транспортних засобів вздовж розрахованих маршрутів у вигляді плавної анімації.

Швидкість анімації має бути налаштовувано для балансу між реалістичністю та зручністю демонстрації.

Інформаційні панелі. Інтерфейс повинен містити панелі з детальною інформацією про маршрути, включаючи покрокові інструкції, відстань, очікуваний час прибуття та інші метрики. Панелі мають бути інтерактивними з можливістю згортання, розгортання та переміщення.

Легенда та додаткові елементи. Система має включати легенду для пояснення значення різних графічних елементів на карті, а також індикатори стану та повідомлення для користувача.

2.2. Розробка концептуальної моделі системи

Концептуальна модель інформаційної системи визначає загальну структуру, основні компоненти та зв'язки між ними на абстрактному рівні. Вона служить основою для подальшого детального проєктування та реалізації системи [35, 36].

2.2.1. Основні компоненти системи

Розроблена інформаційна система складається з декількох взаємопов'язаних компонентів, кожен з яких виконує специфічні функції в рамках загальної архітектури.

Клієнтська частина (Frontend). Представляє собою веб-додаток, що виконується в браузері користувача та забезпечує графічний інтерфейс для взаємодії з системою. Клієнтська частина відповідає за відображення карти, візуалізацію маршрутів, анімацію руху транспортних засобів та обробку користувацьких дій. Реалізована з використанням HTML5, CSS3 та JavaScript з бібліотекою Leaflet.js для роботи з картами.

Модуль управління панелями (PanelManager). Забезпечує функціональність інтерактивних панелей інтерфейсу, включаючи можливість згортання, розгортання, переміщення та закриття панелей. Клас PanelManager

реалізує патерн "Singleton" для централізованого управління всіма панелями в системі.

Модуль візуалізації карти. Відповідає за ініціалізацію та управління інтерактивною картою на базі Leaflet.js. Модуль забезпечує відображення базового шару карти з використанням тайлів OpenStreetMap, створення та управління маркерами для різних об'єктів (склади, клієнти, транспортні засоби, світлофори), відображення маршрутів у вигляді поліліній та організацію шарів для різних типів даних.

Модуль управління даними. Зберігає та управляє інформацією про основні сутності системи. Дані про склади включають координати розташування, назву та тип товарів, що зберігаються. Інформація про клієнтів містить адреси доставки та ідентифікаційні дані. Система також зберігає дані про транспортні засоби, включаючи їх поточні позиції, статус та призначені завдання.

Модуль маршрутизації. Реалізує алгоритми розрахунку оптимальних маршрутів між заданими точками. В системі реалізовано два підходи до маршрутизації: використання алгоритму A* для розрахунку найкоротшого прямого маршруту та адаптований алгоритм Dijkstra для знаходження альтернативних шляхів з урахуванням уникнення світлофорів. Модуль інтегрується з зовнішнім OSRM API для отримання детальних даних про маршрути.

Модуль аналізу дорожньої ситуації. Відстежує стан дорожньої інфраструктури, включаючи позиції та стани світлофорів. Реалізує логіку визначення близькості транспортного засобу до світлофора, розрахунку впливу світлофорів на маршрут та пошуку альтернативних точок для об'їзду затримок.

Модуль анімації. Забезпечує плавну візуалізацію руху транспортних засобів вздовж розрахованих маршрутів. Використовує API requestAnimationFrame для створення оптимізованої анімації з урахуванням

швидкості руху, затримок на світлофорах та інших факторів. Модуль також оновлює інформаційні вікна (popups) з поточними даними про стан доставки.

Модуль управління світлофорами (Traffic Light System). Симулює роботу світлофорів на дорожній мережі. Реалізує циклічну зміну станів світлофорів між червоним та зеленим сигналами, візуалізацію світлофорів на карті з відповідними іконками та радіусами впливу, а також логіку зупинки транспортних засобів при наближенні до світлофора з червоним сигналом.

2.2.2. Архітектура моделі інформаційної системи

Система побудована на основі клієнт-серверної архітектури з використанням зовнішніх веб-сервісів (рис. 2.1).

Рівень представлення (Presentation Layer). На цьому рівні знаходиться користувацький інтерфейс, реалізований за допомогою веб-технологій. Користувач взаємодіє з системою через веб-браузер, де відображається інтерактивна карта та панелі управління. Рівень представлення відповідає за збір користувацького вводу та відображення результатів обробки даних.

Рівень бізнес-логіки (Application Layer). Містить основні алгоритми та логіку роботи системи, реалізовану на мові JavaScript. Цей рівень відповідає за обробку користувацьких запитів, виконання розрахунків маршрутів, управління станом транспортних засобів та координацію роботи різних модулів системи. Бізнес-логіка реалізує алгоритми A* та адаптований Dijkstra, управління анімацією та обробку подій системи.

Рівень зовнішніх сервісів (External Services Layer). Система інтегрується з OSRM API для отримання детальних даних про маршрути та дорожню мережу. Запити до API виконуються асинхронно з використанням Fetch API, що забезпечує неблокуючу взаємодію та підтримує механізми обробки помилок та таймаутів.

Рівень даних (Data Layer). Містить структури даних для зберігання інформації про склади, клієнтів, транспортні засоби та світлофори. В поточній

реалізації дані зберігаються в пам'яті браузера у вигляді JavaScript об'єктів та масивів. Для більш складних систем цей рівень може бути розширений з використанням баз даних.

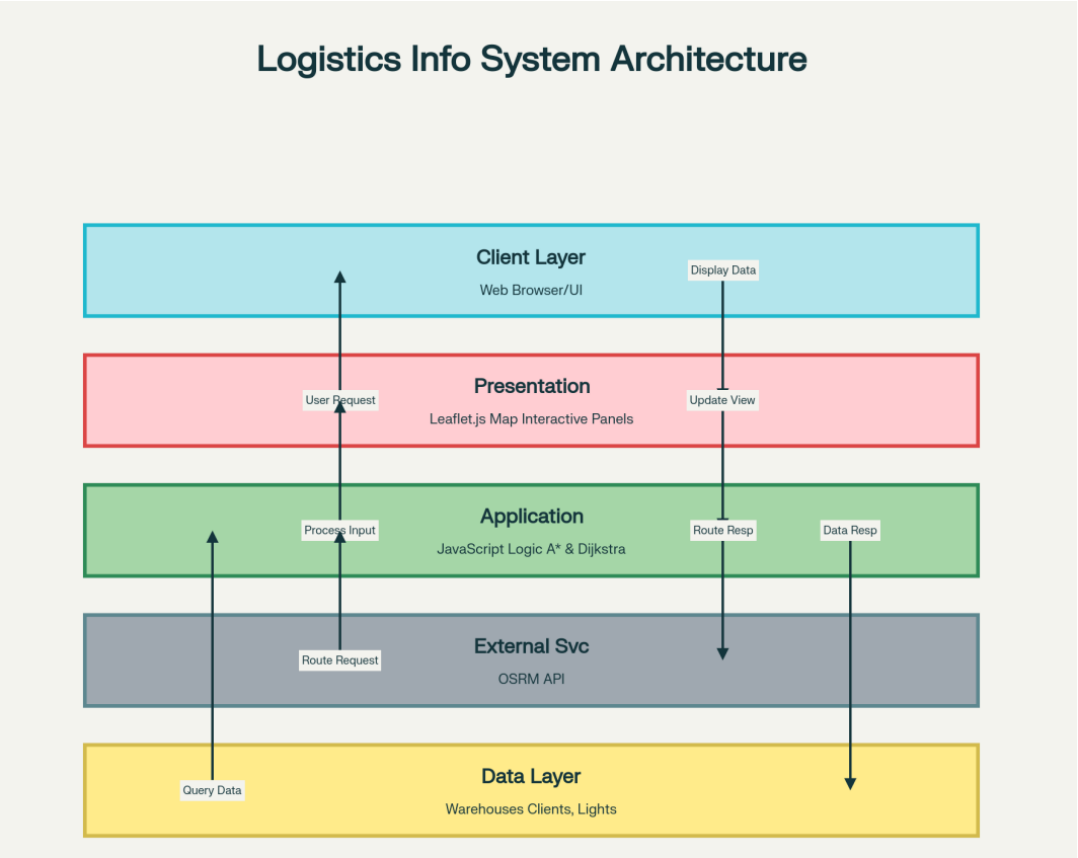

Рисунок 2.1 – Архітектура інформаційної системи контролю логістики транспортних засобів

2.2.3. Модель взаємодії компонентів

Взаємодія між компонентами системи організована за принципом обміну повідомленнями та викликів функцій.

Ініціалізація системи. При завантаженні веб-сторінки виконується послідовність ініціалізацій компонентів. Спочатку створюється екземпляр карти Leaflet з центруванням на заданих координатах та встановленням початкового рівня масштабування. Потім додається базовий шар з тайлами OpenStreetMap [11,

12]. Ініціалізується PanelManager для управління інтерактивними панелями. На карту додаються маркери для складів, клієнтів та світлофорів з відповідними іконками. Створюються екземпляри транспортних засобів з початковими позиціями. Запускається основний цикл виконання доставок через функцію executeInfiniteDeliveries().

Процес виконання доставки. Цикл доставки складається з декількох етапів. Випадковим чином обирається склад-джерело та клієнт-призначення. Паралельно розраховуються два маршрути до складу: прямий маршрут з використанням A* через OSRM API та альтернативний маршрут з використанням адаптованого Dijkstra з уникненням світлофорів. Результати відображаються на карті різними кольорами, а детальна інформація виводиться в інформаційних панелях. Запускається анімація руху обох транспортних засобів до складу. Після прибуття симулюється завантаження товару. Аналогічно розраховуються та виконуються маршрути від складу до клієнта з візуалізацією вантажу, що перевозиться. Після завершення доставки цикл повторюється з новими випадковими точками [8, 9, 10].

Обробка руху та затримок. Під час анімації руху система постійно моніторить позицію транспортного засобу відносно світлофорів. При виявленні близькості до світлофора з червоним сигналом транспортний засіб зупиняється, а до загального часу доставки додається затримка (30 секунд за замовчуванням). Система періодично змінює стани світлофорів, після чого зупинені транспортні засоби продовжують рух.

Множинні альтернативи через OSRM. OSRM API підтримує параметр alternatives, що дозволяє отримати декілька різних маршрутів між заданими точками. Система запитує до 3 альтернативних маршрутів та обирає той, що має найменшу кількість перетинів зі світлофорами [29, 30, 31].

2.2.4. Патерни проєктування

В архітектурі системи використано декілька класичних патернів проєктування для забезпечення модульності та підтримуваності коду [1, 2, 27].

Singleton (Одинак). Використовується для класу PanelManager, оскільки необхідний лише один екземпляр для управління всіма панелями інтерфейсу. Це гарантує централізоване управління станом панелей та уникнення конфліктів при множинних ініціалізаціях.

Observer (Спостерігач). Реалізований через систему подій DOM для обробки користувацьких взаємодій. Панелі управління підписуються на події кліків по кнопках, а карта реагує на події зміни шарів та взаємодії з маркерами.

Strategy (Стратегія). Застосовується для вибору між різними алгоритмами маршрутизації. Система може динамічно обирати між прямим маршрутом (A*) та маршрутом з уникненням перешкод (Dijkstra) в залежності від ситуації.

Factory (Фабрика). Використовується для створення різних типів маркерів на карті з різними іконками для складів, клієнтів, транспортних засобів та світлофорів.

2.3. Математична модель процесів логістики

Математичне моделювання логістичних процесів дозволяє формалізувати задачі оптимізації та розробити ефективні алгоритми їх вирішення [37].

2.3.1. Постановка задачі маршрутизації

Задача маршрутизації транспортних засобів може бути формалізована як задача пошуку оптимального шляху в зваженому графі.

Представлення дорожньої мережі. Дорожня мережа моделюється як орієнтований граф $G=(V,E)$, де $V=\{v_1,v_2,...,v_n\}$ – множина вершин, що представляють перехрестя, склади, клієнтів та інші ключові точки на карті. Множина $E \subseteq V \times V$ – множина ребер, що представляють дорожні сегменти між вершинами. Кожне ребро $e_{ij} \in E$ має вагу w_{ij}, що може представляти відстань між

вершинами v_i та v_j в метрах або кілометрах, час проїзду в секундах або хвилинах, або комбінацію факторів (відстань, час, вартість палива тощо).

Формулювання задачі. Нехай s ∈ V – початкова вершина (склад), а t ∈ V – кінцева вершина (клієнт). Потрібно знайти шлях P = (v_0, v_1, …, v_k), де v_0 = s та v_k = t, який мінімізує функцію вартості:

$$C(P) = \sum_{i=0}^{k-1} w_{v_i v_{i+1}}.$$

Шлях має бути допустимим, тобто для кожної пари послідовних вершин (v_i, v_{i+1}) повинно існувати ребро $e_{v_i v_{i+1}} \in E$.

Додаткові обмеження. У реальних логістичних системах задача часто ускладнюється додатковими обмеженнями:

- часові вікна доставки: доставка має бути здійснена в межах заданого проміжку часу;
- обмеження вантажопідйомності: транспортний засіб може перевозити обмежену кількість товару;
- множинні пункти призначення: необхідно обслужити декілька клієнтів одним маршрутом;
- динамічні перешкоди: стан дорожньої мережі може змінюватись з часом (затори, дорожні роботи).

