Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут
комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: *«Інтелектуальна система оптимізації логістики доставки замовлень меблевої компанії»*

Виконав: студент 6 курсу групи КН-63м
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Карнафель О. Б.
(прізвище та ініціали)

Керівник Флуд Л. О.
(прізвище та ініціали)

Рецензент Косновський О. Т.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України

(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій

Кафедра комп'ютерних наук

Рівень вищої освіти другий (магістерський)

Спеціальність 122 "Комп'ютерні науки"

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри КН

Борецька І. Б.

"10" грудня 2025 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Карнафель Ользі Богданівній

(прізвище, ім'я, по батькові)

1. Тема роботи *Інтелектуальна система оптимізації логістики доставки замовлень меблевої компанії*

керівник роботи *Флуд Любомир Олегович, к. т. н., доцент*

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

затверджені наказом вищого навчального закладу від "29" квітня 2025 року № С-288

2. Термін подання студентом роботи 10 грудня 2025 р.

3. Вихідні дані до роботи Розробити веб-систему для оптимізації логістики меблевої компанії, яка будує маршрути з урахуванням трафіку та дорожніх обмежень. Система має працювати з даними замовлення, адресами доставки та координатами складів.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Стан проблемної області

Інформаційне забезпечення

Математичне забезпечення

Програмне забезпечення

Розроблення стартап-проєкту

Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Підготовка матеріалів до доповіді

6. Дата видачі завдання 1 травня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	*Аналіз предметної області*	01.05.25 – 03.05.25	Виконано
2	*Формування вимог, постановка задачі та визначення архітектури системи*	04.05.25 – 10.06.25	Виконано
3	*Вибір технологій та інструментарію для реалізації*	11.07.25 – 15.07.25	Виконано
4	*Розроблення інформаційного забезпечення*	16.07.25 – 01.08.25	Виконано
5	*Розроблення математичного забезпечення*	02.08.25 – 10.08.25	Виконано
6	*Реалізація Python-модуля маршрутизації з урахуванням трафіку та геоданих*	11.08.25 – 01.09.25	Виконано
7	*Реалізація серверної частини застосунку*	02.09.25 – 01.10.25	Виконано
8	*Оформлення пояснювальної записки*	02.10.25 – 20.11.25	Виконано

Студент _____ Карнафель О. Б.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Флуд Л. О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська робота містить 81 сторінки пояснювальної записки, 14 рисунків, 17 формул, 1 таблиця, 2 додатки, 20 джерел.

У даній роботі розроблено програмне й алгоритмічне забезпечення інтелектуальної веб-системи оптимізації логістики доставки меблевої компанії. Створена веборієнтована платформа забезпечує автоматичне формування оптимальних маршрутів доставки з урахуванням дорожнього трафіку, обмежень для вантажного транспорту та раціонального порядку відвідування адрес. Для реалізації функціональності використано картографічні дані OpenStreetMap, технології обробки дорожньої інформації TomTom Traffic API, алгоритми пошуку найкоротших шляхів та методи розв'язання задачі комівояжера. Система інтегрує серверну логіку на Node.js, базу даних MongoDB та модулі аналітичної обробки даних на Python.

Ключові слова: логістика, оптимізація маршрутів, OpenStreetMap, TomTom Traffic API, TSP, Node.js, MongoDB, Python.

ABSTRACT

The thesis contains 81 pages of explanatory note, 14 figures, 17 formulas, 1 table, 2 appendices, 20 sources.

In this work, the software and algorithmic support of an intelligent web system for optimizing the logistics of delivery of a furniture company has been developed. The created web-based platform provides automatic formation of optimal delivery routes taking into account road traffic, restrictions for freight transport and a rational order of visiting addresses. To implement the functionality, OpenStreetMap map data, TomTom Traffic API road information processing technologies, shortest path search algorithms and methods for solving the traveling salesman problem have been used. The system integrates server logic on Node.js, the MongoDB database and analytical data processing modules on Python.

Keywords: logistics, route optimization, OpenStreetMap, TomTom Traffic API, TSP, Node.js, MongoDB, Python.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити вебзастосунок, основна функція якого полягає в забезпеченні можливості онлайн-оформлення замовлень на доставку меблів із подальшим керуванням логістичними процесами.

Для реалізації цього завдання необхідно створити серверну та клієнтську частини веб-додатку. Серверна частина має бути побудована з використанням технологій Node.js та Express.js, що забезпечить обробку HTTP-запитів, маршрутизацію, обробку даних замовлень та інтеграцію з базою даних. Клієнтська частина повинна забезпечити користувачам зручний інтерфейс для створення нових замовлень та відстеження маршрутів доставки товарів.

У рамках проєкту необхідно також реалізувати систему повідомлень – надсилання листів на електронну пошту користувачам після створення замовлення. Вся система повинна бути реалізована у вигляді монолітного застосунку з модульною архітектурою, що передбачає подальше масштабування проєкту та можливість інтеграції додаткових функцій у майбутньому.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

REST API – інтерфейс, призначений для обміну інформацією між системами

HTTP – протокол передавання даних

JSON – формат обміну даними

JS – JavaScript

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

NoSQL – non-SQL або нереляційна база даних

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

OSM – OpenStreetMap, відкрита картографічна платформа

ETA – Estimated Time of Arrival, орієнтовний час прибуття

TSP – Traveling Salesman Problem, задача комівояжера

DB – Database, база даних

ВСТУП

У сучасних умовах інтенсивного розвитку електронної комерції та зростання обсягів онлайн-замовлень логістичні процеси компаній потребують автоматизації та цифрової трансформації. Особливо це актуально для меблевих компаній, які щодня обробляють значну кількість замовлень на доставку габаритної продукції. Для забезпечення ефективної роботи таких компаній важливо мати зручний інструмент для управління логістикою, оформленням замовлень та відстеженням процесу доставки.

Об'єктом дослідження є процес цифрової організації логістики доставки меблів.

Предметом дослідження є побудова веб додатку, що автоматизує обробку замовлень на доставку меблів та забезпечує інтерактивну взаємодію з користувачем через веб інтерфейс.

Метою роботи є розробка інтелектуальної веб-системи для оптимізації логістики доставки меблевої компанії, що дозволяє реєструвати замовлення, переглядати їх.

Завдання полягає у створенні веб системи для логістики доставки меблів, яка дозволяє додавати та переглядати та замовлення на доставку, а також відслідковувати їх статус у зручному інтерфейсі. Необхідно реалізувати серверну частину на базі Node.js і Express для обробки запитів, спроєктувати структуру даних для зберігання інформації про замовлення, а також створити клієнтський інтерфейс, що забезпечить швидку та інтуїтивну взаємодію з користувачем. Система має бути масштабованою, придатною для використання в умовах реального підприємства.

Наукова новизна полягає у створенні компактної веб системи, яка поєднує класичний стек веб технологій із сучасним підходом до обробки логістичної інформації без використання складних фреймворків.

Практичне значення ро полягає у створенні універсального інструменту, що може бути впроваджений у діяльність меблевих компаній для оптимізації та цифровізації процесів обробки замовлень і доставки товарів.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Логістика доставки у меблевій галузі є ключовим етапом усього бізнес-процесу, який визначає не лише рівень операційної ефективності підприємства, а й ступінь задоволеності клієнтів. Доставка меблів має суттєві особливості, оскільки меблеві вироби є великогабаритними, важкими, часто крихкими, потребують спеціального транспорту та уважного ставлення під час транспортування. Унаслідок цього правильне планування маршруту безпосередньо впливає на витрати компанії, довговічність транспорту, тривалість виконання кожного замовлення та загальну репутацію сервісу. Чим точніше спланована логістика — тим нижча собівартість доставки, вища швидкість обслуговування й стабільніший прибуток компанії.

У традиційній моделі меблеві компанії зазвичай використовують ручне планування маршрутів або покладаються на загальні навігаційні рішення, такі як Google Maps чи Waze. Хоча ці сервіси зручні для повсякденного користувача, вони не розраховані на складні логістичні сценарії, пов’язані з обслуговуванням численних клієнтів та обмеженнями вантажного транспорту [1]. Навіть досвідчені логісти, які роками працюють у певному місті, не можуть врахувати всі фактори, особливо коли кількість адрес перевищує три–чотири точки. У таких умовах людський фактор стає критичним: маршрути будуються інтуїтивно або на основі неповної інформації, що призводить до зайвих витрат на пальне, збільшення тривалості доставки та перевантаження водіїв.

Ще один важливий аспект — реальний дорожній трафік. У великих містах дорожня ситуація постійно змінюється: утворюються затори, з’являються ремонтні роботи, ДТП, тимчасові перекриття. У пікові години час проїзду тієї самої ділянки може збільшуватися у 2–4 рази порівняно з періодами низького завантаження. Традиційні підходи не враховують цих динамічних змін. Це призводить до того, що планування маршруту виконується на основі середніх показників, які не відповідають реальній ситуації на дорозі. У результаті час доставки відхиляється від

прогнозованого, а клієнти отримують замовлення із затримкою, що негативно впливає на їхню довіру до компанії та репутацію сервісу загалом.

Оптимізація логістики у багатоточкових маршрутах є окремою складною задачею. Якщо компанія має доставити меблі одразу до декількох клієнтів, визначення оптимального порядку відвідування адрес стає математичною проблемою високої складності, відомою як задача комівояжера (TSP). При зростанні числа точок кількість можливих комбінацій збільшується експоненційно, і ручне планування перестає бути ефективним [2]. Базові інструменти навігації також не містять функцій оптимізації черги адрес, що робить їх непридатними для реальних логістичних завдань меблевої компанії.

Комплексність проблеми посилюється необхідністю поєднання кількох критеріїв оптимізації. Залежно від ситуації логісту може бути важливо знайти не лише найкоротший маршрут, але й найшвидший або найдоступніший з точки зору комфортності дороги. Різні меблі мають різну вразливість до пошкоджень, тому дороги з поганим покриттям можуть бути небажаними. Водночас у годину пік навіть найкоротший маршрут може бути значно повільнішим через затори, що робить актуальним пошук найшвидшого шляху з урахуванням трафіку. Загальнодоступні навігаційні сервіси не підтримують багатокритеріальну маршрутизацію, що обмежує можливості їхнього використання у професійній логістиці.

Таким чином, аналіз предметної області показує, що сучасна меблева компанія потребує інтелектуального програмного забезпечення, здатного автоматично моделювати дорожній граф, враховувати обмеження вантажного транспорту, обробляти реальні дані про дорожній трафік, оптимізувати чергу доставки та формувати прогнозовані показники часу прибуття. Без такого інструменту логістика компанії суттєво втрачає у ефективності, що негативно впливає як на фінансові результати підприємства, так і на рівень сервісу для клієнтів. Створення інтелектуальної веб-системи, здатної вирішувати перелічені задачі, постає як необхідний крок у розвитку логістичних процесів меблевих компаній в Україні.

1.2 Постановка задачі

У діяльності сучасних меблевих компаній однією з ключових операцій є організація доставки готової продукції до клієнтів. Логістика доставки нерозривно пов’язана зі швидкістю виконання замовлень, якістю сервісу, витратами на транспорт та ефективністю роботи всього підприємства. З розвитком електронної комерції та збільшенням кількості індивідуальних замовлень навантаження на відділи логістики значно зросло, що висунуло нові вимоги до швидкості і точності планування маршрутів [3].

Проблема оптимізації доставки меблів має свої унікальні особливості. Меблі - це великогабаритні об’єкти, які часто потребують транспортування спеціальними вантажними автомобілями, що накладає обмеження на можливість пересування певними вулицями. Таким чином, при плануванні маршруту система повинна враховувати:

- висотні та вагові обмеження, що містяться в інфраструктурі міста;
- односторонній рух, який може бути в критичних районах (центр Львова, наприклад);
- геометрію вулиць, вузькі провулки, круті повороти;
- транспортні вузли, що можуть бути перевантаженими у певні періоди доби.

Особливо актуальною задача логістичної оптимізації постає в умовах українських міст, дорожня інфраструктура яких має низку специфічних особливостей:

- нерівномірний дорожній трафік, з піками у години "пік";
- наявність великої кількості ремонтів, перекриттів та аварій, що здатні суттєво збільшити час руху;
- історична забудова, яка передбачає вузькі центральні вулиці, непридатні для пересування вантажівок;
- обмежені альтернативні маршрути, особливо в спальних районах та приватному секторі;
- недостатність інформації про реальні швидкості руху дорогами, що ускладнює планування часу доставки.

Традиційні способи побудови маршрутів, які застосовуються більшістю компаній (ручне планування, використання окремих навігаторів або стандартних навігаційних сервісів), не здатні повноцінно врахувати всі вищенаведені фактори. Вони також не забезпечують можливості вирішувати такі задачі, як:

- одночасна оптимізація кількох адрес доставки;
- мінімізація часу, а не лише довжини маршруту;
- обходження доріг, що непридатні для вантажівки;
- автоматичне визначення порядку відвідування клієнтів;
- інтеграція системи у веб-інтерфейс компанії.

Таким чином, виникає потреба у створенні інтелектуальної веб-системи, яка здатна на основі дорожнього графа, реальних даних про трафік та логістичних особливостей меблевої галузі побудувати маршрут, максимально адаптований до реальних умов.

1.3 Обґрунтування актуальності

Сучасний ринок меблевої продукції характеризується високою конкуренцією, активним зростанням частки онлайн-покупок та підвищеними вимогами клієнтів до швидкості й надійності доставки. Для меблевих компаній логістика є критичною частиною операційної діяльності, адже саме етап транспортування часто стає найбільш витратним та найменш оптимізованим. З огляду на це розробка інтелектуальної веб-системи, здатної автоматизувати й оптимізувати процеси доставки, набуває особливої актуальності [4].

Однією з ключових причин, що обумовлюють потребу у створенні такого програмного забезпечення, є значні часові втрати, які виникають через недосконаліть традиційних підходів до планування маршрутів. Меблі — це великогабаритна і тяжка продукція, що потребує спеціальних транспортних засобів та ретельного маршрутування. Неефективно вибраний маршрут може збільшити час доставки на 20–40%, що призводить до перенавантаження водіїв, порушення графіку, погіршення якості обслуговування та, як наслідок, — до незадоволеності клієнтів. У сучасних умовах, коли швидкість доставки стає одним із ключових факторів

конкурентної боротьби, скорочення часу транспортування є важливим стратегічним завданням для компаній [5].

Не менш вагомою є економічна складова. Через швидке зростання вартості пального логістичні витрати меблевих компаній з кожним роком збільшуються. Ручне або інтуїтивне прокладання маршрутів часто призводить до надлишкових пробігів, що не лише збільшує витрати на пальне, а й пришвидшує зношення транспортних засобів, потребу у більш частому технічному обслуговуванні та вищому ризику поломок. У таких умовах інтелектуальна система, здатна знаходити мінімальну відстань, мінімізувати час поїздки або вибирати найбільш комфортний шлях, стає важливим важелем економії для підприємства.

Окремо слід відзначити специфіку української дорожньої інфраструктури, особливо у великих містах — таких як Львів, Київ, Харків, Одеса. У багатьох українських населених пунктах дорожня мережа включає велику кількість вулиць з обмеженнями для вантажного транспорту: низькі арки, вузькі проїзди, обмеження маси або довжини транспортного засобу, тимчасові перекриття. Більшість доступних навігаційних сервісів не враховують ці обмеження під час побудови маршрутів. Таким чином, використання вантажних фільтрів на основі даних OpenStreetMap (OSM), а також їхнє програмне опрацювання у власній системі є критично необхідним для практичного застосування у меблевій логістиці.

Додатковим фактором, що підсилює актуальність розробки, є потреба в урахуванні реального стану дорожнього руху. Доступні українським компаніям системи здебільшого використовують середні швидкості без врахування трафіку, аварій, ремонтів та заторів. Інтеграція із сервісами реального часу, зокрема TomTom Traffic API, дозволяє формувати найшвидший маршрут у поточних дорожніх умовах. В умовах густої міської інфраструктури це може скоротити час поїздки на десятки хвилин, що має безпосередній вплив на ефективність логістики.

Важливим є і той факт, що на ринку України практично відсутні комплексні локальні інтелектуальні рішення, орієнтовані на логістику меблевих компаній. Більшість наявних сервісів або не адаптовані до української інфраструктури, або не надають можливості повної кастомізації маршрутизації, або є занадто дорогими для

впровадження у середньому бізнесі. У цьому контексті створення власної веб-системи з використанням відкритих та безкоштовних джерел даних (OSM), оптимізованих алгоритмів, сучасних інструментів аналізу та побудови маршрутів є економічно виправданим і технологічно перспективним рішенням.

Крім того, система надає можливість застосовувати алгоритми оптимізації черги доставок (TSP), що забезпечує знаходження ефективного порядку відвідування точок. Це дозволяє скорочувати сумарну відстань маршруту та час доставки в умовах багатоточкової логістики — типового сценарію для меблевих компаній. Наявність декількох критеріїв оптимізації (найкоротший маршрут, найшвидший з урахуванням трафіку, найбільш комфортний) робить систему гнучкою й універсальною.

З огляду на поєднання технічних, економічних та організаційних факторів, розробка інтелектуальної веб-системи для оптимізації логістики доставки меблів є актуальною, своєчасною та важливою для підвищення конкурентоздатності українських меблевих компаній. Така система здатна суттєво зменшити витрати підприємства, підвищити швидкість доставки, покращити якість обслуговування клієнтів та забезпечити ефективне використання транспортних ресурсів на основі сучасних технологій маршрутизації та аналізу дорожньої інформації.

Висновки до розділу

У ході аналізу предметної області встановлено, що логістика меблевих компаній стикається з низкою специфічних проблем: складні міські умови, велика кількість зупинок, залежність від типу транспорту, обмеження для вантажівок, неповні або застарілі дані навігаційних сервісів, а також потреба у врахуванні реального трафіку, якості доріг і маршрутизації з багатьма точками. На основі цього сформульовано задачу створення інтелектуальної веб-системи, яка забезпечує побудову декількох альтернативних маршрутів (найкоротший, найшвидший, найкомфортніший), оптимізує порядок доставки за допомогою TSP, враховує вантажні обмеження, оперативний дорожній рух (TomTom Traffic), якість покриття та забезпечує клієнта інструментами відстеження. Актуальність цієї розробки визначається тим, що існуючі аналоги або надто дорогі, або не адаптовані до українських дорожніх реалій, а використання відкритих даних OpenStreetMap у

поєднанні з оптимізованими алгоритмами маршрутизації дозволяє створити доступну, точну та продуктивну систему, яка може суттєво зменшити витрати на доставку та підвищити ефективність роботи меблевих компаній в Україні.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Загальна характеристика інформаційного забезпечення

Інформаційне забезпечення інтелектуальної веб-системи оптимізації логістики доставки меблевої компанії являє собою комплекс узгоджених структур даних, методів їхньої обробки, а також каналів їхнього переміщення між окремими компонентами застосунку. Усі елементи системи — від механізму отримання користувацьких замовлень до побудови складних маршрутів із врахуванням трафіку — базуються на якісно організованій інформаційній інфраструктурі, що забезпечує цілісність, актуальність та доступність даних у будь-який момент часу.

У системі використовуються різнорідні інформаційні ресурси, кожен з яких виконує власну функцію у процесі маршрутизації та обслуговування замовлень. Основу становлять геопросторові дані OpenStreetMap, які містять структуру дорожньої мережі, її геометрію, атрибути та обмеження. Їх застосування дозволяє представити місто як орієнтований зважений граф, де вершини є географічними точками, а ребра — дорожніми сегментами із довжиною, допустимою швидкістю руху, кількістю смуг, типом покриття та іншими характеристиками. Для підвищення точності маршрутизації ці дані доповнюються інформацією про реальний стан дорожнього руху, отриманою через TomTom Traffic API, що забезпечує динамічний трафіковий коефіцієнт, рівень завантаженості дороги, індекс заторів та інші параметри [6].

Іншим важливим джерелом інформації є вхідні дані користувача — адреси доставки, контактні дані, вибір типу транспортного засобу, додаткова інформація про замовлення. Ці дані проходять процедуру геокодування через Nominatim API, що забезпечує перетворення текстових адрес у координати. Таким чином система будує логічний місток між описом точки, введеним користувачем, і реальними об’єктами дорожньої мережі.

З метою забезпечення оптимальної обробки інформації система використовує декілька модулів, відповідальних за аналіз, трансформацію та зберігання даних. На серверній частині функціонує набір алгоритмів, що працюють із графовими

структурами: визначення найближчих доступних вузлів, побудова локальних сегментів маршруту, застосування алгоритмів A*, Dijkstra, а також реалізація оптимізації TSP через комбінацію nearest-neighbor та 2-opt. Особливу роль відіграє інформаційне кешування, яке дозволяє зменшити кількість повторних запитів до зовнішніх сервісів та прискорити генерацію результатів.

Інформаційне забезпечення також охоплює модель взаємодії між фронтендом і бекендом. Сервер формує форматовані JSON-структури, які включають координати, мітки, ETA, довжину маршруту та інші характеристики; клієнтська частина інтерпретує ці дані та відображає їх на інтерактивній карті за допомогою Leaflet.js, забезпечуючи користувачу можливість перемикання типів маршрутів і перегляду додаткових властивостей кожного з них.

Комплексно організована інформаційна система гарантує, що всі процеси — від моменту введення замовлення користувачем до побудови оптимальних маршрутів — узгоджено функціонують як єдиний інформаційний механізм. Це дозволяє обробляти складні логістичні задачі, забезпечувати точність планування, враховувати актуальний стан доріг та оперативно формувати найкращі рішення для доставки меблевих замовлень.

2.2 Вхідні та вихідні дані системи

Вхідні дані інтелектуальної системи оптимізації логістики складаються з кількох інформаційних потоків, що надходять як від користувачів, так і від зовнішніх сервісів. Основним джерелом є дані, введені клієнтом через веб-інтерфейс: ім’я, назва компанії, контактні дані, місто доставки, перелік адрес і вибір типу транспортного засобу. Ця інформація має неструктурований текстовий формат, тому система виконує додаткове перетворення — зокрема, геокодування адрес за допомогою Nominatim API, що дозволяє перевести введені користувачем текстові записи у координати реальних географічних точок [6]. Отримані координати надалі використовуються для знаходження найближчих вузлів у дорожньому графі OpenStreetMap.

Другий інформаційний потік формується на основі геопросторових даних OpenStreetMap, з яких під час ініціалізації система будує топологічну модель

дорожньої мережі. Кожен дорожній сегмент у графі містить набір атрибутів: довжину, тип дороги, кількість смуг, поверхню, дозволену швидкість та обмеження руху (maxweight, maxheight, hgv, oneway). Саме ці дані визначають можливість використання сегмента певним типом транспортного засобу, а також впливають на результати вибору оптимального маршруту. На основі цих атрибутів система формує три вагові моделі: мінімальну за довжиною, мінімальну за часом (з урахуванням швидкості та трафіку) та мінімальну за комфортністю, де враховано якість поверхні, кількість смуг і складність дорожніх елементів.