2.3.2. Алгоритм A* для пошуку найкоротшого шляху

Алгоритм A* є одним з найефективніших методів пошуку найкоротшого шляху в графі завдяки використанню евристичної функції.

Основна ідея алгоритму. A* поєднує переваги алгоритму Dijkstra (гарантована оптимальність) та жадібного пошуку Best-First Search (швидкість за рахунок евристики). Для кожної вершини n в графі алгоритм обчислює функцію оцінки:

$$f(n) = g(n) + h(n)f(n),$$

де $g(n)$ – фактична вартість шляху від початкової вершини s до поточної вершини n, $h(n)$ – евристична оцінка вартості шляху від n до цільової вершини t.

Вибір евристичної функції. Для гарантування оптимальності результату евристична функція $h(n)$ повинна бути допустимою, тобто ніколи не переоцінювати фактичну вартість досягнення цілі. У геопросторових задачах найчастіше використовується евристика на основі прямої відстані між точками.

Для координат на земній поверхні застосовується формула гаверсинусів для обчислення відстані між двома точками з координатами (ϕ_1,λ_1) та (ϕ_2,λ_2):

$$h(n)=2R\arcsin(\sin^2(2\phi_2-\phi_1)+\cos(\phi_1)\cos(\phi_2)\sin^2(2\lambda_2-\lambda_1)),$$

де $R\approx6371$ км – радіус Землі.

Код алгоритму A* наведений у додатку Г.

Часова складність A* залежить від якості евристичної функції. У найгіршому випадку (якщо $h(n)=0$), алгоритм вироджується в Dijkstra з складністю O(|E|+|V|log|V|). З хорошою евристикою складність може бути значно меншою, близько $O(b^d)$, де b – фактор розгалуження, d – глибина розв'язку

Переваги A в логістичних системах.* Алгоритм гарантує знаходження найкоротшого шляху (при допустимій евристиці). Він швидший за Dijkstra завдяки спрямованому пошуку в напрямку цілі. A* підходить для статичних дорожніх мереж, де ваги ребер не змінюються в процесі пошуку [28, 32].

2.3.3. Адаптація алгоритму Dijkstra для уникнення перешкод

Класичний алгоритм Dijkstra знаходить найкоротший шлях від початкової вершини до всіх інших вершин графу.

Базовий алгоритм Dijkstra. Алгоритм підтримує множину вершин S, для яких вже знайдено найкоротший шлях, та множину вершин Q=V\S. Для кожної вершини $v\in$V зберігається поточна оцінка відстані $d[v]$ від початкової вершини s.

Ініціалізація: $d[s]=0$, $d[v]=\infty$ для всіх $v\neq s$, $d[s]=0$, $d[v]=\infty$ для всіх $v=s$.

Основний цикл:

```
while Q is not empty:
    u = vertex in Q with minimum d[u]
    S = S ∪ {u}
    Q = Q \ {u}

    for each neighbor v of u:
        alt = d[u] + weight(u, v)
        if alt < d[v]:
            d[v] = alt
            previous[v] = u
```

Модифікація для уникнення перешкод. У контексті логістичної системи перешкодами виступають світлофори, що створюють затримки в русі. Для врахування цих факторів модифікується функція ваги ребер.

Нехай $T=\{t_1,t_2,...,t_m\}$ – множина світлофорів на дорожній мережі. Кожен світлофор t_i має координати (x_i,y_i), радіус впливу r_i та штраф за очікування p_i.

Модифікована вага ребра e_{uv} обчислюється за формулою:

$$w'_{uv} = w_{uv} + \Sigma \text{ (для всіх } t_i \in T) [\delta (e_{uv}, t_i) \cdot p_i],$$

де $\delta (e_{uv}, t_i)$ – індикаторна функція, що дорівнює 1, якщо ребро проходить через зону впливу світлофора t_i, і 0, якщо ні.

Пошук альтернативних маршрутів. Система використовує стратегію генерації альтернативних точок для пошуку об'їзних шляхів. Для прямолінійного маршруту між точками *s* та *t* генеруються альтернативні точки шляхом зсуву перпендикулярно до основного напрямку:

$$s'=s+d\cdot\vec{n}, t'=t+d\cdot\vec{n} s'=s+d\cdot n, t'=t+d\cdot n,$$

де $\vec{n}$ – одиничний вектор, перпендикулярний до напрямку $s \rightarrow t$, а d – відстань зсуву (наприклад, 45 метрів).

Для кожної пари альтернативних точок (s',t') розраховується маршрут та оцінюється його вартість з урахуванням світлофорів. Обирається маршрут з мінімальною загальною вартістю.

2.3.4. Математична модель затримок на світлофорах

Світлофори створюють випадкові затримки, що впливають на загальний час доставки.

Модель циклу світлофора. Нехай світлофор має два стани: червоний (R) та зелений (G) з періодами TR та TG відповідно. Повний цикл світлофора: Tcycle=TR+TG.

Очікувана затримка. Якщо транспортний засіб прибуває до світлофора у випадковий момент часу, очікувана затримка може бути обчислена як:

$$E[delay] = TR^2 / (2 \cdot Tcycle) = TR^2 / (2 \cdot (TR + TG)).$$

При рівних періодах TR=TG=T:

$$E[delay] = T / 4.$$

Накопичення затримок. При проходженні маршруту з k світлофорами загальна очікувана затримка:

$$E\ [total\ delay] = \Sigma\ E[delay_i] \text{ для } i = 1 \ldots k.$$

У розробленій системі використовується спрощена модель з фіксованою затримкою p=30 секунд для кожного світлофора з червоним сигналом.

2.3.5. Критерії оптимізації

Система може оптимізувати маршрути за різними критеріями.

Мінімізація відстані. Пошук маршруту з мінімальною сумарною довжиною:

$$\min P \sum_{i=0}^{k-1} d(v_i, v_{i+1}),$$

де $d(v_i, v_{i+1})$ – фізична відстань між вершинами.

Мінімізація часу доставки. Пошук маршруту з мінімальним часом проїзду з урахуванням швидкості руху та затримок:

$$\min P \left(\sum_{i=0}^{k-1} \frac{v(v_i, v_{i+1})}{d(v_i, v_{i+1})} + \sum_{t_j \in T(P)} p_j \right),$$

де $v(v_i, v_{i+1})$ – середня швидкість на ділянці, T(P) – множина світлофорів на шляху P, p_j – затримка на світлофорі t_j.

Багатокритеріальна оптимізація. У загальному випадку використовується зважена комбінація критеріїв:

$$\min P\left(\alpha \cdot C_{distance}(P) + \beta \cdot C_{time}(P) + \gamma \cdot C_{fuel}(P)\right),$$

де α,β,γ – вагові коефіцієнти, що визначають важливість кожного критерію.

2.4. Архітектура інформаційної системи

Архітектура інформаційної системи визначає організацію програмних компонентів, їх взаємодію та розподіл функціональності.

2.4.1. Багатошарова архітектура

Система побудована на основі класичної багатошарової (multi-tier) архітектури, що забезпечує модульність, розширюваність та можливість незалежного розвитку окремих компонентів.

Шар представлення (Presentation Layer). Відповідає за взаємодію з користувачем та відображення інформації. Основні компоненти: HTML-структура сторінки з контейнерами для карти та панелей управління, CSS-стилі для візуального оформлення елементів інтерфейсу, JavaScript-код для обробки користувацьких подій та динамічного оновлення інтерфейсу.

Технологічний стек: HTML5 для структури документа, CSS3 для стилізації з використанням Flexbox та Grid для адаптивного макету, JavaScript (ES6+) для інтерактивності, Leaflet.js 1.9.4 для відображення інтерактивних карт.

Шар бізнес-логіки (Business Logic Layer). Містить основну функціональність системи, включаючи алгоритми та логіку прийняття рішень.

Основні модулі: PanelManager для управління інтерактивними панелями, модуль маршрутизації з реалізацією A* та адаптованого Dijkstra, модуль управління транспортними засобами, модуль анімації руху, модуль управління світлофорами та перешкодами [17, 21].

Ключові функції: osrmRoute() для запитів до OSRM API, computeSmartTrafficAvoidance() для розрахунку маршрутів з уникненням

світлофорів, animateMarkerAlong() для анімації руху транспортних засобів, executeInfiniteDeliveries() для управління циклом доставок.

Шар зовнішніх сервісів (External Services Layer). Забезпечує інтеграцію з зовнішніми API та веб-сервісами.

OSRM API: базовий URL: https://router.project-osrm.org, ендпоінт для розрахунку маршрутів: /route/v1/driving/{coordinates}, підтримувані параметри: overview, geometries, steps, annotations, alternatives.

Формат запиту:

```
GET /route/v1/driving/36.233,49.989;36.237,49.993?
    overview=full&
    geometries=geojson&
    steps=true&
    annotations=duration,distance&
    alternatives=3
```

Формат відповіді: JSON з полями code, routes (масив маршрутів), waypoints (масив точок). Кожен маршрут містить: geometry (координати у форматі GeoJSON), distance (відстань в метрах), duration (час в секундах), legs (сегменти маршруту з детальними інструкціями).

OpenStreetMap Tiles: використовується для базового шару карти, URL тайлів: https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png, параметри: {s} – субдомен (a, b, c для розподілу навантаження), {z} – рівень масштабування (0-18), {x}, {y} – координати тайла в системі Web Mercator [26, 27].

Шар даних (Data Layer). Зберігає та управляє даними системи.

Структури даних: масив складів (warehouses) з полями id, pos (координати), label, iconUrl; масив клієнтів (clients) з аналогічною структурою; масив світлофорів (trafficLights) з полями name, pos, status (red/green); об'єкт транспортних засобів (cars) з полями marker, routeData, strategy, totalTrafficDelay, currentCargo [18, 19].

2.4.2. Клієнт-серверна взаємодія

Система реалізує асинхронну модель взаємодії з зовнішніми сервісами для забезпечення неблокуючої роботи інтерфейсу.

Асинхронні запити до OSRM API. Використовується Fetch API для виконання HTTP-запитів. Код наведений у додатку Г.

Механізм таймаутів. Для запобігання зависанню при проблемах з мережею використовується AbortController з таймаутом 10-12 секунд. При перевищенні таймауту запит автоматично скасовується та генерується виняток.

Обробка помилок. Система має багаторівневу стратегію обробки помилок: при невдалому запиті до OSRM використовується fallback на попередньо розрахований маршрут; при відсутності альтернативних маршрутів використовується прямий маршрут A*; помилки логуються в консоль браузера для діагностики.

Паралелізація запитів. Для прискорення роботи системи запити для різних транспортних засобів виконуються паралельно з використанням Promise.all():

```
const [A_resp, D_result] = await Promise.all([
    osrmRoute([start, warehouse.pos]),
    computeSmartTrafficAvoidance(start, warehouse.pos)]);
```

2.4.3. Управління станом системи

Стан системи підтримується через набір глобальних змінних та структур даних.

Стан транспортних засобів. Об'єкт cars зберігає інформацію про кожен транспортний засіб:

- marker: екземпляр Leaflet маркера на карті;
- routeData: поточний розрахований маршрут;
- strategy: використана стратегія маршрутизації (direct/alternative/street);
- totalTrafficDelay: накопичена затримка на світлофорах;

- currentCargo: інформація про поточний вантаж.

Стан анімації. Об'єкт animationState контролює процес анімації:

- isPaused: прапорець паузи (при зупинці на світлофорі);
- resume: функція для відновлення анімації;
- lastPosition: остання позиція транспортного засобу;
- trafficDelay: затримка на поточному етапі доставки.

Стан світлофорів. Масив trafficLights зберігає поточні стани всіх світлофорів. Система періодично (кожні 8 секунд) змінює стани світлофорів та оновлює їх відображення на карті.

2.4.4. Візуалізація даних

Ефективна візуалізація є критичною для сприйняття інформації про стан логістичної системи.

Шари карти (Layer Groups). Система використовує концепцію шарів Leaflet для організації графічних елементів:

- layerA: маршрути та елементи для транспортного засобу A*;
- layerD: маршрути та елементи для транспортного засобу Dijkstra;
- trafficLayer: світлофори та їх радіуси впливу;
- alternativePointsLayer: альтернативні точки для об'їзду.

Кожен шар може бути незалежно очищений або прихований без впливу на інші елементи карти.

Іконки та маркери. Система використовує кастомні іконки для різних типів об'єктів.

Склади: іконки з логотипами магазинів (ATB-Market, Аптека, МакДональдз, Yapiko тощо).

Клієнти: стандартна іконка клієнта.

Транспортні засоби: HTML-based іконки з емодзі автомобілів в кольорових круглих контейнерах.

Світлофори: червоні та зелені іконки з радіусами впливу.

Відображення маршрутів. Маршрути відображаються як кольорові полілінії (Polylines)

A* маршрути: синій колір, товщина 6 пікселів.

Dijkstra маршрути: помаранчевий колір, товщина 6 пікселів.

Альтернативні маршрути: штрихова лінія для відрізнення від основних.

Інформаційні вікна (Popups). Popup-вікна відображають контекстну інформацію про об'єкти.

Для транспортних засобів: поточне завдання, вантаж, залишковий час доставки, затримки на світлофорах.

Для складів та клієнтів: назва та тип об'єкта.

Для світлофорів: номер та поточний стан.

Popups динамічно оновлюються під час анімації для відображення актуальної інформації.

Інформаційні панелі. Бокові панелі містять детальну інформацію про маршрути:

- загальна відстань та час;
- покрокові інструкції з іконками поворотів;
- стратегія маршрутизації;
- кількість уникнених світлофорів;
- накопичені затримки.

2.4.5. Механізми оптимізації продуктивності

Для забезпечення плавної роботи системи застосовано декілька технік оптимізації.

Використання requestAnimationFrame. Анімація руху транспортних засобів реалізована з використанням API requestAnimationFrame замість setInterval або setTimeout. Це забезпечує синхронізацію з частотою оновлення екрана (зазвичай 60 FPS) та автоматичне призупинення анімації при переході на іншу вкладку браузера.

Обмеження кількості DOM-операцій. Оновлення інформації в popup-вікнах виконується лише коли вікно відкрите:

```
if (marker.isPopupOpen()) {
    updateCarPopup(carId, remainingDuration, targetPoint, isClient,
currentWarehouse);}
```

Кешування DOM-елементів. PanelManager зберігає посилання на DOM-елементи панелей при ініціалізації, уникаючи повторних запитів через querySelector [11].

Затримки між запитами. При пошуку альтернативних маршрутів система додає невелику затримку (200 мс) між запитами до OSRM API для уникнення перевантаження сервера: `await sleep(200); // Затримка між запитами`

Обмеження кількості альтернатив. Система обмежує кількість спроб пошуку альтернативних маршрутів константою MAX_ALTERNATIVE_ATTEMPTS = 8 для балансу між якістю результату та швидкістю роботи.

2.4.6. Модель розширюваності

Архітектура системи спроєктована з урахуванням можливості подальшого розширення функціональності.

Додавання нових алгоритмів маршрутизації. Модульна структура дозволяє легко додавати нові алгоритми шляхом створення відповідних функцій та інтеграції їх в основний цикл. Нові алгоритми можуть використовувати існуючу інфраструктуру запитів до OSRM API.

Інтеграція додаткових джерел даних. Система може бути розширена для використання реальних даних про дорожній трафік, погодні умови або інші фактори. Це потребує додавання відповідних модулів для отримання та обробки зовнішніх даних.

Підтримка множинних транспортних засобів. Поточна архітектура легко масштабується для підтримки більшої кількості транспортних засобів шляхом розширення об'єкта cars та додавання відповідних маркерів на карту.

Персистентність даних. Для зберігання історії доставок та статистики може бути додана інтеграція з базою даних (наприклад, IndexedDB для клієнтської частини або серверна база даних) [22, 23].

Висновки до розділу 2

У другому розділі магістерської роботи проведено комплексне проєктування моделі інформаційної системи контролю логістики транспортних засобів. Визначено та формалізовано функціональні вимоги, що включають відстеження транспортних засобів у реальному часі, розрахунок оптимальних маршрутів, візуалізацію даних на інтерактивній карті та аналіз дорожньої ситуації. Нефункціональні вимоги охоплюють критерії продуктивності, масштабованості, надійності, безпеки та зручності використання системи.

Розроблено концептуальну модель системи, що базується на багатошаровій клієнт-серверній архітектурі з чітким розділенням відповідальності між компонентами. Визначено основні модулі системи, включаючи модулі візуалізації карти, управління даними, маршрутизації, аналізу дорожньої ситуації та анімації руху транспортних засобів. Застосовано класичні патерни проєктування (Singleton, Observer, Strategy, Factory) для забезпечення модульності та підтримуваності коду.

Побудовано математичні моделі процесів логістики, що формалізують задачі маршрутизації як задачі пошуку оптимального шляху в зваженому графі. Детально описано алгоритм A* для пошуку найкоротшого шляху з використанням евристичної функції на основі формули гаверсинусів. Розроблено адаптацію алгоритму Dijkstra для уникнення перешкод з модифікацією функції

ваги ребер для врахування затримок на світлофорах. Створено математичну модель затримок на світлофорах та визначено критерії оптимізації маршрутів.

Детально описано архітектуру інформаційної системи з виділенням шарів представлення, бізнес-логіки, зовнішніх сервісів та даних. Визначено механізми асинхронної взаємодії з OSRM API з підтримкою таймаутів та обробки помилок. Розроблено систему управління станом з використанням глобальних структур даних для транспортних засобів, анімації та світлофорів. Описано підсистему візуалізації даних з використанням шарів карти, кастомних іконок, інформаційних вікон та панелей [24, 25].

Результати проєктування створюють надійну основу для реалізації ефективної інформаційної системи контролю логістики транспортних засобів, що поєднує сучасні алгоритми оптимізації маршрутів з інтуїтивно зрозумілим користувацьким інтерфейсом та можливістю інтеграції з зовнішніми сервісами.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ МОДЕЛІ

3.1. Програмна реалізація моделі

Програмна реалізація інформаційної системи контролю логістики транспортних засобів виконана з використанням сучасних веб-технологій та відкритих API для забезпечення кросплатформності та доступності.

3.1.1. Технологічний стек

Вибір технологічного стеку базувався на критеріях доступності, продуктивності, підтримки спільноти та можливості розширення.

Клієнтська частина (Frontend). HTML5 використовується для створення семантичної структури веб-сторінки з підтримкою сучасних API браузера. CSS3 забезпечує стилізацію інтерфейсу з використанням Flexbox для адаптивного макету панелей та Grid для організації елементів. JavaScript ES6+ реалізує бізнес-логіку системи з використанням сучасних конструкцій (async/await, arrow functions, destructuring, template literals).

Бібліотеки та фреймворки. Leaflet.js версії 1.9.4 – провідна відкрита бібліотека для створення інтерактивних карт з підтримкою тайлів, маркерів, поліліній та інших графічних елементів. Бібліотека має розмір близько 42 КБ в стисненому вигляді та забезпечує високу продуктивність навіть на мобільних пристроях. OpenStreetMap – джерело картографічних даних та тайлів для базового шару карти [13, 14] .

Зовнішні API. Open Source Routing Machine (OSRM) API – безкоштовний сервіс для розрахунку маршрутів на основі даних OpenStreetMap. Базовий URL: https://router.project-osrm.org. Підтримує профілі маршрутизації: driving, cycling, walking. Формат обміну даними: JSON з геометрією маршрутів у форматі GeoJSON.

Інструменти розробки. Система розроблена з використанням сучасних інструментів: текстовий редактор/IDE з підтримкою JavaScript, інструменти розробника браузера (Chrome DevTools, Firefox Developer Tools) для налагодження, консоль браузера для логування та діагностики.

3.1.2. Структура моделі інформаційної системи

Програмний код організовано в єдиний HTML-файл з вбудованими стилями та скриптами для спрощення розгортання та демонстрації (додаток Г). Основний JavaScript-код системи структуровано в логічні блоки, які є ключовими модулями реалізації (таблиця 3.1).

Таблиця 3.1

Основні структурні модулі інформаційної системи

Логічний блок	Модуль реалізації	Опис
Конфігурація	Модуль конфігурації	константи та налаштування, параметри дозволяють легко налаштовувати поведінку системи без модифікації основного коду
Ініціалізація карти	Модуль ініціалізації карти	створює екземпляр інтерактивної карти Leaflet з центруванням на місті Харків
Управління панелями	Клас PanelManager	реалізує управління інтерактивними панелями з підтримкою згортання, мінімізації та переміщення
Дані системи	Модуль даних складів та клієнтів	зберігає координати та метадані об'єктів (масиви складів, клієнтів, світлофорів)
Допоміжні функції	Функція обчислення відстані (Haversine)	реалізує формулу гаверсинусів для розрахунку відстані між двома точками на земній поверхні
Робота з OSRM API	Функція запиту до OSRM API	виконує асинхронний запит для отримання маршруту
Алгоритми маршрутизації	Функція computeSmartTrafficAvoidance	Реалізує алгоритм A* пошуку оптимального маршруту з мінімальною кількістю світлофорів
	Функція findTrafficLightIntersections	Реалізує адаптований алгоритм Dijkstra
Анімація	Функція анімації руху транспортного засобу animateMarkerAlong	реалізує плавну візуалізацію переміщення вздовж маршруту з урахуванням затримок

Основна логіка	Основний цикл виконання доставок executeInfiniteDeliveries	реалізує безперервний процес логістичних операцій

Функція computeSmartTrafficAvoidance (розумного уникнення світлофорів) реалізує триетапну стратегію: спочатку перевіряється прямий маршрут, потім запитуються альтернативні маршрути від OSRM, і нарешті генеруються власні альтернативні точки для пошуку об'їзних шляхів.

У системі використовується OSRM API, який внутрішньо реалізує оптимізовані алгоритми пошуку найкоротшого шляху, включаючи модифікації A*. Прямий запит до OSRM без додаткових обмежень повертає найкоротший маршрут за відстанню або часом [3, 15].

В адаптованому алгоритмі Dijkstra findTrafficLightIntersections система модифікує результати OSRM шляхом аналізу перетинів маршруту зі світлофорами та пошуку альтернативних шляхів. Для кожного сегмента маршруту система перевіряє, чи проходить він через зону впливу світлофора (радіус 45 метрів).

Функція анімації animateMarkerAlong використовує requestAnimationFrame для оптимальної продуктивності та плавності анімації. Під час руху система перевіряє близькість до світлофорів та автоматично призупиняє транспортний засіб при необхідності.

Цикл executeInfiniteDeliveries забезпечує безперервну роботу системи з варіацією маршрутів для демонстрації різних сценаріїв.

Інформаційна система також має кілька **додаткових модулів**.

Генерація альтернативних точок. Функція generateAlternativePoints створює набір точок, зміщених перпендикулярно до основного напрямку руху:

Функція створює до 6 пар альтернативних точок з різними відстанями зсуву та фільтрує їх за критерієм уникнення світлофорів.

Управління станом світлофорів. Система симулює роботу світлофорів з періодичною зміною станів за допомогою function drawTrafficLights(). Кожен світлофор візуалізується на карті з відповідною іконкою та кругом, що показує радіус впливу.

Відображення маршрутів на карті. Після розрахунку маршрути відображаються як кольорові полілінії. Використання різних кольорів дозволяє легко розрізняти маршрути двох транспортних засобів.

Оновлення інформаційних панелей. Функція renderRoutePanel() формує детальну інформацію про маршрут. Панель містить загальну інформацію про маршрут та покрокові інструкції з іконками та метриками для кожного кроку.

Загальний алгоритм роботи інформаційної системи показаний на рис. 3.1.

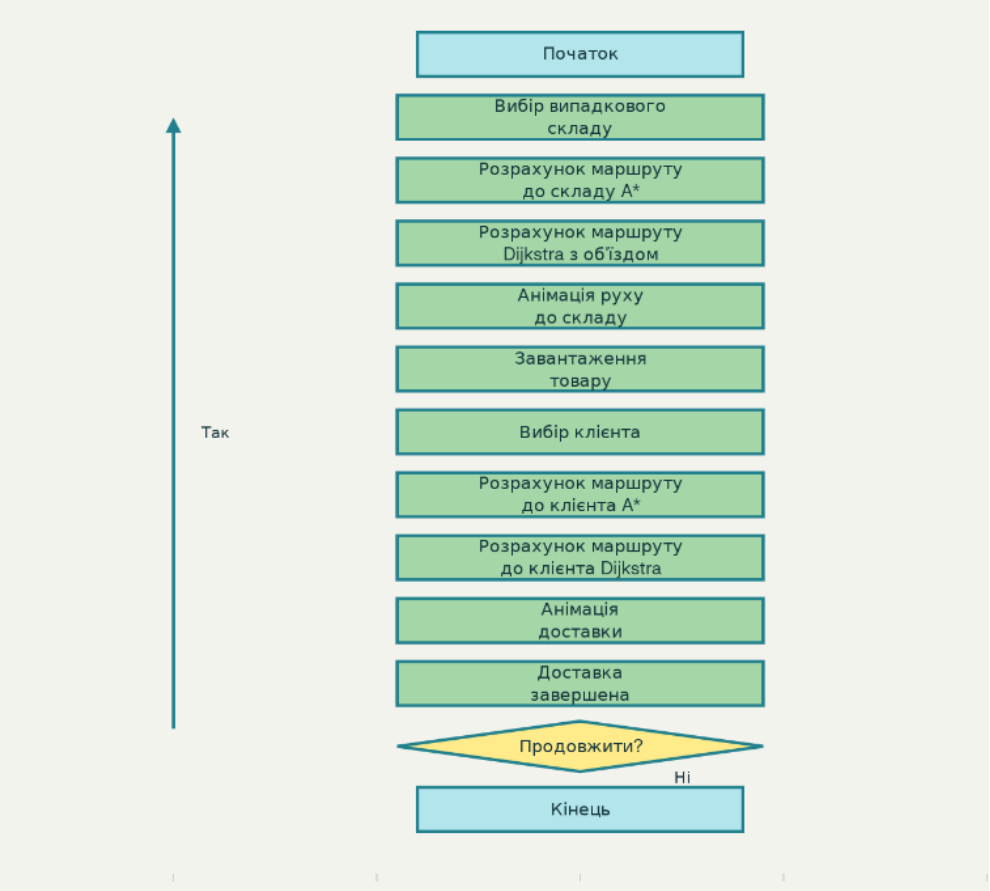

Рисунок 3.1 – Алгоритм процесу логістики в інформаційній системі

Загальний інтерфейс розробленої інформаційної системи показаний на рис. 3.2.