Третім джерелом вхідної інформації є динамічні дані про дорожній рух, отримані через TomTom Traffic API. Цей сервіс забезпечує доступ до реальної ситуації на дорогах: швидкості руху, індексу заторів, коефіцієнтів уповільнення, а в разі розширення — інформацію про ДТП та ремонтні роботи. Трафікові дані інтегруються у вагову модель найшвидшого маршруту і дозволяють синхронізувати побудовані маршрути з фактичним станом транспортного потоку. Для підвищення продуктивності система застосовує кешування результатів запитів до TomTom, оскільки отримання трафіку в режимі реального часу є найбільш ресурсоємним процесом [7].

Вихідні дані системи формуються у структурованому форматі JSON і передаються на клієнтську частину. До них входять три маршрути: найкоротший, найшвидший та найкомфортніший, кожен із яких містить послідовність координатних точок разом із відповідними мітками ключових локацій. Крім того, сервер обчислює та повертає довжину кожного маршруту, його орієнтовний час проходження (ETA), а також допоміжні параметри, необхідні для коректної візуалізації. На основі цих даних користувач отримує можливість перемикатися між різними типами маршрутів, переглядати час і відстань для кожного з них та бачити ключові точки доставки на мапі.

Завдяки чіткій структурованості вхідних та вихідних даних веб-система забезпечує точність роботи алгоритмів маршрутизації, стабільність обміну інформацією між модулями, а також зручність і зрозумілість для кінцевого користувача. Інтеграція даних із різних джерел — користувацьких форм,

геопросторових моделей OSM та динамічних трафікових API — дає можливість формувати маршрути, що максимально враховують реальні умови транспортної інфраструктури міста.

2.3 Використані програмні інструменти, фреймворки та технології

У процесі створення інтелектуальної веб-системи для оптимізації логістики доставки меблів застосовано комплекс програмних засобів і технологій, які забезпечують коректну обробку геопросторових даних, реалізацію алгоритмів маршрутизації, взаємодію з користувачем та зберігання інформації про замовлення. Вибір кожного інструмента обумовлений його відповідністю поставленим технічним вимогам, можливістю інтеграції з іншими компонентами та ефективністю виконання відповідних функцій у складі системи.

Серверна частина реалізована мовою Python 3.10, яка надає розвинений набір бібліотек для аналітичних обчислень та роботи з графами. Для побудови HTTP-інтерфейсу, прийому запитів від клієнта та формування відповідей застосовується фреймворк Flask, що дозволяє гнучко організувати REST-архітектуру та відокремити логіку маршрутизації від мережевої логіки [8]. Основним інструментом роботи з дорожніми графами є OSMnx, який забезпечує завантаження карти OpenStreetMap, аналіз її атрибутів, побудову структур «вузол–ребро» та формування зважених моделей дорожньої мережі. Для обчислення найкоротших, найшвидших і комфортних маршрутів використовується бібліотека NetworkX, яка надає реалізації алгоритмів A*, Dijkstra, двонаправленого пошуку, а також дозволяє інтегрувати власні вагові функції та евристики. Усі взаємодії зі сторонніми сервісами виконуються через бібліотеку Requests, що забезпечує обмін даними з Nominatim API та TomTom Traffic API [9].

Клієнтська частина системи побудована на основі HTML, CSS та JavaScript, що гарантує доступність і коректне відтворення застосунку в будь-якому сучасному браузері. Для відображення карти та маршрутів використовується бібліотека Leaflet.js, яка дозволяє працювати з геопросторовою інформацією у вигляді інтерактивного інтерфейсу — масштабувати карту, додавати маркери, полігони, лінії маршрутів та супутні елементи. Для позначення напрямків руху використовується

доповнення leaflet-polyline-decorator, яке дозволяє наносити стрілки на полілайн-маршрути. Така комбінація інструментів забезпечує наочне подання результатів обчислень та дозволяє користувачу переглядати різні варіанти маршрутів у зручному форматі.

Дані про замовлення та геокодовані адреси зберігаються у MongoDB, що є документоорієнтованою базою даних і дозволяє гнучко працювати з об’єктами, структура яких може відрізнятися залежно від кількості адрес чи додаткових параметрів. Для взаємодії серверної логіки Node.js із базою даних використовується ORM-бібліотека Mongoose, яка забезпечує опис схем колекцій, перевірку валідності даних та стандартизацію структури документів. Завдяки такому рішенню система легко масштабується та дозволяє швидко виконувати операції пошуку, оновлення чи додавання інформації.

Зовнішні геосервіси відіграють важливу роль у формуванні коректних вихідних даних. Сервіс Nominatim API здійснює геокодування введених текстових адрес, перетворюючи їх у координати, які можуть бути зіставлені з вузлами графа OpenStreetMap. Сервіс TomTom Traffic API надає інформацію про фактичний стан дорожнього руху, включно зі швидкістю руху на сегменті, рівнем заторів та коефіцієнтами уповільнення. Дані цього сервісу інтегруються у вагові моделі маршрутизації, дозволяючи будувати маршрути не лише з урахуванням статичної структури дороги, а й відповідно до реальної транспортної ситуації.

Застосований набір інструментів та технологій забезпечує узгоджену роботу всіх компонентів системи: отримання та обробку вхідних даних, побудову графа дорожньої мережі, виконання алгоритмів оптимізації, інтеграцію з зовнішніми джерелами інформації та відображення результатів користувачу. У сукупності вони формують надійну програмну платформу для розв’язання завдань логістичної оптимізації в межах меблевої компанії.

2.4 Джерела та структура текстових даних

Інтелектуальна логістична веб-система використовує широкий спектр текстових даних, що надходять із різних джерел та забезпечують коректну роботу механізмів геокодування, побудови маршрутів, оптимізації та взаємодії між

компонентами. Текстові записи формують основу як користувацького вводу, так і метаданих дорожньої інфраструктури, зібраних через OpenStreetMap та зовнішні API. Вони є необхідними для коректної інтерпретації адрес, доріг, характеристик покриття та обмежень руху.

Основною групою текстових даних є інформація, яку вводить користувач під час створення замовлення. До неї належать ім’я та прізвище замовника, назва компанії, місто доставки, список адрес, контактні дані та додаткові коментарі. Усі ці значення надходять у текстовому вигляді, перевіряються, нормалізуються та після цього зберігаються у базі у вигляді JSON-документів. Такі дані використовуються як вхідні точки для геокодування та побудови маршруту.

Другим важливим джерелом текстової інформації є Nominatim API, який обробляє введені адреси та повертає їх у структурованому вигляді. Відповідь сервісу включає деталізовані текстові описи місць, повні назви об’єктів, типи локацій та інші відповідні атрибути. Ця інформація дозволяє однозначно визначити координати кожної адреси, навіть якщо користувач вводить її у довільному форматі.

Окремий обсяг текстових даних надходить із OpenStreetMap. Дорожня мережа містить численні текстові атрибути, такі як типи доріг (highway), назви вулиць, типи покриття, правила руху, обмеження для вантажівок та параметри доступу. Ці поля визначають поведінку алгоритмів маршрутизації та впливають на формування трьох типів маршрутів — найкоротшого, найшвидшого та найкомфортнішого. Зміст текстових тегів напряму впливає на використання вагових коефіцієнтів.

Також важливим джерелом текстових даних є TomTom Traffic API, що постачає інформацію про фактичний стан доріг. Хоча більшість значень у відповіді API мають числовий характер, частина метаданих представлена у текстовій формі — наприклад, статус дорожньої ситуації, тип затору, опис інцидентів та повідомлення про перекриття руху. Ці дані використовуються системою для врахування динамічного трафіку під час розрахунку часу в дорозі.

У межах системи текстові дані зберігаються переважно у форматі JSON, що забезпечує гнучкість структури та зручну передачу між клієнтом і сервером. У базі даних MongoDB вони також представлені у вигляді вкладених документів, що

дозволяє зберігати адреси, статуси, властивості замовлень та службову інформацію без потреби у суворій схемі. Окрім цього, у графовій структурі OSMnx/NetworkX текстові атрибути є невід'ємною частиною ребер та вузлів дорожньої мережі.

Обробка текстових даних включає кілька етапів: нормалізацію, валідацію та стандартизацію. Нормалізація дозволяє перетворити введені адреси у формати, придатні до геокодування, а валідація забезпечує коректність контактних даних та інших полів. Стандартизація необхідна для приведення отриманих даних до уніфікованих структур, які можуть використовуватися в алгоритмах оптимізації.

2.5 Інформаційні процеси, обробки та взаємодії компонентів системи

Функціонування інтелектуальної логістичної системи базується на послідовній взаємодії між програмними компонентами, які виконують різні етапи обробки даних — від введення користувачем інформації до побудови оптимізованого маршруту та відображення його на карті. Ці процеси охоплюють збір, передавання, фільтрацію, маршрутизацію, оптимізацію та візуалізацію даних, а також інтеграцію із зовнішніми джерелами інформації про дорожні умови. Узгоджена робота цих елементів забезпечує коректність системи, її продуктивність та точність побудови логістичних сценаріїв.

Користувач вводить інформацію про замовлення, включаючи контактні дані, перелік адрес та місто доставки. Дані з HTML-форматів трансформуються у JSON-структури, після чого передаються методом HTTP POST до бекенд-сервера. На цьому етапі здійснюється базова валідація обов'язкових полів, а дані переводяться у строгий формат, придатний до подальшої обробки.

Сервер отримує дані замовлення, зберігає їх у базі даних MongoDB та формує запит до Python-модуля маршрутизації. Тут відбувається трансформація текстових адрес у формат, придатний для геокодування, а також ініціалізація маршрутизатора з параметрами (місто, координати, вантажний режим).

Кожна введена користувачем текстова адреса перетворюється на точні географічні координати (довгота/широта). Отримані координати кешуються, що мінімізує повторні запити та пришвидшує повторні обчислення. На основі координат формується набір точок, які є стартовими даними для маршрутизації та оптимізації.

Дані OpenStreetMap завантажуються у вигляді графа за допомогою бібліотеки OSMnx. Кожне ребро містить атрибути дороги: довжину, тип, покриття, кількість смуг, швидкісні обмеження, умови доступу для вантажного транспорту. У процесі обробки граф розширюється розрахованими полями — прогнозованою швидкістю руху, часом подолання сегмента, комфортністю. Для вантажного режиму граф додатково фільтрується відповідно до обмежень на масу, висоту й доступність.

Для кожного сегмента між двома точками маршруту система виконує один оптимізований запит до TomTom Traffic API, який повертає дані про актуальний потік, середню швидкість, коефіцієнт заторів і відхилення від вільного руху. Ці дані перетворюються у трафік-фактор, який масштабує час подолання ребер графа. Використання кешування дозволяє суттєво скоротити кількість звернень до API та прискорити маршрутизацію.

На основі отриманих координат будується матриця евклідових відстаней. Для розв’язання задачу товарного мандрівника використовується комбінація алгоритму найближчого сусіда та покращення маршруту методом 2-opt. Це дає змогу знайти ефективну послідовність відвідування точок, не перевантажуючи систему складними оптимізаційними моделями.