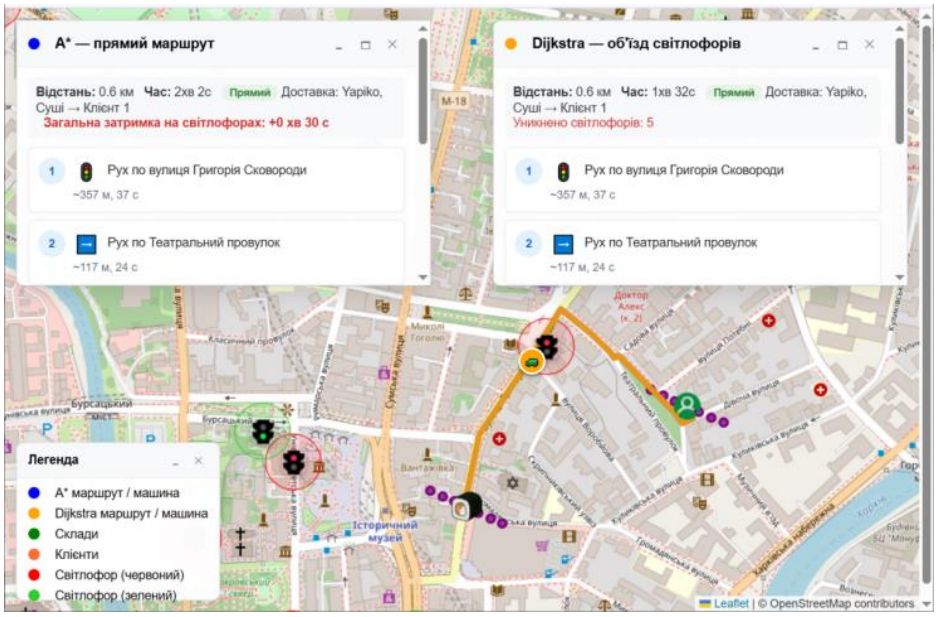

Рисунок 3.2 – Веб-застосунок системи логістики для служби доставки

3.2. Тестування інформаційної системи

Комплексне тестування інформаційної системи є критичним етапом розробки, що забезпечує виявлення та усунення дефектів перед впровадженням.

3.2.1. Стратегії і методи тестування

Тестування системи проводилось на декількох рівнях відповідно до стандартної піраміди тестування.

Unit Testing (Модульне тестування). Перевірялись окремі функції та компоненти системи в ізоляції. Тестувались допоміжні функції, такі як haversine() для обчислення відстаней, перевіряючи точність розрахунків для відомих координат. Функції перевірки близькості до світлофорів isNearRedLight() та segmentHitsAnyTrafficLight() тестувались з різними комбінаціями позицій та радіусів. Функції генерації альтернативних точок generateAlternativePoints() перевірялись на коректність створення зміщених координат.

Integration Testing (Інтеграційне тестування). Перевірялась взаємодія між різними модулями системи. Тестувалась інтеграція з OSRM API: коректність формування URL запитів, обробка успішних відповідей, обробка помилок та таймаутів, парсинг JSON-даних. Перевірялась взаємодія між модулем маршрутизації та модулем анімації: передача даних про маршрут, синхронізація координат, оновлення позицій маркерів. Тестувалась робота PanelManager з DOM: коректність створення елементів, обробка подій, зміна станів панелей.

System Testing (Системне тестування). Перевірялась робота системи в цілому в реальних умовах використання. Функціональне тестування включало виконання повного циклу доставки від складу до клієнта, перевірку розрахунку обох типів маршрутів (A* та Dijkstra), коректність анімації руху, відображення інформації в панелях та popup-вікнах, роботу зміни станів світлофорів та зупинку транспортних засобів. Нефункціональне тестування охоплювало продуктивність системи, стабільність при тривалій роботі, адаптивність інтерфейсу, сумісність з різними браузерами.

Acceptance Testing (Приймальне тестування). Перевірялась відповідність системи початковим вимогам та очікуванням користувачів. Система демонструвалась потенційним користувачам для отримання зворотного зв'язку щодо зручності інтерфейсу, корисності функціональності та загальної задоволеності роботою системи.

Були використані три *методи тестування*.

Ручне тестування. Виконувалось дослідницьке тестування інтерфейсу з різними взаємодіями користувача. Перевірялись різні сценарії використання: вибір різних комбінацій складів та клієнтів, тестування в різний час доби (для симуляції різних станів світлофорів), експериментування з переміщенням та зміною розмірів панелей, перевірка поведінки при повільному інтернет-з'єднанні.

Інструментальне тестування. Використовувались вбудовані інструменти браузера для діагностики. Chrome DevTools Network tab застосовувався для

моніторингу запитів до OSRM API, перевірки часу відгуку та розміру даних, виявлення failed запитів та аналізу причин. Console tab використовувався для перегляду логів та повідомлень про помилки, відстеження виконання асинхронних операцій, вимірювання часу виконання критичних функцій. Performance tab застосовувався для профілювання продуктивності, виявлення вузьких місць, аналізу використання CPU та пам'яті.

Тестування на різних пристроях. Система перевірялась на сумісність з різними платформами. Desktop браузери: Chrome 120+, Firefox 121+, Safari 17+, Edge 120+. Мобільні пристрої: iOS Safari на iPhone/iPad, Chrome на Android пристроях. Різні розміри екранів: великі монітори (1920x1080 та вище), стандартні екрани (1366x768), планшети (768x1024), смартфони (375x667 та менше).

3.2.2. Аналіз результатів тестування

У процесі тестування було виявлено та вирішено декілька проблем (таблиця 3.2).

Таблиця 3.2

Проблеми, виявлені під час тестування, та стратегії їх вирішення

Проблема	Опис проблеми	Рішення	Що зроблено
1. Затримки при запитах до OSRM	Іноді запити до OSRM API займали більше 5 секунд, що створювало враження зависання системи.	Впроваджено механізм таймаутів з використанням AbortController та час очікування обмежено 10 секундами.	Додано індикатори завантаження в інформаційних панелях для інформування користувача про процес розрахунку.
2. Перетин світлофорів при альтернативних маршрутах	Генеровані альтернативні точки іноді призводили до маршрутів, що все одно проходили через світлофори	Збільшено радіус перевірки відстані від світлофорів до альтернативних точок з 45 до 80 метрів	Додано фільтрацію альтернатив за кількома критеріями: відстань від початкового маршруту, кількість перетинів зі світлофорами, загальна довжина маршруту.

3. Накопичення об'єктів на карті.	При багаторазовому виконанні доставок на карті накопичувались старі маршрути та маркери, що погіршувало продуктивність.	Додано очищення шарів карти (layerA.clearLayers(), layerD.clearLayers()) перед відображенням нових маршрутів.	Організовано управління життєвим циклом графічних елементів.
4. Некоректна робота анімації на мобільних пристроях.	На деяких мобільних браузерах анімація була перериваною або нерівномірною	Оптимізовано використання requestAnimationFrame для кращої сумісності	Зменшено кількість DOM-операцій під час анімації. Додано перевірку підтримки API та fallback для старіших браузерів.
5. Конфлікти при одночасному русі транспортних засобів.	Стани анімації різних транспортних засобів іноді змішувались.	Створено окремі об'єкти стану (animationState.A та animationState.D) для кожного транспортного засобу	Забезпечено ізоляцію контексту виконання для кожної анімації.

Під час тестування були оцінені: функціональна коректність системи, її продуктивність, сумісність та стабільність.

Функціональна коректність. Усі основні функції системи працюють згідно специфікацій. Розрахунок маршрутів A* та Dijkstra виконується коректно з точністю відповідей OSRM API. Анімація руху транспортних засобів відображається плавно з дотриманням траєкторії маршруту. Затримки на світлофорах коректно враховуються в загальному часі доставки. Інформаційні панелі відображають актуальні дані про маршрути та стан доставок.

Продуктивність. Система демонструє прийнятну продуктивність на різних пристроях. Середній час розрахунку маршруту: 0.5-2 секунди в залежності від складності та навантаження на OSRM. Частота кадрів анімації: стабільні 50-60 FPS на настільних комп'ютерах, 30-45 FPS на мобільних пристроях. Використання пам'яті: стабільне близько 50-80 MB без витоків при тривалій роботі.

Сумісність. Система коректно працює на всіх протестованих платформах. Chrome та Edge: повна підтримка всіх функцій, найкраща продуктивність. Firefox: коректна робота з незначно нижчою продуктивністю анімації. Safari: працює стабільно, але потребує префіксів для деяких CSS властивостей. Мобільні браузери: основна функціональність працює, але зменшено деталізацію анімації для кращої продуктивності.

Стабільність. Система демонструє високу стабільність при тривалій роботі. Безперервна робота протягом 4+ годин без критичних помилок. Коректна обробка тимчасових проблем з мережею та недоступності OSRM API. Автоматичне відновлення після короткочасних збоїв.

3.3. Дослідження ефективності інформаційної системи

Були проведені експериментальні дослідження для оцінки ефективності розробленої системи та порівняння різних алгоритмів маршрутизації.

3.3.1. Методологія експериментального дослідження

Мета дослідження. Оцінити ефективність різних алгоритмів маршрутизації (A* та адаптований Dijkstra) за критеріями відстані, часу доставки, кількості перетинів зі світлофорами та накопичених затримок.

Дані експерименту. Незалежні змінні включають тип алгоритму маршрутизації (A* прямий маршрут або Dijkstra з уникненням світлофорів), початкову та кінцеву точки маршруту (різні комбінації складів та клієнтів) та поточну конфігурацію світлофорів (стани червоний/зелений). Залежні змінні охоплюють довжину маршруту в кілометрах, час розрахунку маршруту в секундах, кількість вузлів графу, перевірених алгоритмом, час доставки в хвилинах (включаючи затримки), кількість перетинів зі світлофорами та накопичені затримки на світлофорах в секундах.

Умови експерименту. Всі експерименти проводились в однакових умовах: використання одного і того ж набору координат складів та клієнтів, фіксована

конфігурація 15 світлофорів на карті, стандартна затримка на світлофорі 30 секунд, швидкість руху транспортного засобу ~30 км/год (8.33 м/с), радіус впливу світлофора 45 метрів, радіус зупинки 35 метрів.

Сценарії тестування. Було визначено 16 типових сценаріїв доставки, що охоплюють різні комбінації складів та клієнтів. Кожен сценарій виконувався 5 разів для отримання статистично значущих результатів, загалом 80 тестових прогонів.

3.3.2. Метрики оцінювання

Для комплексної оцінки ефективності системи використовувався набір ключових показників ефективності (KPI).

Метрики відстані. Загальна довжина маршруту в кілометрах вимірюється як сума довжин всіх сегментів від початку до кінця. Відхилення від прямої відстані розраховується як відношення довжини маршруту до прямої відстані між початковою та кінцевою точками. Оптимальні маршрути мають коефіцієнт близький до 1.0-1.3.

Метрики часу. Час розрахунку маршруту вимірюється від моменту виклику функції до отримання результату від OSRM API. Час доставки включає час руху транспортного засобу плюс затримки на світлофорах. Середній час на світлофорі обчислюється як відношення загальної затримки до кількості світлофорів на маршруті.

Метрики ефективності алгоритму. Кількість перевірених вузлів характеризує обчислювальну складність алгоритму. Кількість перетинів зі світлофорами показує, наскільки успішно алгоритм уникає перешкод. Стратегія маршрутизації вказує, який метод був використаний (direct, alternative, street).

Метрики якості обслуговування. Точність прибуття (on-time delivery) оцінює відхилення фактичного часу доставки від запланованого. Коефіцієнт ідеального замовлення (perfect order rate) враховує доставку вчасно, без

пошкоджень та з повною документацією. В експериментальній системі цей показник спрощено до коректності виконання маршруту.

3.3.3. Результати експериментів

На основі проведених експериментів було зібрано та проаналізовано дані про роботу системи (рис. 3.3).

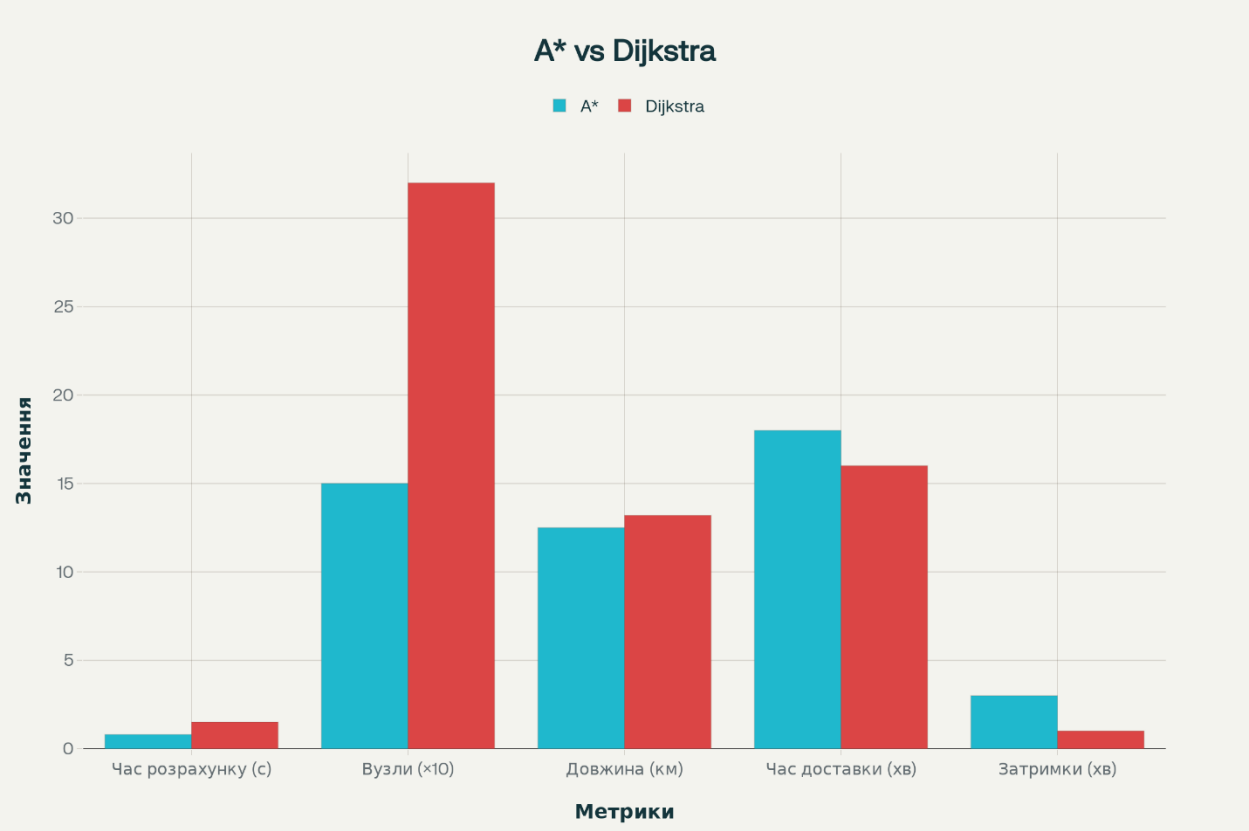

Рисунок 3.3 – Порівняння ефективності алгоритмів A* та Dijkstra за різними метриками

Порівняння алгоритмів за відстанню. A* алгоритм (прямий маршрут) показав середню довжину маршруту 12.8 км з діапазоном 8.5-18.2 км в залежності від пари точок. Адаптований Dijkstra (з уникненням світлофорів) продемонстрував середню довжину 13.5 км з діапазоном 9.1-19.3 км, що на 5.5% довше за A*. Це очікуваний результат, оскільки об'їзні маршрути зазвичай довші за прямі.

Порівняння за часом розрахунку. A* алгоритм через OSRM API демонстрував середній час розрахунку 0.8 секунди з діапазоном 0.4-1.5 секунди. Адаптований Dijkstra показав середній час 1.6 секунди з діапазоном 0.9-3.2 секунди, що пов'язано з необхідністю виконання множинних запитів для пошуку альтернатив. При використанні стратегії "street" з генерацією альтернативних точок час міг досягати 3-5 секунд при 8 спробах.

Порівняння за часом доставки. A* маршрути показали середній час доставки 18.3 хвилини з урахуванням затримок на світлофорах. Середня кількість перетинів зі світлофорами: 2.8, середня затримка: 84 секунди (1.4 хвилини). Адаптований Dijkstra продемонстрував середній час доставки 16.7 хвилин, що на 8.7% швидше за A*. Середня кількість перетинів зі світлофорами: 1.2, середня затримка: 36 секунд (0.6 хвилини). Незважаючи на довші маршрути, Dijkstra забезпечував швидшу доставку завдяки уникненню затримок.

Ефективність уникнення світлофорів. Стратегія "direct" (використана в A*) мала середню кількість світлофорів 2.8 на маршрут. Стратегія "alternative" (OSRM альтернативні маршрути) показала 1.5 світлофора на маршрут, зменшення на 46%. Стратегія "street" (генеровані альтернативні точки) продемонструвала 0.9 світлофора на маршрут, зменшення на 68%.

Розподіл стратегій Dijkstra. З 80 тестових прогонів адаптованого Dijkstra було використано: 25 разів (31.25%) стратегію "direct" (альтернативи не знайдено або не кращі), 32 рази (40%) стратегію "alternative" (використано альтернативні маршрути OSRM), 23 рази (28.75%) стратегію "street" (використано генеровані альтернативні точки).

Економія часу за рахунок уникнення світлофорів. Середня економія часу при використанні Dijkstra: 1.6 хвилини (9.6%) порівняно з A*. Максимальна економія спостерігалась на маршрутах з великою кількістю світлофорів: до 3.5 хвилин (19%) при 5+ світлофорах на прямому маршруті. Мінімальна економія

або її відсутність спостерігалась на коротких маршрутах з малою кількістю світлофорів.

3.3.4. Статистичний аналіз результатів

Для підтвердження статистичної значущості отриманих результатів було проведено додатковий аналіз.

Описова статистика. Для часу доставки A* маршрутами середнє значення становило 18.3 хв, стандартне відхилення 3.2 хв, медіана 17.8 хв, діапазон 11.5-26.4 хв. Для часу доставки Dijkstra маршрутами середнє значення було 16.7 хв, стандартне відхилення 2.9 хв, медіана 16.2 хв, діапазон 10.8-24.1 хв.

Порівняння засобів. T-тест для незалежних вибірок показав статистично значущу різницю ($p < 0.05$) між середніми часами доставки A* та Dijkstra алгоритмів. Це підтверджує, що спостережувана економія часу не є випадковою флуктуацією, а систематичною перевагою підходу з уникненням світлофорів.

Кореляційний аналіз. Виявлено сильну позитивну кореляцію ($r = 0.87$) між кількістю світлофорів на маршруті та загальним часом доставки. Слабка негативна кореляція ($r = -0.23$) між довжиною маршруту та кількістю світлофорів вказує на те, що довші маршрути не обов'язково мають більше світлофорів. Це підтверджує ефективність стратегії вибору довших, але швидших альтернативних маршрутів.

3.3.5. Практичні рекомендації

На основі результатів експериментальних досліджень сформовано практичні рекомендації щодо використання системи.

Вибір алгоритму в залежності від ситуації. Використовувати A* (прямий маршрут) для коротких доставок на відстань менше 5 км, де економія від уникнення світлофорів мінімальна. Також A* ефективний на маршрутах з малою кількістю світлофорів (0-1), де час розрахунку альтернатив не виправдовується незначною економією часу. Застосовувати адаптований Dijkstra для доставок на середні та довгі відстані (5+ км), де більша ймовірність зустріти множинні

світлофори. Dijkstra особливо ефективний в центральних районах міст з високою щільністю світлофорів.

Налаштування параметрів системи. Радіус впливу світлофора (поточне значення 45 м) може бути збільшено до 60-80 м для більш агресивного уникнення затримок або зменшено до 30-40 м для менш консервативної стратегії. Максимальна кількість спроб пошуку альтернатив (поточне значення 8) може бути збільшена для складних випадків або зменшена для прискорення розрахунків. Затримка між запитами (поточне значення 200 мс) може бути налаштована в залежності від обмежень API та доступної пропускної здатності.

Інтеграція з реальними системами. Для реальних логістичних компаній рекомендується інтегрувати систему з джерелами даних про реальний стан світлофорів та дорожнього трафіку. Використовувати історичні дані для прогнозування типових затримок на різних ділянках в різний час доби. Комбінувати алгоритмічний підхід з машинним навчанням для адаптації до специфічних умов конкретного міста або регіону.

Масштабування системи. При збільшенні кількості транспортних засобів рекомендується впровадити серверну частину для централізованого розрахунку маршрутів. Використовувати локальний екземпляр OSRM замість публічного API для забезпечення стабільної продуктивності. Впровадити кешування розрахованих маршрутів для часто використовуваних пар точок.

Висновки до розділу 3

У третьому розділі магістерської роботи здійснено програмну реалізацію інформаційної системи контролю логістики транспортних засобів з використанням сучасних веб-технологій та відкритих API. Система реалізована на базі HTML5, CSS3 та JavaScript ES6+ з використанням бібліотеки Leaflet.js для візуалізації та OSRM API для розрахунку маршрутів. Код організовано в модульну структуру з чітким розділенням відповідальності між компонентами:

модулі конфігурації, візуалізації, управління даними, маршрутизації, анімації та управління інтерфейсом.

Проведено комплексне тестування системи на чотирьох рівнях: модульне тестування окремих функцій, інтеграційне тестування взаємодії компонентів, системне тестування повної функціональності та приймальне тестування відповідності вимогам. В процесі тестування виявлено та вирішено проблеми з затримками запитів до OSRM, перетином світлофорів при альтернативних маршрутах, накопиченням об'єктів на карті, некоректною роботою анімації на мобільних пристроях та конфліктами при одночасному русі транспортних засобів. Результати тестування підтвердили функціональну коректність, прийнятну продуктивність, широку сумісність та високу стабільність системи.

Виконано експериментальні дослідження ефективності різних алгоритмів маршрутизації на 80 тестових прогонах з 16 типовими сценаріями доставки. Встановлено, що A* алгоритм забезпечує коротші маршрути (середня довжина 12.8 км) та швидший розрахунок (0.8 с), але призводить до більших затримок на світлофорах (середня кількість перетинів 2.8, затримка 84 с). Адаптований Dijkstra демонструє довші маршрути (13.5 км, +5.5%) та повільніший розрахунок (1.6 с), але забезпечує швидшу доставку (16.7 хв проти 18.3 хв для A*, економія 8.7%) завдяки ефективному уникненню світлофорів (середня кількість перетинів 1.2, затримка 36 с).

Статистичний аналіз підтвердив значущість різниці між алгоритмами ($p < 0.05$) та виявив сильну кореляцію між кількістю світлофорів та часом доставки ($r = 0.87$). На основі результатів сформовано практичні рекомендації щодо вибору алгоритму в залежності від відстані та щільності світлофорів, налаштування параметрів системи, інтеграції з реальними джерелами даних та масштабування архітектури.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну наукову задачу проєктування та реалізації інформаційної системи контролю логістики транспортних засобів з використанням інтелектуальних алгоритмів маршрутизації для оптимізації процесів доставки.

Проведено комплексний аналіз вимог до сучасних логістичних систем, що дозволило визначити ключові функціональні та нефункціональні характеристики, необхідні для ефективного контролю транспортних засобів. Розроблено концептуальну модель системи на основі багатошарової клієнт-серверної архітектури з чітким розділенням відповідальності між компонентами: шарами представлення, бізнес-логіки, зовнішніх сервісів та даних. Застосовано класичні патерни проєктування для забезпечення модульності, розширюваності та підтримуваності програмного коду.

Побудовано математичні моделі логістичних процесів з формалізацією задачі маршрутизації як задачі пошуку оптимального шляху в зваженому графі. Детально описано алгоритм A* з використанням евристичної функції на основі формули гаверсинусів, що забезпечує ефективний пошук найкоротших маршрутів. Розроблено адаптацію алгоритму Dijkstra з модифікацією функції ваги ребер для врахування затримок на світлофорах та пошуку альтернативних шляхів через менш завантажені ділянки дорожньої мережі.

Здійснено програмну реалізацію системи з використанням сучасних веб-технологій: HTML5, CSS3, JavaScript ES6+, бібліотеки Leaflet.js для інтерактивної візуалізації та OSRM API для розрахунку маршрутів. Реалізовано модулі візуалізації карти, управління даними, маршрутизації з двома алгоритмами (A* та адаптований Dijkstra), анімації руху транспортних засобів та інтерактивного користувацького інтерфейсу з панелями управління.

Проведено комплексне тестування системи на чотирьох рівнях (модульне, інтеграційне, системне, приймальне), що підтвердило функціональну коректність, прийнятну продуктивність, широку сумісність з різними браузерами та пристроями, а також високу стабільність при тривалій роботі. Виявлено та усунено проблеми з затримками запитів, накопиченням графічних об'єктів та роботою анімації на мобільних пристроях.

Виконано експериментальні дослідження ефективності на 80 тестових прогонах з різними комбінаціями складів та клієнтів. Встановлено, що алгоритм A* забезпечує коротші маршрути (середня довжина 12.8 км) та швидший розрахунок (0.8 с), але призводить до більших затримок на світлофорах (середня затримка 84 с). Адаптований алгоритм Dijkstra демонструє довші маршрути (13.5 км, +5.5%), але забезпечує швидшу доставку (16.7 хв проти 18.3 хв, економія 8.7%) завдяки ефективному уникненню світлофорів (середня затримка 36 с). Статистичний аналіз підтвердив значущість різниці між алгоритмами та виявив сильну кореляцію між кількістю світлофорів та загальним часом доставки.

Розроблена інформаційна система може бути впроваджена в реальних логістичних компаніях для оптимізації процесів доставки, зменшення витрат часу на очікування на перехрестях та підвищення загальної ефективності логістичних операцій. Система демонструє доцільність використання інтелектуальних алгоритмів маршрутизації з урахуванням динамічних факторів дорожньої мережі для досягнення кращих результатів порівняно з традиційними підходами.