Далі відбувається побудова трьох типів маршрутів: найкоротшого, найшвидшого та найкомфортнішого. Залежно від типу маршруту змінюється вагова функція графа: довжина, час з урахуванням трафіку або складність дорожнього покриття. Для кожної пари послідовних точок обчислюється маршрут за допомогою алгоритму A* або двонаправленого Дейкстри з fallback-режимом. Вихідні маршрути об’єднуються у три повноцінні ланцюги, що відповідають різним критеріям оптимізації.

Обчислюються метрики маршруту, таких як загальна відстань, орієнтовний час подолання та комфортність. Ці значення використовуються як аналітична інформація при відображенні на фронтенді. ETA враховує як структуру дорожнього графа, так і зовнішній трафік, що дозволяє отримувати більш реалістичні прогнози.

Далі відбувається повернення результатів на фронтенд та їх візуалізація. Node.js передає клієнту три набори координат маршрутів разом із супутніми метриками.

Інтерфейс використовує бібліотеку Leaflet для відображення полілайнів та маркерів на карті, а PolylineDecorator створює стрілки руху. Користувач може перемикати тип маршруту, і система динамічно перерисовує карту без повторних запитів до сервера. Під картою відображаються ETA та протяжність маршруту, що забезпечує повноцінний інформаційний супровід.

Таким чином, інформаційні процеси системи формують цілісний багатокомпонентний цикл обробки — від текстових даних користувача до аналізу графів, отримання даних про трафік, оптимізації та інтерактивної візуалізації. Взаємодія між компонентами реалізована за модульною архітектурою, що забезпечує масштабованість, можливість удосконалення алгоритмів та інтеграцію нових джерел даних у майбутньому.

Висновки до розділу

У другому розділі було здійснено комплексне дослідження інформаційного забезпечення системи оптимізації логістики меблевих доставок, у межах якого визначено ключові об’єкти обробки, структуру та склад вхідних і вихідних даних, а також їхні функціональні обмеження. Проаналізовано сукупність програмних інструментів, фреймворків і технологій, що утворюють програмну платформу системи, серед яких особливе значення мають OSMnx і NetworkX для роботи з графами доріг, Node.js і Flask для серверної логіки, Leaflet для геовізуалізації та зовнішні сервіси геокодування і трафіку. Розглянуто джерела та структуру текстових даних, включаючи адресні рядки, географічні координати та метадані дорожньої інфраструктури, які формують основу для аналізу та маршрутизації. Окрему увагу приділено інформаційним процесам — від збору даних користувача до багатоступеневої маршрутизації, інтеграції трафіку, оптимізації та відображення маршруту на карті — а також взаємодії між компонентами системи, що забезпечують узгоджений обмін даними і стабільність роботи. Синтез результатів розділу дає змогу чітко окреслити структуру, інформаційні потоки та технічні засоби, необхідні для коректного функціонування розробленої веб-системи.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Теоретичні основи моделювання транспортних мереж

У логістиці доставки замовлень транспортна мережа є основним об'єктом моделювання. Її математичне подання базується на теорії графів, що дає змогу моделювати вулично-дорожню інфраструктуру у вигляді орієнтованого зваженого графа:

$$G = (V, E), \tag{3.1}$$

де вершинами V виступають перехрестя, дорожні розв'язки, мости або визначні транспортні вузли, а ребрами E — дороги між ними. Кожне ребро має пов'язаний набір атрибутів: довжину, допустиму швидкість, кількість смуг, тип дорожнього покриття, обмеження руху для вантажівок, наявність одностороннього руху, наявність світлофорів чи фактори складності маневрів. Таким чином транспортна мережа стає багатошаровою структурою, де кожен атрибут призводить до появи нового критерію для маршрутизації.

З точки зору математичного моделювання, дорожній граф є статичним лише частково: на нього впливає трафік, ремонтні роботи, аварійні ситуації та інші динамічні чинники. Тому в даній роботі статична модель графа доповнюється динамічним параметром — коефіцієнтом сповільнення руху, отриманим за допомогою зовнішнього трафік-API (TomTom). Це переводить задачу побудови маршруту з класичної статичної оптимізації у задачу оптимізації на графах зі змінними вагами.

Важливим аспектом є те, що різні маршрути оптимізуються за різними критеріями: мінімальна довжина шляху, мінімальний час, максимальна комфортність руху. З математичної точки зору це формулює мультикритеріальну оптимізацію, де виконуються три незалежні задачі пошуку шляху із власними ваговими функціями.

3.2 Математична модель ваг та метрик дорожнього графа

Ефективність алгоритмів маршрутизації безпосередньо залежить від того, наскільки коректно та повно описано кожен елемент дорожнього графа. У транспортній мережі ребра графа можуть відрізнятися не лише геометрією, а й

низкою якісних характеристик: станом дорожнього покриття, дозволеною швидкістю, кількістю смуг, наявністю заторів тощо. Саме тому ваги ребер мають бути багатокомпонентними та відображати взаємодію різних факторів, що впливають на рух транспортного засобу.

У даній роботі застосовано чотири незалежні метрики — довжина, час, комфортність та час із врахуванням трафіку. Кожна з них має власну вагову функцію і використовується для побудови одного з трьох типів маршрутів.

Найпростіша та базова метрика ґрунтується на обчисленні довжини ребра:

$$\omega_{dist}(e) = length(e). \tag{3.2}$$

Довжина береться із геометрії вуличного сегмента, наданої OpenStreetMap. Це дає можливість будувати маршрут, який мінімізує суму всіх пройдених відрізків:

$$D(P) = \sum_{e \epsilon P} length(e)\,. \tag{3.3}$$

Такий маршрут є корисним для задач типу "мінімізувати пробіг" або "оптимізувати витрати палива". Проте він часто не враховує реальних умов руху: швидкість може бути низькою, а маневри – складними.

Щоб оцінити реальний час проходження дороги, кожному ребру призначається час на основі його довжини та швидкості:

$$\omega_{time}(e) = \frac{length(e)}{v(e)}, \tag{3.4}$$

де $length(e)$ — довжина дорожнього сегмента в метрах, $v(e)$ — допустима або типова швидкість руху на цьому сегменті.

У випадку відсутності точних даних OSM використовується середня швидкість для кожного типу вулиці [6]. В таблиці 3.1 наведено середні швидкості для кожного типу вулиць.

Таблиця 3.1 – Середні швидкості для кожного типу вулиці в OSM

Тип вулиці	**Середні швидкості**
1	**2**
motorway	90 км/год

Продовження таблиці 3.1

1	2
trunk	80 км/год
primary	70 км/год
secondary	60 км/год
tertiary	50 км/год
residential	30 км/год
service	20 км/год

Загальний час маршруту обчислюється як:

$$T(P) = \sum_{e \epsilon P} \omega_{time}(e)\,. \tag{3.5}$$

Таким чином утворюється маршрут, який оптимізований щодо середньостатистичних умов руху, без урахування змін у трафіку.

На відміну від попередніх метрик, комфортність враховує не лише геометрію та швидкість, а й якість дорожнього сегмента. Базова формула використовується така:

$$\omega_{comfort}(e) = length(e) \cdot k_{surface} \cdot k_{lanes} \cdot k_{urban}, \tag{3.6}$$

де $k_{surface}$ — коефіцієнт стану покриття, k_{lanes} — кількість смуг, k_{urban} — міська щільність.

Комфортний маршрут намагається уникати:

- вузьких вулиць;
- поганих доріг;
- погано освітлених районів;
- щільно забудованих кварталів;
- ділянок, що вимагають частих гальмувань або різких маневрів.

Він мінімізує суму незручних відрізків, тому може бути довшим, але значно плавнішим та передбачуванішим — що важливо при перевезенні меблів.

Оскільки реальний дорожній рух змінюється протягом дня, часу доби, сезону та навіть погодних умов, для точнішої побудови маршрутів використовується динамічний коефіцієнт трафіку, отриманий із TomTom Traffic API:

$$k_{traffic} = \frac{v_{free}}{v_{current}} \cdot (1 + \frac{jamFactor}{10}), \quad (3.7)$$

де v_{free} — швидкість при вільному русі, $v_{current}$ — реальна поточна швидкість; $jamFactor \in [0,10]$ — індекс заторів.

Тоді динамічна вага ребра є:

$$\omega_{traffic}(e) = \omega_{time}(e) \cdot k_{traffic}. \quad (3.8)$$

У підсумку дорожня мережа перетворюється на множину графів, кожен з яких використовує однакову топологію, але різні вагові функції. Це забезпечує незалежне існування:

- найкоротшої траєкторії,
- найшвидшої теоретичної траєкторії,
- найкомфортнішої траєкторії,
- найшвидшої з урахуванням реального трафіку.

Таке математичне розділення метрик робить систему адаптивною, гнучкою та здатною вирішувати різні логістичні сценарії.

3.3 Пошук оптимальних маршрутів у дорожньому графі

Пошук оптимальних шляхів у дорожньому графі є центральним математичним механізмом розробленої системи. Маршрутизація виконується у багатокритеріальному середовищі, де кожне ребро графа має кілька незалежних типів ваг (довжина, час, комфортність, час з трафіком). Це потребує застосування не одного, а декількох алгоритмів пошуку шляху, кожен з яких використовується строго залежно від поставленої задачі.

У роботі застосовано A*, алгоритм Дейкстри, та двонаправлений Дейкстра (Bidirectional Dijkstra) як додатковий механізм пришвидшення. Кожен із цих алгоритмів виконує окрему роль і доповнює інші, формуючи надійну і гнучку математичну основу системи.

Алгоритм A* є модифікацією Дейкстри, що використовує евристику, тому він значно швидший у графах, де є геометричний зміст (зокрема дорожня мережа). Його загальний принцип полягає у мінімізації функції:

$$f(n) = g(n) + h(n), \tag{3.9}$$

де $g(n)$ — фактична вартість шляху від початкової точки до вузла n, $h(n)$ — евристична оцінка відстані від n до цілі.

У дорожньому графі найбільш природною евристикою є евклідова відстань між вузлами:

$$h(n) = \sqrt{\left(x_n - x_{goal}\right)^2 + \left(y_n - y_{goal}\right)^2}. \tag{3.10}$$

Ця евристика є допустимою (адмісивною), тобто вона ніколи не перевищує реальну дальність між точками, що гарантує оптимальність роботи A*.

Алгоритм Дейкстри є фундаментальним методом пошуку найкоротших шляхів:

$$g(n) = min \sum_{e \in path} \omega(e), \tag{3.11}$$

де ваги $\omega(e)$, що відповідають обраній метриці.

3.4 Математична постановка задачі оптимального обходу адрес (TSP)

У процесі доставки меблів важливо не лише знайти оптимальні маршрути між окремими точками транспортної мережі, а й визначити раціональну послідовність відвідування всіх адрес замовлення. Проблема упорядкування кількох точок доставки формулюється як задача комівояжера (Travelling Salesman Problem, TSP), у якій необхідно знайти найкоротший можливий цикл, що проходить через усі задані пункти рівно один раз. У контексті логістичної системи задача дещо модифікується — замість циклу формується маршрут із фіксованою початковою точкою (складом) та множиною адрес доставки, які потрібно відвідати у найвигіднішому порядку. Через високу обчислювальну складність задачі при великій кількості точок використання точних методів оптимізації є недоцільним, тому система застосовує евристичні та метаевристичні підходи, що забезпечують досягнення ефективних розв’язків за прийнятний час.