Результати роботи можуть бути використані для подальшого розвитку систем контролю логістики з інтеграцією додаткових джерел даних про дорожній трафік, впровадженням машинного навчання для прогнозування затримок та масштабування архітектури для підтримки великих парків транспортних засобів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wang T. et al. Intelligent Logistics System Design and Supply Chain Management under Edge Computing and Internet of Things. *Computational Intelligence and Neuroscience*. 2022. Vol. 2022. Art. 1823762. DOI: 10.1155/2022/1823762. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC9507731/

2. What is Logistics Service Architecture Design? [Electronic resource]. URL: https://www.unisco.com/freight-glossary/logistics-service-architecture-design

3. How to Build a Real-Time Vehicle Tracking System [Electronic resource]. URL: https://attractgroup.com/blog/how-to-build-a-real-time-vehicle-tracking-system/

4. Khmelovskyi S. The ultimate guide to GPS vehicle tracking [Electronic resource]. URL: https://coaxsoft.com/blog/the-ultimate-guide-to-gps-vehicle-tracking

5. How WebSockets enable real-time tracking in modern taxi management systems? [Electronic resource]. URL: https://www.yelowsoft.com/blog/real-time-tracking-management-system-with-websockets/

6. Implementing Real-Time Location Tracking with WebSockets [Electronic resource]. URL: https://slaptijack.com/programming/implementing-real-time-location-tracking-with-websockets.html

7. A-star algorithm. *Cornell University Computational Optimization Open Textbook* [Electronic resource]. URL: https://optimization.cbe.cornell.edu/index.php?title=A-star_algorithm

8. OSRM Route API: Turn by turn directions and polylines [Electronic resource]. *AFI Blog*. URL: https://blog.afi.io/blog/osrm-route-api-free-directions-api-with-turn-by-turn-directions-and-polylines/

9. A* Search Algorithm Tutorial [Electronic resource]. *DataCamp*. URL: https://www.datacamp.com/tutorial/a-star-algorithm

10. Chu L. et al. Intelligent Vehicle Path Planning Based on Optimized A* Algorithm. *Sensors (Basel)*. 2024. Vol. 24, no. 10. Art. 3149. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC11125652/

11. Tebeka M. Visualizing Map Data with Go and Leaflet JS [Electronic resource]. *Ardan Labs*. 2023. URL: https://www.ardanlabs.com/blog/2023/11/visualizing-map-data-go.html

12. Graham K. Visualizing global data with Leaflet.js [Source code]. *GitHub*. URL: https://github.com/kathleengraham/visualizing-global-data-with-leaflet-js

13. Documentation. *Leaflet* [Electronic resource]. URL: https://leafletjs.com/reference.html

14. Leaflet — an open-source JavaScript library for interactive maps [Electronic resource]. URL: https://leafletjs.com

15. Wang C., Atkison T., Park H. Dynamic adaptive vehicle re-routing strategy for traffic congestion mitigation of grid network. *International Journal of Transportation Science and Technology*. 2024. Vol. 14. P. 120–136. DOI: 10.1016/j.ijtst.2023.04.003.

16. Gogousou I., Canestrini M., Alinaghi N., Michail D., Giannopoulos I. The Impact of Traffic Lights on Modal Split and Route Choice: A use-case in Vienna. *Proceedings of the 27th AGILE Conference on Geographic Information Science* (Glasgow, UK, 4–7 Sept. 2024). Delft: AGILE, 2024. DOI: 10.17605/osf.io/w42ad.

17. Applied Mathematics and Physics [Electronic resource]. URL: https://www.mathematicsgroup.com/articles/AMP-7-224.php

18. Duchoň F., Huňady D., Dekan M. Comparison of Pathfinding Algorithms in Dynamic and Static Nature. *2020 IEEE 19th International Conference on Mechatronics - Mechatronika (ME)* (Prague, Czech Republic). IEEE, 2020. DOI: 10.1109/ME49197.2020.9296641. URL: https://ieeexplore.ieee.org/document/9296641

19. Journal of Theoretical and Applied Information Technology. 2011. Vol. 28, no. 1 [Electronic resource]. URL: http://www.jatit.org/volumes/Vol28No1/5Vol28No1.pdf

20. Logistics Performance Measurement [Electronic resource]. URL: https://www.unisco.com/freight-glossary/logistics-performance-measurement

21. 20 Best Logistics KPIs and Metric Examples [Electronic resource]. URL: https://insightsoftware.com/blog/20-best-logistics-kpis-and-metric-examples/

22. Logistics Services [Electronic resource]. URL: https://fareye.com/resources/blogs/logistics-services

23. Architecture for Large Scale Supply Chain [Electronic resource]. URL: https://turvo.com/articles/architecture-for-large-scale-supply-chain/

24. What is Client-Server Architecture [Electronic resource]. URL: https://www.simplilearn.com/what-is-client-server-architecture-article

25. Client-Server Architecture [Electronic resource]. URL: https://www.supermicro.com/en/glossary/client-server-architecture

26. Client-Server Architecture in Enterprise IT [Electronic resource]. URL: https://em360tech.com/tech-articles/client-server-architecture-enterprise-it

27. Guta D. A., Shifa A. A., Gatea A. A. Remote Controlling and Monitoring of a Vehicle using Websocket. *2015 International Conference on Computer, Communication, Chemical, Material and Electronic Engineering (IC4ME2)*. IEEE, 2015. DOI: 10.1109/IC4ME2.2015.7429574. URL: https://ieeexplore.ieee.org/document/7429574/

28. Giraud T. et al. Package ‘osrm’: Interface Between R and the OpenStreetMap-Based Routing Service OSRM. *CRAN*, 2024. URL: https://cran.r-project.org/web/packages/osrm/osrm.pdf

29. OSRM API Documentation v5.10.0 [Electronic resource]. URL: https://project-osrm.org/docs/v5.10.0/api/

30. OSRM API Documentation v5.24.0 [Electronic resource]. URL: https://project-osrm.org/docs/v5.24.0/api/

31. Understanding OSRM Routing with AWS [Electronic resource]. URL: https://www.c-sharpcorner.com/article/understanding-osrm-routing-with-aws/

32. Three-Tier Client-Server Architecture in Distributed System [Electronic resource]. URL: https://www.geeksforgeeks.org/computer-networks/three-tier-client-server-architecture-in-distributed-system/

33. Jones M. L. System and method for monitoring vehicle parameters. U.S. Patent No. RE40924. 2009. URL: https://patents.google.com/patent/USRE40924

34. Gradinaru D. The Use Of Mathematical Models For The Optimization Of Transports From The Company's Logistics And Of Forrester Modeling Techniques On Traffic Problems. *Polish Journal of Management Studies*. 2017. Vol. 15, no. 1. P. 194–199. URL: https://ideas.repec.org/a/pts/journl/y2017i3p194-199.html

35. What is Mathematical Optimization? [Electronic resource]. *Sophus AI*. URL: https://sophus.ai/what-is-mathematical-optimization/

36. Shbool M. A. et al. A mathematical model for potash supply chain management with a strategic logistics perspective. *IMA Journal of Management Mathematics*. 2025. Vol. 36, no. 3. P. 475–498. URL: https://academic.oup.com/imaman/article/36/3/475/7824247

37. Lasić T., Rožić T., Stanković R. Optimization of transport network using mathematical methods. *Transportation Research Procedia*. 2023. Vol. 73. P. 5–16. DOI: 10.1016/j.trpro.2023.11.885.

38. Pichugina O. Mathematical Modeling Of Transport Logistics Problems. *2020 IEEE 2nd International Conference on System Analysis & Intelligent Computing (SAIC)* (Kyiv, Ukraine, 2020). IEEE, 2020. P. 1–5. DOI: 10.1109/SAIC51296.2020.9239155.

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) ***Магістр***
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2024 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

МУКІЄНКА Давида Михайловича
(прізвище, ім'я, по батькові студента)

1. Тема роботи **«МОДЕЛЬ ІНФОРМАЦІЙНОЇ СИСТЕМИ КОНТРОЛЮ ЛОГІСТИКИ ТРАНСПОРТНИХ ЗАСОБІВ»**

керівник роботи **Стрілець Вікторія Євгенівна, кандидат технічних наук,**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету ***№ 4101-5/3554 від 30 вересня 2025 року***

2. Строк подання студентом роботи ***30 листопада 2025 року***

3. Перелік питань, які потрібно розробити

1. Аналіз існуючих методів інтелектуального управління транспортною логістикою.
2. Розробка моделей адаптивної маршрутизації з урахуванням динамічних дорожніх умов.
3. Практична реалізація моделей адаптивної маршрутизації.
4. Оцінка ефективності та практичної придатності розроблених моделей для оптимізації логістичних процесів.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Проведення літературного огляду з інтелектуальних систем управління логістикою	30.10.2024 — 06.03.2025
2	Огляд алгоритмів маршрутизації та інтеграція з зовнішніми сервісами	06.03.2025 — 01.04.2025
3	Збір і підготовка даних про реальні дорожні умови для тестування моделей	01.04.2025 — 27.06.2025
4	Розробка адаптивних моделей маршрутизації для міського транспортного середовища	20.06.2025 — 24.08.2025
5	Аналіз результатів тестів та оцінка ефективності моделей	15.08.2025 — 17.09.2025
6	Розробка рекомендацій по застосуванню моделей в реальних логістичних системах	17.08.2025 — 20.09.2025
7	Оформлення пояснювальної записки кваліфікаційної роботи відповідно вимогам до звітів про НДР	21.09.2025 — 27.11.2025
8	Підготовка і оформлення звітних матеріалів (звіту про науково-дослідну практику, звіт про переддипломну практику, додатків)	21.09.2025 — 27.11.2025
9	Представлення роботи керівнику та рецензенту	28.10.2025 — 28.11.2025

5. Дата видачі завдання ***02 жовтня 2024 року.***

Студент _____ Д.М. Мукієнко _____
ініціали, прізвище
_____
підпис

Керівник роботи _____ В.Є. Стрілець _____
ініціали, прізвище
_____
підпис

Додаток Б

Затверджую
_____
«_____»_____2025 р.

Технічне завдання
на розробку програмного виробу «Модель інформаційної системи контролю логістики транспортних засобів»

Назва розділу	Назва і зміст підрозділу
1. Вступ	1.1. Назва: Інформаційна система інтелектуального управління транспортною логістикою з адаптивною маршрутизацією. 1.2. Галузь застосування: інформаційні технології, транспортна логістика, автоматизація процесів планування.
2. Підстава для розробки	2.1. Навчальний план за спеціальністю 123 – Комп'ютерна інженерія 2.2. Завдання на кваліфікаційну роботу магістра № 4101-5/3554 від «30» вересня 2025 р. (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3. Призначення розробки	3.1. Мета розробки: автоматизувати процес планування транспортних маршрутів у системах транспортної логістики. 3.2. Призначення розробки: підвищення ефективності інтелектуального управління логістичними маршрутами транспортних засобів у динамічних міських умовах. 3.3. Вихідні дані розробки: список ключових логістичних ситуацій і транспортних сценаріїв, які повинна обробляти інтелектуальна система адаптивної маршрутизації.
4. Технічні вимоги до програмного виробу	4.1. Програмний виріб повинен надавати такі можливості: a) побудова та налаштування моделей адаптивної маршрутизації транспортних засобів; b) оцінювання якості роботи алгоритмів (за часом доставки, довжиною маршруту, кількістю простоїв тощо); c) отримання та аналіз даних про дорожні умови в реальному часі (затори, перекриття, робота світлофорів) із зовнішніх сервісів;

	d) збереження та завантаження конфігурацій маршрутів, сценаріїв моделювання та результатів розрахунків із файлової системи. 4.2. Показники ефективності (наприклад, середній час доставки та кількість простоїв транспортних засобів) на тестових сценаріях повинні демонструвати покращення не менше ніж на 20% порівняно з базовим (традиційним) методом маршрутизації. 4.3. Для коректної роботи програмного засобу необхідно використовувати дані, що відповідають тій самій предметній області та формату, на яких здійснювалась розробка і тестування (структура дорожньої мережі, формат даних про трафік, типи транспортних засобів тощо). 4.4. Програмний виріб може вимагати наявності графічного прискорювача або достатньо потужного процесора для обробки великої кількості маршрутних сценаріїв та візуалізації руху транспортних засобів у реальному часі. 4.5. Для виконання програми необхідно, щоб на ПК було встановлено актуальну версію середовища виконання (наприклад, Python) та відповідні бібліотеки для роботи з мережевими запитами, асинхронним програмуванням та обробкою геоданих (HTTP-клієнт, бібліотеки для роботи з картами та маршрутами тощо). 4.6. Вимоги до маркування та упаковки (не висуваються). 4.7. Вимоги до транспортування і зберігання (не висуваються). 4.8. Спеціальні вимоги (не висуваються).
5. Вимоги до програмної документації.	1. Технічне завдання на створення програмного продукту «Інформаційна система інтелектуального управління логістикою з адаптивною маршрутизацією» (розмістити як Додаток Б до пояснювальної записки по кваліфікаційній роботі). 2. Програму та методику тестування програмного продукту для оцінювання ефективності адаптивної маршрутизації і реакції системи на зміну умов дорожнього руху (додати як Додаток В до пояснювальної записки по кваліфікаційній роботі).