Математична складова цього етапу починається з формування простору відстаней між усіма парами адрес. Для кожної пари точок i та j визначається евклідова відстань за формулою:

$$g(i,j) = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}, \tag{3.12}$$

що дозволяє апроксимувати реальні міжадресні переміщення до виконання розрахунків на дорожньому графі [10]. На основі цих значень формується квадратна матриця D, у якій кожен елемент містить відстань між відповідними пунктами доставки. Саме ця матриця надалі використовується алгоритмами попередньої оптимізації маршруту, оскільки вона дозволяє порівнювати адреси між собою, визначати близькість точок та оцінювати загальний шлях.

Першим етапом побудови маршруту є застосування алгоритму «найближчого сусіда». Починаючи зі складу або стартової адреси, алгоритм послідовно вибирає з множини ще невідвіданих пунктів той, що має найменшу відстань до поточного місця. Таким чином формується попередній маршрут, який відображає логічну структуру руху, але не гарантує глобальної оптимальності. Алгоритм має лінійну складність щодо кількості точок і дозволяє швидко отримати початкове наближення до оптимального розв’язку задачі комівояжера, що є важливим для систем, у яких швидкість розрахунку відіграє ключову роль [11].

Оскільки маршрут, побудований за методом «найближчого сусіда», може містити неефективні сегменти або «петлі», наступним етапом є його локальна оптимізація за допомогою методу 2-opt. Суть методу полягає в тому, що для двох довільних відрізків маршруту виконується інверсія порядку точок між ними, після чого порівнюється загальна довжина старого і нового варіантів. Якщо новий варіант виявляється коротшим, система приймає його та продовжує пошук інших потенційних покращень. Формально операція описується як перетворення маршруту:

$$route' = route[0:i] + reverse(route[i:j]) + route[j:N], \tag{3.13}$$

яке зменшує загальну довжину, якщо виконується умова $L(route')<L(route)$. Процес повторюється доти, доки не залишається можливості локального покращення, що приводить маршрут до стану локального оптимуму.

Таким чином, комбінація алгоритму найближчого сусіда та локальної оптимізації 2-opt дозволяє системі швидко й ефективно визначати найбільш раціональну черговість відвідування пунктів доставки меблів. Хоча ці методи не гарантують знаходження глобального оптимуму задачі TSP, вони забезпечують високу якість рішень при низьких обчислювальних витратах, що є критично важливим для веб-системи, яка має працювати в режимі реального часу. Крім того, попередня оптимізація послідовності адрес дозволяє значно зменшити навантаження на Python-модуль маршрутизації, оскільки скорочується кількість запитів до алгоритмів пошуку шляху на графі дорожньої мережі. У результаті забезпечується стабільність, передбачуваність і ефективність роботи всієї логістичної платформи.

3.5. Математична модель оцінки ETA (приблизного часу прибуття)

Одним із ключових елементів сучасних логістичних систем є можливість точного прогнозування часу прибуття транспортного засобу до точки доставки, що позначається абревіатурою ETA (Estimated Time of Arrival). Таке прогнозування дозволяє підвищити прозорість роботи служби доставки, зменшити кількість запитань від клієнтів, оптимізувати розподіл навантаження між водіями та забезпечити загальну керованість транспортними потоками. У контексті доставки меблів задача ускладнюється наявністю дорожніх заторів, варіаціями середньої швидкості на різних типах доріг, можливими ремонтами та обмеженнями руху. Тому математична модель ETA повинна бути здатна враховувати не лише геометричну довжину маршруту, а й динамічні характеристики дорожньої мережі [12].

У класичному вигляді час проходження окремого відрізка дороги визначається відповідно до формули:

$$t = \frac{L}{v}, \tag{3.14}$$

де L — довжина ребра дорожнього графа, а v — середня швидкість транспортного засобу на цьому відрізку. Однак у реальному міському середовищі середня швидкість суттєво варіюється залежно від ситуації на дорозі, тому у моделі використовуються не сталі швидкості, а динамічно скориговані значення, що враховують інтенсивність трафіку. Для цього застосовується коефіцієнт трафіку $\omega_{traffic}(e)$, який береться зі

сторонньої системи моніторингу дорожньої ситуації (TomTom Traffic). Цей коефіцієнт відображає співвідношення між швидкістю вільного руху та фактичною швидкістю на конкретному сегменті дороги, додатково враховуючи показники загруженості, так званий jam factor. Таким чином, час проходження одного ребра e дорожнього графа визначається як:

$$t(e) = travel\_time(e) \cdot \omega_{traffic}(e), \tag{3.15}$$

де *travel_time(e)* — базовий час проходження ребра, обчислений ще на етапі збагачення графа, а $\omega_{traffic}(e)$ — коефіцієнт збільшення часу через затори. У випадку сильної загруженості коефіцієнт може збільшуватися в 2–4 рази, що суттєво впливає на загальний прогноз [13]. Для кожного маршруту, побудованого за різними критеріями (найкоротший, найшвидший, найкомфортніший), оцінка ETA виконується окремо, оскільки структура маршруту та характер доріг на ньому різняться.

Загальний ETA для всього маршруту розраховується як сума скоригованого часу для всіх ребер, що входять до оптимізованої послідовності вузлів. Формально це записується як:

$$ETA = \sum_{e \in pat} \omega_{traffic}(e), \tag{3.16}$$

Оскільки базові значення *travel_time* подаються в секундах, підсумований ETA переводиться у хвилини за формулою:

$$ETA_{min} = \frac{ETA}{60}. \tag{3.17}$$

Важливо підкреслити, що модель ETA у системі має декілька рівнів узагальнення. На першому рівні кожному ребру графа приписується базовий час проходження, що враховує довжину дороги, її категорію, середню дозволену швидкість, кількість смуг та якість дорожнього покриття. На другому рівні базовий час коригується глобальним коефіцієнтом трафіку, який отримується з TomTom Traffic API та відповідає середній загруженості району, де проходить маршрут. На третьому рівні виконується агрегування всіх сегментів маршруту та отримання загального прогнозу з максимальною можливою точністю. Комбінація цих рівнів

дозволяє забезпечити баланс між точністю моделі та швидкодією, що є критично важливим у реальних системах маршрутизації.

Запропонована модель ETA не є статичною — вона реагує на динамічні зміни дорожньої ситуації і може відрізнятися залежно від часу доби, погодних умов або дорожніх подій. Це надає системі значних переваг у порівнянні з традиційними підходами, які враховують лише геометричні характеристики маршруту. Використання зовнішніх даних про трафік дозволяє отримати більш реалістичне уявлення про те, скільки реально триватиме доставка, і суттєво підвищує точність планування.

У підсумку математична модель ETA, застосована в системі, забезпечує комплексний підхід до прогнозування часу прибуття, що ґрунтується на поєднанні дорожніх характеристик, динамічної інформації про трафік та оптимізованих маршрутів. Завдяки цьому система може адаптуватися до реальної дорожньої ситуації, надаючи точнішу, узгоджену з практикою логістики оцінку часу доставки, яка є необхідною складовою інтегрованої веб-платформи для оптимізації логістичних процесів меблевої компанії.

3.6 Додаткові математичні моделі для підвищення точності маршрутизації

Оптимізація логістичних маршрутів у реальних міських умовах потребує не лише базових моделей довжини, часу та дорожнього покриття, але й урахування низки додаткових факторів, які здатні суттєво впливати на якість побудованих шляхів. До таких факторів належать поворотні маневри, наявність світлофорів, а також специфічні обмеження, що стосуються руху великогабаритного транспорту. Математичне моделювання цих елементів дозволяє створити вагові функції більш високого рівня точності, що робить можливим формування маршрутів, максимально наближених до реальних умов дорожньої мережі.

Одним із ключових компонентів удосконаленої моделі руху є урахування поворотних маневрів. У міському середовищі саме повороти часто зумовлюють затримки, збільшення часу маневру, накопичення трафіку та додаткове навантаження на транспортний засіб. Математично поворот може бути представлений як

додатковий штраф, який додається до ваги відповідного ребра графа транспортної мережі.

Окрему групу моделей становлять обмеження, пов’язані з експлуатаційними характеристиками вантажного транспорту. Багато міських вулиць мають обмеження щодо максимально дозволеної висоти, маси або довжини транспортного засобу.

Упровадження зазначених математичних моделей дає змогу значно підвищити точність алгоритмів маршрутизації. Поворотні та світлофорні штрафи забезпечують наближення теоретично оптимальних маршрутів до реальних дорожніх умов, враховуючи поведінкові та інфраструктурні особливості міста. Модель перевірки обмежень для вантажівок підвищує надійність та безпечність роботи системи, запобігаючи ситуаціям, коли транспортний засіб опиняється на непридатній для руху ділянці дороги. Сукупність таких моделей формує розширене вагове представлення дорожнього графа, на основі якого алгоритми A*, Dijkstra або модифіковані евристики здатні обчислювати маршрути, що оптимально відповідають логістичним потребам меблевої компанії.

Висновки до розділу

У межах третього розділу було сформовано повне математичне підґрунтя системи оптимізації логістичних маршрутів, що охоплює моделі представлення дорожньої мережі, обчислення відстані, часу та комфортності, а також алгоритмічні засоби пошуку найкращих шляхів у транспортному графі. Виконано аналіз принципів графового моделювання дорожніх мереж із використанням узагальнених структур вузлів і ребер, що визначають просторову та топологічну організацію міста. Подано математичні моделі вагових характеристик ребер, що формують основу розрахунків: довжина, орієнтовна швидкість, час руху, коефіцієнти комфортності та інфраструктурні обмеження. Було розглянуто методи оптимізації порядку відвідування точок доставки, зокрема алгоритм найближчого сусіда та процедуру 2-opt, які дозволяють зменшити сумарний шлях та забезпечити ефективну маршрутизацію. Проаналізовано евристичний апарат пошукових алгоритмів — насамперед A* — що забезпечує швидкий пошук оптимальних шляхів у зважених графах із використанням евклідової евристики. Проведено деталізацію додаткових

моделей, включаючи поворотні штрафи, затримки на світлофорах, врахування трафіку та обмеження вантажності, які підвищують відповідність маршрутів реальним дорожнім умовам. Сукупність розглянутих математичних механізмів формує узгоджений і багатогранний інструментарій, що забезпечує точне, гнучке й адаптивне розв’язання задач маршрутизації в системі доставки меблів.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Архітектура програмної системи та обґрунтування вибору інструментарію

Розроблена веб-система оптимізації логістики доставки меблів має багаторівневу архітектуру клієнт–серверного типу, що включає фронтенд-застосунок, серверну частину на основі платформи Node.js, окремий модуль маршрутизації, реалізований мовою Python, а також систему зберігання даних на базі NoSQL СУБД MongoDB. Така структурна побудова забезпечує розподіл відповідальності між компонентами, підвищує гнучкість, масштабованість і полегшує подальший розвиток системи. На рисунку 4.1 зображено узагальнену схему архітектури системи, на якій відображено взаємодію користувацького інтерфейсу, серверної частини, модулю маршрутизації та бази даних.