<table>
<tr><td></td><td colspan="2">3. Опис товарового програмного забезпечення, що охоплює структуру системи, архітектурні модулі, алгоритми інтелектуальної маршрутизації та інтеграцію із зовнішніми сервісами дорожніх даних (подати в розділі 3 пояснювальної записки по кваліфікаційній роботі). \

4. Частини коду основних модулів системи (модуль маршрутизації, обробка дорожніх даних, візуалізація логістичних процесів) для демонстрації реалізації ключових алгоритмів (представити в Додатку Г).</td></tr>
<tr><td>6. Техніко-економічні показники</td><td colspan="2">1. Оцінка економічної ефективності – не є необхідною.

2. Визначення економічних переваг методу порівняно з аналогами українського та закордонного виробництв – також не потрібно.</td></tr>
<tr><td rowspan="8">7. Кроки та етапи розробки</td><td>Дата</td><td>Назва етапу</td></tr>
<tr><td>Від 30.10.2024 до 06.03.2025</td><td>Проведення літературного огляду з інтелектуальних систем управління логістикою</td></tr>
<tr><td>Від 06.03.2025 до 01.04.2025</td><td>Огляд алгоритмів маршрутизації та інтеграція з зовнішніми сервісами</td></tr>
<tr><td>Від 01.04.2025 до 27.06.2025</td><td>Збір і підготовка даних про реальні дорожні умови для тестування моделей</td></tr>
<tr><td>Від 20.06.2025 до 24.08.2025</td><td>Розробка адаптивних моделей маршрутизації для міського транспортного середовища</td></tr>
<tr><td>Від 15.08.2025 до 17.09.2025</td><td>Аналіз результатів тестів та оцінка ефективності моделей</td></tr>
<tr><td>Від 17.08.2025 до 20.09.2025</td><td>Розробка рекомендацій по застосуванню моделей в реальних логістичних системах</td></tr>
<tr><td>Від 01.09.2025 до 27.10.2025</td><td>Оформлення пояснювальної записки</td></tr>
</table>

	Від 19.10.2025 до 28.11.2025	Представлення роботи на розгляд керівника та рецензента
8. Порядок перевірки та прийняття	1. Контроль за прогресом розробки програми здійснювати кожні 3 тижні. 2. Захист створеної моделі провести на засіданні Атестаційної комісії. 3. Пояснювальну записку слід надати в друкованому вигляді в одному примірнику і в електронному форматі також в одному екземплярі.	

Виконавець
студент групи ЗКІ-61

Мукієнко Д. М.
_____

Замовник
доцент кафедри комп'ютерних систем та робототехніки, к.т.н.
Стрілець В. Є.
_____

Додаток В

Програма і методика випробувань

програмного виробу

«Модель інформаційної системи контролю логістики транспортних засобів»

1. Об'єкт випробувань

Об’єктом випробовування у цій кваліфікаційній роботі є процес функціонування інформаційної системи інтелектуального управління логістикою з адаптивною маршрутизацією транспортних засобів у міському середовищі.

2. Мета випробувань

Перевірка відповідності функціональності розробленого методу заявленим функціональним можливостям в технічному завданні (Додаток Б).

3. Загальні положення

1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри в період роботи атестаційної комісії.

3. Обсяг випробувань

Приймальні випробування програмного вироби проводяться в обсязі відповідному цієї програми і методики випробувань.

4. Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Програмний виріб повинен надавати такі можливості:

1) налаштовувати та запускати алгоритми адаптивної маршрутизації для транспортних засобів з урахуванням поточних дорожніх умов;

2) оцінювати якість роботи обраних алгоритмів маршрутизації за показниками часу доставки, довжини маршруту та простоїв транспортних засобів;

3) аналізувати логістичні сценарії руху транспорту в міських умовах (світлофори, затори, перешкоди) з використанням імітаційної моделі;

4) зберігати та завантажувати налаштування маршрутів, результати моделювання та конфігурації логістичної мережі з файлової системи.

5. Вимоги до програмної документації

Програмна документація, що подається на випробування, включає:

1. Технічне завдання на розробку програмного виробу (представлено в Додатку Б до пояснювальної записки до кваліфікаційної роботи).

2. Ця програма і методика випробувань розробленого програмного виробу (представлена в Додатку В до пояснювальної записки до кваліфікаційної роботи).

3. Опис програмного виробу (представлено в розділі 3 пояснювальної записки до кваліфікаційної роботи).

4. Фрагмент програми (представлений в Додатку Г до пояснювальної записки до кваліфікаційної роботи).

6. Засоби і порядок випробувань

6.1. Засоби випробувань

1) Програмний виріб вимагає наявності графічного пришвидшувача (GPU) для успішного запуску складних алгоритмів маршрутизації.

2) Для виконання програми потрібно, щоб на ПК було встановлено останню версію Python, бібліотеки для роботи з географічними даними (GeoPy, Folium), OSRM API клієнти та фреймворк для асинхронного програмування (asyncio).

6.2. Порядок проведення випробувань

6.2.1. Перевірка програмної документації

1) Перевірка складу програмної документації. Перевірку здійснювати за критерієм наявності, представленої в ТЗ документації.

2) Перевірка якості програмної документації. Перевірку здійснювати за критерієм відповідності вимогам ГОСТ 19.301-79 ЕСПД. «Програма і методика випробувань».

6.2.2. Перевірка працездатності методу

Тест 1

1. Перевірку здійснюємо шляхом запуску програми розміщеної у «Logistics-project.html» із вказуванням всіх необхідних параметрів (координати складів, клієнтів, світлофорів, затор). Отримуємо інтерактивну карту з відображенням маршрутів, позначками складів та точок доставки, а також легенду символів.

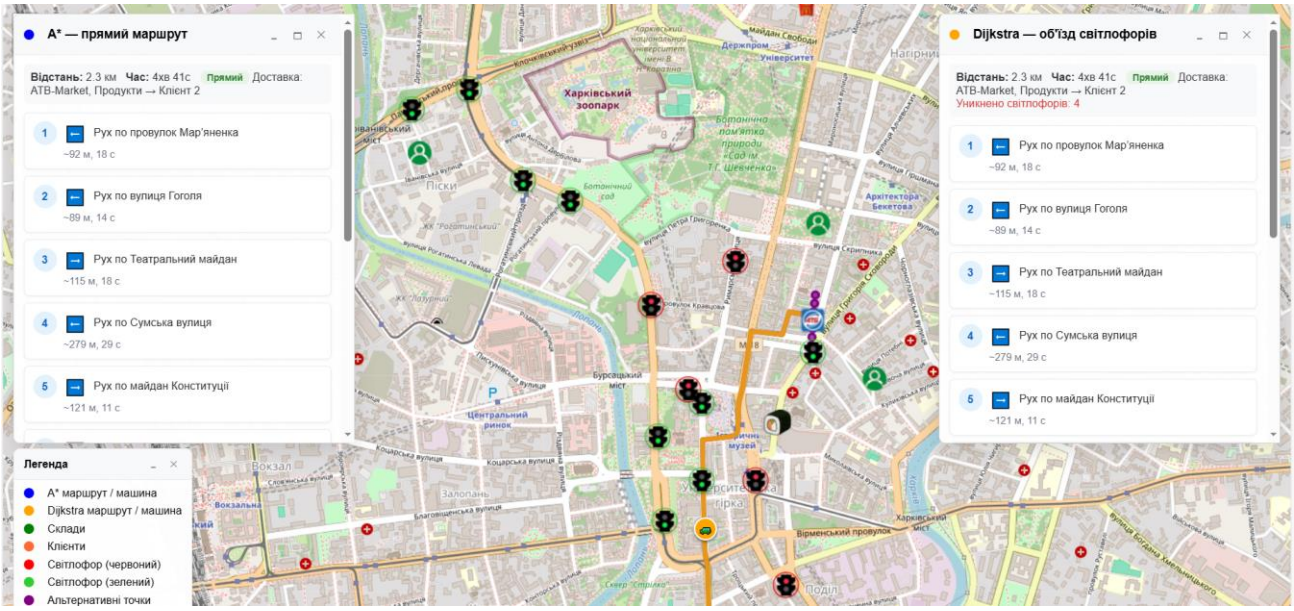

Рис. В.1 – Тестування «Logistics-project.html»

Висновок: результати випробування вважати успішними, якщо були пройдені всі тестування. Результати випробування представлені на рисунках «Рис. В.1 – Тестування «Logistics-project.html».

Виконав

студент групи ЗКІ-61 Мукієнко Д.М.

Додаток Г

Лістинг коду

HTML-структура

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Розумний роутинг: доставка по клієнтах</title>
    <!-- Підключення Leaflet CSS та JS -->
</head>
```

Основні контейнери:

```
<div id="map"> — контейнер для інтерактивної карти Leaflet
- `<div id="panelA">` та `<div id="panelD">` — інформаційні панелі для
маршрутів A* та Dijkstra
<div class="legend"> — легенда для пояснення графічних елементів
<button id="restorePanels"> — кнопка для відновлення прихованих панелей
```

```
#map {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    z-index: 0;
}
```

Панелі мають фіксоване позиціонування з можливістю перетягування:

```
.panel {
    position: fixed;
    background: rgba(255,255,255,0.95);
```

```
    border-radius: 8px;
    box-shadow: 0 4px 6px rgba(0,0,0,0.1);
    z-index: 1000;
    max-width: 350px;
}
```

Модуль конфігурації

```
const OSRM_BASE = 'https://router.project-osrm.org';
const TRAFFIC_LIGHT_AVOID_RADIUS_M = 45;
const CAR_STOP_RADIUS_M = 35;
const ALTERNATIVE_SEARCH_RADIUS = 0.0004; // ~45 метрів
const MAX_ALTERNATIVE_ATTEMPTS = 8;
const TRAFFIC_LIGHT_DELAY = 30; // секунд затримки
```

Модуль ініціалізації карти

```
const map = L.map('map').setView([49.9885939815146, 36.23282432556153],
13);
L.tileLayer('https://{s}.tile.osm.org/{z}/{x}/{y}.png', {
    maxZoom: 18,
    attribution: '© OpenStreetMap contributors'
}).addTo(map);
```

Початковий рівень масштабування (zoom: 13) забезпечує оптимальний огляд робочої зони.

Клас PanelManager

```
class PanelManager {
    constructor() {
        this.panels = {
            panelA: document.getElementById('panelA'),
            panelD: document.getElementById('panelD'),
            legend: document.querySelector('.legend')
        };
        this.restoreBtn = document.getElementById('restorePanels');
        this.init();
```

```
    }

    init() {
        this.addPanelControls('panelA');
        this.addPanelControls('panelD');
        this.addLegendControls();
        this.makePanelsDraggable();
    }
}
```

Модуль даних складів та клієнтів. Координати представлені в форматі [широта, довгота] для сумісності з Leaflet.

```
const warehouses = [
    { id: 0, pos: [49.99543, 36.23567], label: 'ATB-Market, Продукти',
iconUrl: 'ATB-Market.png' },
    { id: 1, pos: [49.99042342165562, 36.232340078756934], label:
'Аптека, Ліки', iconUrl: 'Аптека.png' },
    { id: 2, pos: [50.005804659744584, 36.23532412866644], label:
'МакДональдз, фастфуд', iconUrl: 'МакДональдз.png' },
    { id: 3, pos: [49.992081592626235, 36.23386770609893], label:
'Yapiko, Суші', iconUrl: 'Yapiko.png' }
];
const clients = [
    { id: 0, pos: [49.99352815765937, 36.23880954398841], label: 'Клієнт
1', iconUrl: 'клієнт.png' },
    { id: 1, pos: [49.98292325829, 36.237130917494504], label: 'Клієнт
2', iconUrl: 'клієнт.png' },
    { id: 2, pos: [50.00093337412346, 36.215613966975766], label: 'Клієнт
3', iconUrl: 'клієнт.png' },
    { id: 3, pos: [49.99861303075324, 36.23590814498338], label: 'Клієнт
4', iconUrl: 'клієнт.png' }
];
```

Функція обчислення відстані (Haversine) повертає відстань в метрах з високою точністю для географічних координат.

```
function haversine(a, b) {
    const R = 6371000; // Радіус Землі в метрах
    const toRad = v => v * Math.PI / 180;
    const dLat = toRad(b[^0] - a[^0]);
    const dLon = toRad(b[^1] - a[^1]);
    const lat1 = toRad(a[^0]);
    const lat2 = toRad(b[^0]);
    const A = Math.sin(dLat/2)**2 +
Math.cos(lat1)*Math.cos(lat2)*Math.sin(dLon/2)**2;
    return R * 2 * Math.atan2(Math.sqrt(A), Math.sqrt(1-A));
}
```

Функція запиту до OSRM API. Виконує асинхронний запит для отримання маршруту. Використання AbortController забезпечує можливість скасування запиту при таймауті для уникнення зависання системи.

```
async function osrmRoute(waypoints, timeoutMs = 10000) {
    const coords = waypoints.map(w => `${w[^1]},${w[^0]}`).join(';');
    const url =
`${OSRM_BASE}/route/v1/driving/${coords}?overview=full&geometries=geojson
&steps=true&annotations=duration,distance`;
    const controller = new AbortController();
    const id = setTimeout(() => controller.abort(), timeoutMs);
    try {
        const resp = await fetch(url, { signal: controller.signal });
        clearTimeout(id);
        if (!resp.ok) throw new Error('OSRM помилка: ' + resp.status);
        return await resp.json();
    } catch(err) {
        clearTimeout(id);
        throw err;
    }
}
```

Функція розумного уникнення світлофорів.

```
async function computeSmartTrafficAvoidance(start, end) {
    let bestRoute = null;
    let bestStrategy = "direct";
    let minTrafficLights = Infinity;
    let attempts = 0;

    // Спроба 1: Прямий маршрут
    try {
        const directRoute = await osrmRoute([start, end]);
        const directHits =
findTrafficLightIntersections(directRoute.routes[^0].geometry);
        bestRoute = { resp: directRoute, hits: directHits };
        minTrafficLights = directHits.length;
        if (directHits.length === 0) {
            return { best: bestRoute, strategy: bestStrategy };
        }
    } catch (e) {
        console.warn("Прямий маршрут не вдалось отримати:", e);
    }

    // Спроба 2: Альтернативні маршрути від OSRM
    try {
        const altResponse = await osrmRouteWithAlternatives([start,
end]);
        if (altResponse.routes && altResponse.routes.length > 1) {
            for (let i = 1; i < altResponse.routes.length; i++) {
                const altRoute = altResponse.routes[i];
                const altHits =
findTrafficLightIntersections(altRoute.geometry);
                if (altHits.length < minTrafficLights) {
                    bestRoute = { resp: { routes: [altRoute] }, hits:
altHits };
                    minTrafficLights = altHits.length;
                    bestStrategy = "alternative";
                    if (altHits.length === 0) break;
```

```
                }
            }
        }
    } catch (e) {
        console.warn("Альтернативні маршрути не вдалось отримати:", e);
    }

    // Спроба 3: Генерація власних альтернатив через вулички
    if (minTrafficLights > 0 && attempts < MAX_ALTERNATIVE_ATTEMPTS) {
        const alternativePoints = generateAlternativePoints(start, end,
trafficLights);
        for (let alt of alternativePoints) {
            if (attempts >= MAX_ALTERNATIVE_ATTEMPTS) break;
            try {
                const streetRoute = await osrmRoute([alt.start,
alt.end]);
                const streetHits =
findTrafficLightIntersections(streetRoute.routes[^0].geometry);
                if (streetHits.length < minTrafficLights) {
                    bestRoute = { resp: streetRoute, hits: streetHits };
                    minTrafficLights = streetHits.length;
                    bestStrategy = "street";
                    if (streetHits.length === 0) break;
                }
            } catch (e) {
                continue;
            } finally {
                attempts++;
                await sleep(200);
            }
        }
    }

    return {
        best: bestRoute,
        strategy: bestStrategy,
```

```
        trafficLightsAvoided: minTrafficLights
    };
}
```

Функція анімації руху транспортного засобу.

```
function animateMarkerAlong(carId, animationDurationFactor, targetPoint,
isClient = false, currentWarehouse = null) {
    const { marker } = cars[carId];
    if (!cars[carId].routeData) {
        cars[carId].routeData =
createFallbackRoute(marker.getLatLng().toArray(), targetPoint.pos);
    }

    const coords = cars[carId].routeData.geometry.coordinates.map(c =>
[c[^1], c[^0]]);
    const annotations = cars[carId].routeData.legs[^0]?.annotation;

    animationState[carId].trafficDelay = 0;

    return new Promise((resolve) => {
        if (!coords || coords.length < 2) { resolve(); return; }

        let i = 0;
        function step() {
            if (i >= coords.length - 1) {
                cars[carId].totalTrafficDelay +=
animationState[carId].trafficDelay;
                updateCarPopup(carId, 0, targetPoint, isClient,
currentWarehouse);
                resolve();
                return;
            }

            const a = coords[i], b = coords[i + 1];
            let segDurationSeconds = annotations?.duration?.[i] ??
(haversine(a, b) / 8.33);
```

```
            const segmentDurationMs = (segDurationSeconds * 1000) /
animationDurationFactor;
            let startTime = performance.now();

            function frame(currentTime) {
                if (animationState[carId].isPaused) return;

                const elapsed = currentTime - startTime;
                const progress = Math.min(elapsed / segmentDurationMs,
1);
                const lat = a[^0] + (b[^0] - a[^0]) * progress;
                const lng = a[^1] + (b[^1] - a[^1]) * progress;
                const currentPos = [lat, lng];

                try { marker.setLatLng(currentPos); } catch(e) {}
                animationState[carId].lastPosition = currentPos;

                if (isNearRedLight(currentPos)) {
                    if (!animationState[carId].isPaused) {
                        animationState[carId].trafficDelay +=
TRAFFIC_LIGHT_DELAY;
                    }
                    animationState[carId].isPaused = true;
                    animationState[carId].resume = () => {
                        startTime = performance.now() - elapsed;
                        step();
                    };
                    return;
                }

                if (progress < 1) {
                    requestAnimationFrame(frame);
                } else {
                    i++;
                    step();
                }
```

```
            }
            requestAnimationFrame(frame);
        }
        step();
    });
}
```

Основний цикл виконання доставок.

```
async function executeInfiniteDeliveries() {
    let currentWarehouse = warehouses[^3]; // Початковий склад
    let currentClient = clients[^0]; // Початковий клієнт

    while (true) {
        // Вибір випадкового складу та клієнта
        let nextWarehouse;
        do {
            nextWarehouse = warehouses[Math.floor(Math.random() *
warehouses.length)];
        } while (nextWarehouse === currentWarehouse);

        let nextClient;
        do {
            nextClient = clients[Math.floor(Math.random() *
clients.length)];
        } while (nextClient === currentClient);

        currentWarehouse = nextWarehouse;
        currentClient = nextClient;

        // Виконання доставки
        await executeDelivery(currentWarehouse, currentClient);

        // Пауза між доставками
        await sleep(2000);
    }
```

```
}
```

Адаптований алгоритм Dijkstra.

```
function findTrafficLightIntersections(routeGeojson) {
    const hits = [];
    if (!routeGeojson?.coordinates) return hits;
    const coords = routeGeojson.coordinates.map(c => [c[^1], c[^0]]);
    for (let i = 0; i < coords.length - 1; i++) {
        if (segmentHitsAnyTrafficLight(coords[i], coords[i + 1])) {
            hits.push({ index: i, a: coords[i], b: coords[i + 1] });
        }
    }
    return hits;
}

function segmentHitsAnyTrafficLight(a, b) {
    const midPoint = [(a[^0] + b[^0]) / 2, (a[^1] + b[^1]) / 2];
    for (const t of trafficLights) {
        if (haversine(t.pos, midPoint) <= TRAFFIC_LIGHT_AVOID_RADIUS_M) {
            return true;
        }
    }
    return false;
}
```

Генерація альтернативних точок.

```
function generateAlternativePoints(start, end, trafficLights) {
    const alternatives = [];
    const mainBearing = Math.atan2(end[^1] - start[^1], end[^0] -
start[^0]);
    const distance = haversine(start, end);

    const offsets = [
        -ALTERNATIVE_SEARCH_RADIUS * 0.7,
        ALTERNATIVE_SEARCH_RADIUS * 0.7,
        -ALTERNATIVE_SEARCH_RADIUS * 1.4,
```

```
        ALTERNATIVE_SEARCH_RADIUS * 1.4,
        -ALTERNATIVE_SEARCH_RADIUS * 2.1,
        ALTERNATIVE_SEARCH_RADIUS * 2.1
    ];

    for (let offset of offsets) {
        const perp = mainBearing + Math.PI / 2;
        const altStart = [
            start[^0] + Math.sin(perp) * offset,
            start[^1] + Math.cos(perp) * offset
        ];
        const altEnd = [
            end[^0] + Math.sin(perp) * offset,
            end[^1] + Math.cos(perp) * offset
        ];

        // Перевірка відстані від світлофорів
        let avoidsLights = true;
        for (let tl of trafficLights.filter(t => t.status === 'red')) {
            if (haversine(tl.pos, altStart) < 80 || haversine(tl.pos,
altEnd) < 80) {
                avoidsLights = false;
                break;
            }
        }

        if (avoidsLights && haversine(altStart, start) < 150 &&
haversine(altEnd, end) < 150) {
            alternatives.push({
                start: altStart,
                end: altEnd,
                offset: offset,
                distanceFromOriginal: haversine(altStart, start)
            });
        }
    }
```

```
    return alternatives.sort((a, b) => a.distanceFromOriginal -
b.distanceFromOriginal);
}
```

Управління станом світлофорів.

```
let trafficLights = [
    { name:'t1', pos:[50.002994846659156, 36.21814727783204],
status:'red' },
    { name:'t2', pos:[50.000015711174115,36.22089385986329], status:'red'
},
    // ... інші світлофори
];

function drawTrafficLights() {
    trafficLayer.clearLayers();
    trafficLights.forEach(t => {
        const icon = t.status === 'red' ? redLightIcon : greenLightIcon;
        const marker = L.marker(t.pos, { icon: icon
}).addTo(trafficLayer);

        marker.bindPopup(`<b>Світлофор ${t.name}</b><br>Статус:
${t.status === 'red' ? 'Червоний' : 'Зелений'}`);

        L.circle(t.pos, {
            radius: TRAFFIC_LIGHT_AVOID_RADIUS_M,
            color: t.status === 'red' ? 'red' : 'limegreen',
            weight: 1,
            fillColor: t.status === 'red' ? 'red' : 'limegreen',
            fillOpacity: 0.1
        }).addTo(trafficLayer);
    });
}

// Циклічна зміна станів кожні 8 секунд
setInterval(() => {
```

```
    trafficLights.forEach(t => t.status = (t.status === 'red' ? 'green' :
'red'));
    drawTrafficLights();
    resumePausedCars();
}, 8000);
```

Відображення маршрутів на карті. Після розрахунку маршрути відображаються як кольорові полілінії:

```
// Маршрут A*
const coords = cars.A.routeData.geometry.coordinates.map(c => [c[^1],
c[^0]]);
L.polyline(coords, { color: 'blue', weight: 6, opacity: 0.9
}).addTo(layerA);

// Маршрут Dijkstra
const coords = cars.D.routeData.geometry.coordinates.map(c => [c[^1],
c[^0]]);
L.polyline(coords, { color: 'orange', weight: 6, opacity: 0.9
}).addTo(layerD);
```

Оновлення інформаційних панелей.

```
function renderRoutePanel(container, resp, strategy, additionalInfo = "",
carId = null) {
    container.innerHTML = '';
    const r = resp.routes[^0];

    const totalDelay = carId ? cars[carId].totalTrafficDelay : 0;
    const totalTime = r.duration + totalDelay;

    const summary = document.createElement('div');
    summary.className = 'summary';
    summary.innerHTML = `
        <b>Відстань:</b> ${(r.distance/1000).toFixed(1)} км
          <b>Час:</b> ${Math.floor(totalTime/60)}хв
${Math.round(totalTime%60)}с
```

```
        ${getStrategyBadge(strategy)}
        ${additionalInfo}
    `;
    container.appendChild(summary);

    const stepsContainer = document.createElement('div');
    stepsContainer.className = 'steps';
    let stepCounter = 0;

    r.legs.forEach(leg => {
        leg.steps.forEach(st => {
            stepCounter++;
            const div = document.createElement('div');
            div.className = 'step';
            div.innerHTML = `
                <div class="num">${stepCounter}</div>
                <div style="flex:1">
                    <div style="display:flex;gap:8px;align-items:center">
                        <div
class="icon">${maneuverIcon(st.maneuver)}</div>
                        <div
class="text">${getDetailedInstruction(st)}</div>
                    </div>
                    <div class="meta">~${Math.round(st.distance)} м,
${Math.round(st.duration)} c</div>
                </div>
            `;
            stepsContainer.appendChild(div);
        });
    });

    container.appendChild(stepsContainer);
```

Псевдокод алгоритму A*

```
function A_Star(graph, start, goal):
    openSet = {start}
```

```
    cameFrom = empty map

    gScore = map with default value of Infinity
    gScore[start] = 0

    fScore = map with default value of Infinity
    fScore[start] = h(start)

    while openSet is not empty:
        current = node in openSet with lowest fScore value

        if current == goal:
            return reconstruct_path(cameFrom, current)

        openSet.remove(current)

        for each neighbor of current:
            tentative_gScore = gScore[current] + distance(current, neighbor)

            if tentative_gScore < gScore[neighbor]:
                cameFrom[neighbor] = current
                gScore[neighbor] = tentative_gScore
                fScore[neighbor] = gScore[neighbor] + h(neighbor)

                if neighbor not in openSet:
                    openSet.add(neighbor)
    return failure
```

Асинхронні запити до OSRM API

```
async function osrmRoute(waypoints, timeoutMs = 10000) {
    const coords = waypoints.map(w => `${w[^1]},${w[^0]}`).join(';');
    const url =
`${OSRM_BASE}/route/v1/driving/${coords}?overview=full&geometries=geojson
&steps=true&annotations=duration,distance`;

    const controller = new AbortController();
    const id = setTimeout(() => controller.abort(), timeoutMs);
```

```
    try {
        const resp = await fetch(url, { signal: controller.signal });
        clearTimeout(id);
        if (!resp.ok) throw new Error('OSRM error: ' + resp.status);
        return await resp.json();
    } catch(err) {
        clearTimeout(id);
        throw err;
    }
}
```