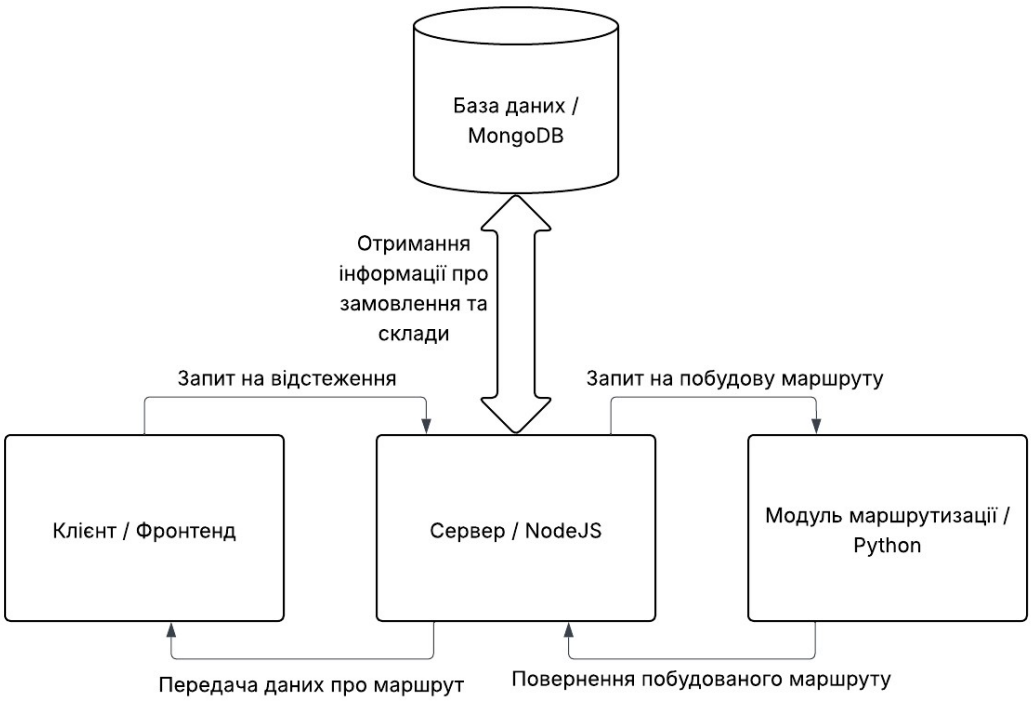

Рисунок 4.1 — Схема архітектури системи

На рівні клієнта (фронтенду) система реалізована як веб-застосунок, який працює у браузері користувача. Інтерфейс дозволяє створювати нові замовлення на доставку меблів, вказуючи персональні дані, місто, одну чи кілька адрес доставки, а також параметри транспорту (наприклад, необхідність використання

великовантажного автомобіля). Окрема сторінка призначена для відстеження стану замовлення та відображення побудованого маршруту на інтерактивній карті. Для реалізації картографічного інтерфейсу використовується JavaScript-бібліотека Leaflet, яка забезпечує відображення підкладки OpenStreetMap, нанесення маркерів, візуалізацію маршрутів у вигляді полілініій і надає зручні інструменти для інтерактивної роботи користувача з картою (масштабування, панорамування, перегляд інформації про точки тощо). Вибір Leaflet обґрунтовується його відкритістю, сумісністю з OSM, простою інтеграцією у веб-інтерфейс та відсутністю ліцензійних обмежень.

Серверна частина системи реалізована на платформі Node.js з використанням фреймворку Express, що забезпечує обробку HTTP-запитів, реалізацію REST API та узгоджену взаємодію між фронтендом, базою даних і модулем маршрутизації. Node.js обрано завдяки його подієво-орієнтованій моделі та неблокуючому введенню/виведенню, що дозволяє ефективно обробляти велику кількість одночасних запитів, характерних для веб-сервісів [14]. Express, у свою чергу, спрощує опис маршрутів, обробку параметрів запитів, підключення middleware-компонентів (наприклад, для валідації, логування, обробки помилок) та структуризацію серверної логіки [15]. На стороні Node.js реалізуються основні REST-ендпоїнти: створення замовлення, отримання інформації про нього, а також виклик функції побудови маршруту для вже збереженого замовлення.

Збереження даних про замовлення та склади здійснюється у MongoDB — документоорієнтованій NoSQL системі керування базами даних. Формат зберігання у вигляді JSON-подібних документів (BSON) добре узгоджується з природою даних веб-застосунку: інформація про замовлення містить поля з даними клієнта, містом, масивом рядкових адрес, булевим параметром використання вантажівки, а також автоматично генерованим ідентифікатором. Окрема колекція Warehouses зберігає інформацію про склади (місто та координати у вигляді широти й довготи). Вибір MongoDB зумовлений її гнучкою схемою (schema-less), можливістю швидкого розширення структури документів без міграцій, а також зручністю роботи з

вкладеними структурами даних, що є характерним для логістичних задач, де адреси, параметри доставок і налаштування можуть змінюватися [16].

Ключовою особливістю архітектури системи є винесення складної логіки побудови маршрутів у окремий Python-модуль маршрутизації, який запускається як веб-сервіс (мікросервіс) на основі фреймворку Flask. Node.js-сервер, отримавши запит на побудову маршруту, звертається до MongoDB для отримання даних про замовлення, визначає відповідний склад за містом замовлення, формує структуру вхідних даних (стартова точка та список адрес), а потім відправляє HTTP-запит до Python-сервера. Така роздільна архітектура дозволяє використовувати сильні сторони Python для наукових та алгоритмічних обчислень, не ускладнюючи код на Node.js. Це також спрощує модульне тестування та дає змогу незалежно масштабувати фронтенд, бекенд і маршрутизатор.

У модулі маршрутизації використовується бібліотека OSMnx для завантаження та обробки дорожнього графа OpenStreetMap, а також бібліотека NetworkX для реалізації алгоритмів графової оптимізації. OSMnx автоматизує побудову графа вулично-дорожньої мережі за назвою населеного пункту (у даному випадку — «Львів») та забезпечує можливість доступу до геометрії ребер, типів доріг, обмежень руху та інших атрибутів. NetworkX надає реалізації алгоритмів A*, Дейкстри та двонаправленого Дейкстри, що використовуються для пошуку найкоротших, найшвидших або найбільш комфортних маршрутів у графі. Обраний стек інструментів Python є природним рішенням для задач маршрутизації, оскільки поєднує розвинуту екосистему бібліотек для роботи з графами, геоданими та математичними обчисленнями [17].

Таким чином, узгоджена комбінація Node.js + Express + MongoDB для реалізації веб-сервісу та Python + Flask + OSMnx + NetworkX для реалізації інтелектуального модуля маршрутизації формує архітектуру, що одночасно є гнучкою, розширюваною та практично орієнтованою. Такий підхід забезпечує чіткий розподіл ролей між компонентами: фронтенд відповідає за взаємодію з користувачем, Node.js — за обробку бізнес-логіки та доступ до даних, Python-модуль — за складні обчислення маршрутів, а MongoDB — за зберігання всіх необхідних сутностей системи. Це дає

змогу ефективно розв’язувати завдання оптимізації логістики меблевої компанії в умовах реальних транспортних обмежень та динамічних дорожніх умов.

4.2 Серверна частина застосунку

Серверна частина розробленої веб-системи виконує ключову роль у забезпеченні стабільної взаємодії між користувачем, базою даних та модулем маршрутизації. Архітектура серверного рівня побудована на основі платформи Node.js із застосуванням фреймворка Express, що створює гнучке, масштабоване та продуктивне середовище для розробки веб-додатків. Основною метою серверної частини є приймання та обробка запитів від клієнтського інтерфейсу, виконання бізнес-логіки, забезпечення доступу до сховища даних, а також комунікація з Python-модулем, який відповідає за побудову оптимального логістичного маршруту. Логіка роботи серверної частини включає кілька взаємопов’язаних етапів, що послідовно виконуються під час створення та відстеження замовлення.

Сервер застосунку приймає запити на створення замовлення, попередньо виконуючи їхню валідацію. На цьому етапі система перевіряє коректність заповнення даних користувачем, зокрема відповідність номера телефону встановленому формату, наявність імені, міста, масиву адрес та інших полів, необхідних для подальшої обробки. Після успішної валідації сервер формує документ замовлення, який передається до бази даних MongoDB [16]. Базовий принцип взаємодії між Express-сервером і СУБД полягає у використанні об’єктно-документного мапера Mongoose, який надає зручні інструменти для опису схем документів та доступу до колекцій. Mongoose дозволяє працювати зі структурою документів у вигляді класів і моделей, полегшуючи виконання CRUD-операцій, автоматичну генерацію ідентифікаторів, роботу з вкладеними структурами, полями масивів та іншими можливостями NoSQL-сховищ.

У системі передбачено дві ключові колекції: замовлення (Orders) та склади (Warehouses). Структура колекції замовлень включає такі поля, як ім’я клієнта, назва компанії, місто, електронна пошта, номер телефону, список адрес доставки, додаткове повідомлення, а також булевий параметр, що вказує на необхідність використання великовантажного автомобіля. Додатково зберігається інформація про

час створення замовлення, який генерується автоматично. Кожне поле має визначений тип відповідно до схеми моделі: рядок, масив рядків, булевий тип або дата. Структура бази даних дозволяє виконувати швидкий пошук замовлень за їх ідентифікатором, фільтрувати по місту, категоризувати за ознакою використання вантажівки та виконувати інші типові запити, що спрощує подальшу інтеграцію з логістичними модулями.

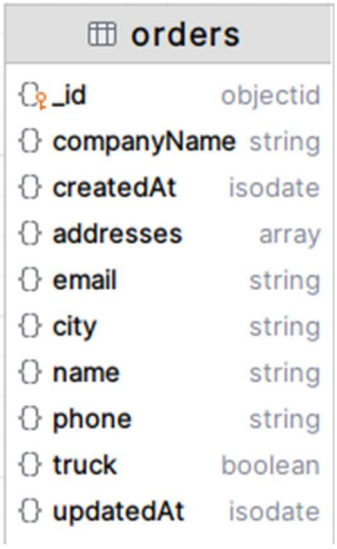

Рисунок 4.2 — Схема документа колекції із замовленнями

Колекція складів містить записи про логістичні хаби компанії, розміщені в різних містах. Кожен документ у колекції складається з назви міста та координат складу у форматі довготи та широти. У випадку надходження запиту на побудову маршруту система знаходить відповідний склад за параметром міста клієнта та передає його координати у Python-модуль маршрутизації. На рисунку 4.3 подано схему документа колекції складів.

Рисунок 4.3 — Схема документа колекції зі складами

Для зручності обробки запитів у сервері застосовується модульна структура, яка передбачає розділення функціональності між різними файлами. У роутерах

Express описуються шляхи HTTP-запитів, наприклад маршрути створення замовлення, отримання деталей замовлення, а також запити для маршрутизації (Рисунок 4.4). Контролери містять бізнес-логіку, що виконується для кожного відповідного запиту — перевірку наявності замовлення, вибір складу, формування запиту до Python-сервера та обробку відповіді (Рисунок 4.5). Така структуризація робить код логічним, чистим і легко підтримуваним, що є важливим у системах, які з часом розширюються.

```
const {Router} = require("express");

const {commonMiddleware} = require("../middlewares/common.middleware");
const {orderMiddleware} = require("../middlewares/order.middleware");
const {OrderValidator} = require("../validators/order.validator");
const {orderController} = require("../controllers/order.controller");

const router = Router();

router.post(
    "/",
    commonMiddleware.isBodyValid(OrderValidator.create),
    orderController.create
)
router.post(
    "/:orderId",
    commonMiddleware.isIdValid("orderId"),
    commonMiddleware.isBodyValid(OrderValidator.getOrder),
    orderMiddleware.isOrderExist("orderId"),
    orderMiddleware.checkPhoneNumber("orderId"),
    orderController.findById
)
router.post(
    "/track/:orderId",
    commonMiddleware.isIdValid("orderId"),
    commonMiddleware.isBodyValid(OrderValidator.getOrder),
    orderMiddleware.isOrderExist("orderId"),
    orderMiddleware.checkPhoneNumber("orderId"),
    orderController.findRoute
)

const orderRouter = router;

module.exports = {orderRouter}
```

Рисунок 4.4 — Файл роутера order.router.js

```
async create(data) {
    try {
        const order = await Order.create(data);

        const context = {
            id: order._id,
            name: data.name,
            companyName: data.companyName,
            phone: data.phone,
            email: data.email,
            city: data.city,
            addresses: data.addresses.toString(),
            message: data.message,
            truck: data.truck ? "Так" : "Ні"
        }

        await emailService.sendMail(data.email, EEmailActions.ORDER_CREATED, context)

        return order
    } catch (e) {
        throw new ApiError(e.message, e.status)
    }
}
```

Рисунок 4.5 — Уривок файлу order.service.js

У межах серверної частини також реалізовано обробку можливих помилок, наприклад, якщо замовлення не знайдено, якщо склад відсутній у базі для відповідного міста, або якщо Python-модуль не відповідає. Для цього використовуються middleware Express, які перехоплюють помилки та формують стандартизовані відповіді. Це підвищує надійність роботи системи та забезпечує коректне інформування користувача про можливі проблеми. Логування запитів реалізується за допомогою інтеграції зі спеціалізованими модулями, що дає можливість відстежувати активність, аналізувати проблемні ситуації та виконувати технічну підтримку. На рисунку 4.6 зображено приклад однієї з middleware.

```
isOrderExist(idField) {
    return async (req, res, next) => {
        try {
            const order = await Order.findById(req.params[idField]);
            if (!order) {
                throw new ApiError("Order not found", 404)
            }

            next()
        } catch (e) {
            next(e)
        }
    }
}
```

Рисунок 4.6 — Уривок файлу order.middleware.js

Таким чином, реалізація серверної частини застосунку поєднує в собі гнучкість, розширюваність та ефективність, що необхідні для сучасної логістичної системи. Використання Node.js та Express дозволяє працювати з високою продуктивністю та обробляти велику кількість одночасних клієнтських запитів, а MongoDB забезпечує зручне та масштабоване зберігання інформації.

4.3 Модуль маршрутизації

Модуль маршрутизації, реалізований у середовищі Python [18], є одним із ключових компонентів системи оптимізації логістики доставки меблів. Його основним призначенням є обробка географічних даних, побудова оптимальних маршрутів із урахуванням дорожньої інфраструктури, транспортних обмежень, особливостей кожного сегмента дороги та актуального стану дорожнього руху.

Модуль взаємодіє з Node.js-сервером через REST API: після отримання даних про склад, адресу клієнта, список зупинок і параметри транспортного засобу він виконує розрахунки та повертає маршрут у вигляді набору координат, підписів точок, загальної довжини, часу та показників комфортності.

Основою маршрутизаційного блоку є використання бібліотеки OSMnx, яка дозволяє отримувати та обробляти дорожні графи OpenStreetMap [19]. Завантаження графа дорожньої мережі певного міста забезпечує доступ до вузлів і ребер, що описують транспортні сегменти, включаючи їхню довжину, покриття, тип дороги, кількість смуг руху, наявні транспортні обмеження та інші характеристики. На етапі попередньої обробки модуль виконує збагачення графа — розрахунок базової швидкості пересування для кожного сегмента дороги, визначення орієнтовного часу проїзду, присвоєння коефіцієнтів комфортності залежно від типу покриття та ширини дороги, а також застосування додаткових параметрів, які допомагають сформувати більш реалістичну та варіативну модель дорожньої мережі [9].

У модулі реалізовано механізм фільтрації графа для вантажного транспорту. Оскільки меблеві компанії можуть використовувати як легкові, так і великовантажні автомобілі, система повинна враховувати обмеження, пов’язані з масою, висотою та довжиною транспортного засобу. У дорожньому графі присутні атрибути OSM, такі як maxweight, maxheight, maxlength, hgv та access, які визначають допустимість проїзду вантажівки конкретним сегментом дороги. Модуль видаляє всі недоступні сегменти, створюючи альтернативний “вантажний граф”, що гарантує побудову безпечного та реалістичного маршруту. На рисунку 4.7 наведено функцію, яка займається фільтрацією графа.

```python
TRUCK = {"weight": 10, "height": 3.5, "length": 8}

def filter_graph_for_truck(G):
    G2 = G.copy()
    remove = []

    for u, v, k, d in G2.edges(keys=True, data=True):

        if d.get("hgv") == "no":
            remove.append((u, v, k))

        if d.get("access") == "no":
            remove.append((u, v, k))

        for key, par in [("maxweight", "weight"), ("maxheight", "height"), ("maxlength", "length")]:
            if key in d:
                try:
                    if float(d[key]) < TRUCK[par]:
                        remove.append((u, v, k))
                except:
                    pass

    G2.remove_edges_from(remove)
    return G2

print("→ Applying truck filter…")
G_truck = filter_graph_for_truck(G_base)
```

Рисунок 4.7 — Функція фільтрування графу для вантажівок

Для уникнення тупикових ситуацій реалізовано розширений механізм пошуку найближчого доступного вузла (Рисунок 4.8). Якщо обрана користувачем адреса знаходиться у внутрішньодворовій зоні, тупиковому провулку або на недоступній для вантажного автомобіля дорозі, модуль намагається знайти найближчу допустиму точку підключення до дорожнього графа, виконуючи пошук у радіусі та перевіряючи наявність сусідніх вузлів. Такий підхід дає змогу уникнути ситуацій, за яких неможливо побудувати маршрут через ізоляцію сегмента.

```python
def find_accessible_node(G, lon, lat):
    try:
        n = ox.distance.nearest_nodes(G, lon, lat)
        if len(list(G.neighbors(n))) > 0:
            return n
    except:
        pass
    return None
```

Рисунок 4.8 — Функція пошуку найближчого доступного вузла

Однією з ключових функцій модуля є підтримка трьох незалежних типів маршрутів: найкоротшого (min distance), найшвидшого (min time) та найбільш комфортного (max comfort). Для кожного типу застосовується власна вагова функція (Рисунок 4.9).

```python
def distance_weight(u, v, d):
    length = d.get("length", 1.0)
    lat, lon = edge_mid(u, v, d)
    tf = traffic_factor(lat, lon)
    curv = d.get("curvature_factor", 1.0)
    return length * (1 + (tf - 1) * 0.3) * (0.5 + 0.5 * curv)


def time_weight(u, v, d):
    base_tt = d.get("travel_time", 1.0)
    sig = d.get("signal_delay", 0.0)
    lat, lon = edge_mid(u, v, d)
    tf = traffic_factor(lat, lon)
    curv = d.get("curvature_factor", 1.0)
    return (base_tt * tf + sig) * (0.7 + 0.3 * curv)


def comfort_weight(u, v, d):
    base_c = d.get("comfort", 1.0)
    lat, lon = edge_mid(u, v, d)
    tf = traffic_factor(lat, lon)
    curv = d.get("curvature_factor", 1.0)
    return base_c * (1 + (tf - 1) * 0.4) * curv
```

Рисунок 4.9 — Вагові функції кожного типу маршрутів

Найкоротший маршрут мінімізує довжину ребер у графі. Найшвидший маршрут враховує не лише базовий travel_time, але й актуальні дані про трафік, отримані з TomTom Traffic API [7]. Для цього модуль здійснює обмежену кількість запитів до API — по одному на сегмент між вузлами TSP-маршруту. Отримані дані містять швидкість у вільному потоці, поточну швидкість, коефіцієнт заторів і jam factor. На основі цих даних формується коригуючий коефіцієнт швидкості, який накладається на travel_time кожного ребра. Маршрут комфортності використовує складнішу метрику, що враховує тип покриття (асфальт, бруківка, ґрунт), кількість смуг, тип дороги, ймовірність заторів, а також можливі локальні особливості. У цього методу метою є мінімізувати “дискомфортність” — умовну величину, пропорційну поганому покриттю та завантаженості дороги.

Ще одним важливим елементом реалізації є алгоритмічний блок, що забезпечує оптимальний порядок відвідування адрес доставки (Рисунок 4.10). Для цього використовується алгоритмічна зв’язка: евклідова матриця відстаней, жадібний алгоритм nearest neighbor та локальна оптимізація методом 2-opt. Такий підхід дозволяє швидко знайти прийнятне рішення задачі комівояжера, не перевищуючи допустимий час обробки, що важливо для інтерактивного використання.

```python
def euc(p1, p2):
    return math.dist(p1, p2)

def dist_matrix(points):
    n = len(points)
    M = [[0]*n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            M[i][j] = euc(points[i]["coords"], points[j]["coords"])
    return M

def nearest_neighbor(M):
    n = len(M)
    un = set(range(1, n))
    route = [0]
    c = 0
    while un:
        nxt = min(un, key=lambda j: M[c][j])
        un.remove(nxt)
        route.append(nxt)
        c = nxt
    return route

def two_opt(route, M):
    def rlen(r):
        return sum(M[r[i]][r[i+1]] for i in range(len(r)-1))

    best = route[:]
    best_len = rlen(best)
    improved = True

    while improved:
        improved = False
        for i in range(1, len(best)-2):
            for j in range(i+1, len(best)-1):
                new = best[:]
                new[i:j+1] = reversed(new[i:j+1])
                new_len = rlen(new)
                if new_len < best_len:
                    best = new
                    best_len = new_len
                    improved = True
    return best
```

Рисунок 4.10 — Функції для отримання оптимального порядку відвідування адрес

Далі для кожної пари послідовних точок у маршруті викликається функція маршрутизації, яка генерує окремий сегмент шляху. Усі сегменти з’єднуються в один цілісний маршрут із усуненням дублювання вершин.

Наприкінці обчислень модуль формує підсумкову відповідь: набір трьох маршрутів (distance, time, comfort), оцінку загального часу прибуття (ETA), а також сумарні відстані та характеристики. Усі результати повертаються у форматі JSON, що дає змогу фронтенду легко побудувати візуальні маршрути на карті (Рисунок 4.11).

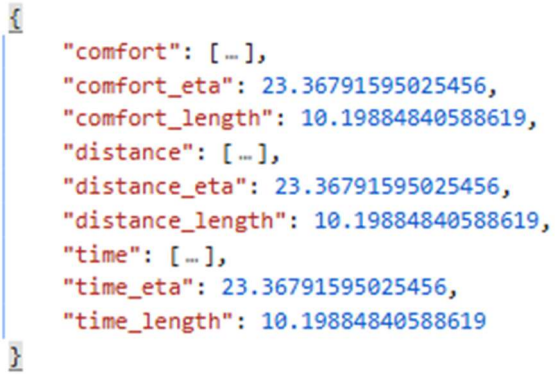

Рисунок 4.11 — Приклад результату у форматі JSON

Завдяки поєднанню алгоритмічної точності, розширеного аналізу дорожніх характеристик, інтеграції з сервісами трафіку та оптимізації обчислювальних витрат модуль маршрутизації забезпечує високу реалістичність і точність побудови логістичних маршрутів.

4.4 Клієнтська частина застосунку

Клієнтська частина веб-системи виконує роль інтерфейсу взаємодії користувача з логістичною платформою, забезпечуючи зручний, інтуїтивний і функціонально повноцінний доступ до ключових можливостей сервісу. Основні операції, які реалізовано у фронтенд-частині, — це створення замовлення, перегляд його даних, відстеження маршруту доставки в режимі реального часу та вибір типу маршруту залежно від потреб користувача. Архітектура клієнтської частини орієнтована на простоту інтеграції, швидку реакцію на події та коректне відображення логістичної інформації, отриманої від серверної частини системи.

Основа інтерфейсу побудована за допомогою HTML5 та CSS3. Для стилізації сторінок застосовано сучасний підхід до компонування блоків, що забезпечує адаптивне відображення незалежно від розміру екрана. Інтерфейс поділено на декілька функціональних зон: головну сторінку сервісу з описом можливостей системи, форму створення замовлення, а також сторінку відстеження доставки, де користувач може переглядати карту маршруту. Важливим елементом є відображення карти з нанесенням точок доставки, складу та маршрутів. Для цієї цілі використовується бібліотека Leaflet — сучасне JavaScript-рішення для роботи з інтерактивними картами, що підтримує велике число плагінів, можливість кастомізації вигляду елементів та плавну роботу навіть на обмежених пристроях [20].

Сторінка створення замовлення містить детальну форму, де користувач вводить дані про себе, компанію, контакти, місто та перелік адрес доставки. Реалізовано динамічне додавання необмеженої кількості адрес, можливість видалення полів, а також інтерактивну валідацію введених даних. Після натискання кнопки “Надіслати запит” дані збираються у структуру JSON і передаються до серверної частини через REST-API. На рисунку 4.12 показано форму для створення замовлення.

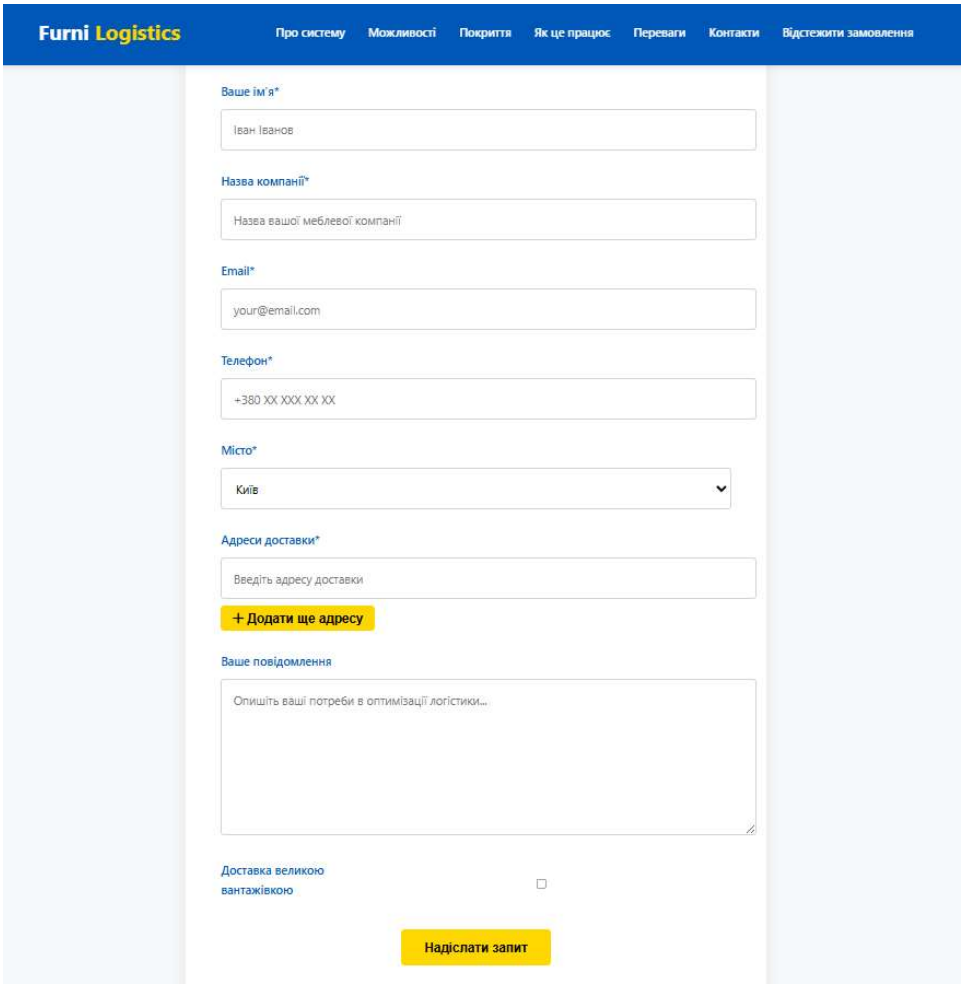

Рисунок 4.12 — Форма для створення замовлення

Найважливішим елементом інтерфейсу є сторінка відстеження маршруту (Рисунок 4.13). При введенні користувачем номера замовлення та номера телефону виконується запит до серверної частини, яка повертає інформацію про клієнта та підготовлені Python-модулем маршрути. Фронтенд опрацьовує відповідь та ініціалізує карту. Усі маршрути відображаються у вигляді прикладених один до одного ліній, які містять набір координат точок переміщення. Реалізовано систему перемикання між трьома типами маршрутів — найкоротшим, найшвидшим та

найбільш комфортним. Користувач має можливість у реальному часі змінювати тип відображуваного маршруту, що особливо корисно для аналізу альтернативних варіантів доставки.

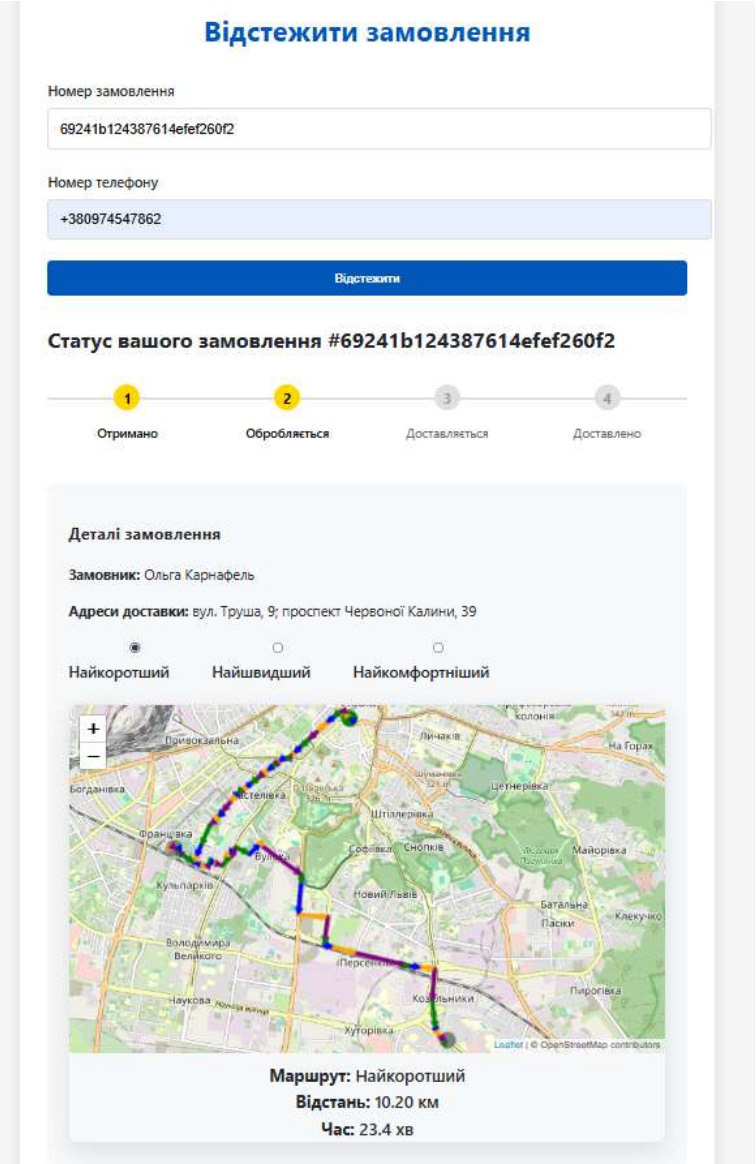

Рисунок 4.14 — Сторінка відстеження маршруту

Одним із додаткових функціональних компонентів сторінки відстеження є відображення метрик маршруту під картою. Зокрема, відображаються орієнтовний час прибуття (ETA), довжина маршруту та інші характеристики, які передає Python-модуль. Завдяки цьому користувач отримує не лише графічну візуалізацію, а й точні числові показники для оцінки складності та тривалості доставки. Ця частина інтерфейсу є важливою для клієнтів, які прагнуть контролювати хід виконання замовлення та планувати свій час.

Усі функції взаємодії з сервером реалізовано за допомогою Fetch API, що дозволяє відправляти HTTP-запити та обробляти асинхронні відповіді без перезавантаження сторінки. Такий підхід забезпечує високу швидкодію, плавність роботи та відсутність затримок у відображенні актуальних даних. Логіка обробки відповіді Python-модуля побудована з урахуванням можливості отримання різних типів маршрутів та обробки випадків помилок, що підвищує стабільність системи.

Загалом клієнтська частина забезпечує зручний доступ до функціоналу системи, відображення складних логістичних даних у наочному вигляді та комфортну взаємодію користувачів з інтелектуальною логістичною платформою. Вона виступає важливою ланкою між технічно складними маршрутизаційними алгоритмами та кінцевим користувачем, забезпечуючи простоту використання та високу інформативність відображення даних.

Висновки до розділу

У цьому розділі було детально розглянуто програмне забезпечення, що лежить в основі розробленої інтелектуальної логістичної системи. Встановлено архітектуру, яка поєднує у собі три ключові компоненти — клієнтську частину, серверну частину на Node.js та модуль маршрутизації на Python. Така багаторівнева структура забезпечує гнучкість, масштабованість і високу точність розрахунків, оскільки кожен компонент виконує чітко визначену роль. Було показано, що серверна частина реалізує поділ відповідальності між обробкою даних, валідацією, взаємодією з базою даних MongoDB та передаванням запитів до обчислювального модуля. Окрему увагу приділено структурі бази даних, яка організована таким чином, щоб забезпечити швидкий доступ до основних сутностей: замовлень та складів.

Особливе значення у розділі відведено Python-модулю маршрутизації, який виконує складні математичні та алгоритмічні розрахунки. У модулі реалізовано побудову трьох типів маршрутів — найкоротшого, найшвидшого та найбільш комфортного — із застосуванням алгоритмів графової теорії та інтеграції з сервісами дорожнього трафіку. Це дозволило створити гнучкий інструмент для вирішення логістичних задач різної складності. Було розглянуто ключові алгоритмічні

компоненти, включаючи обчислення ваг ребер графа, урахування дорожніх обмежень, побудову евристичних маршрутів та оптимізаційні методи.

Клієнтська частина, описана у відповідному підпункті, виступає важливим інтерфейсом для користувача, забезпечуючи інтуїтивну роботу та відображення результатів складних алгоритмів у наочній формі. Реалізований інструментарій взаємодії з сервером, інтерактивні карти, система перемикання між маршрутами та відображення ключових метрик доставки забезпечують повноцінну функціональність системи з точки зору кінцевого користувача. З технічної точки зору клієнтська частина доповнює серверну та обчислювальну підсистеми, забезпечуючи цілісність усієї інфраструктури.

Узагальнюючи, можна стверджувати, що у межах цього розділу сформовано повний опис архітектури, інструментарію та алгоритмічних рішень, які визначають працездатність і ефективність інтелектуальної системи оптимізації логістики. Усі підсистеми гармонійно взаємодіють, забезпечуючи продуктивність, точність маршрутів та зручність використання, що є ключовими вимогами до сучасних логістичних IT-рішень.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Ідея стартап-проєкту ґрунтується на створенні інтелектуальної веб-системи, здатної комплексно оптимізувати логістичні процеси доставки меблевої компанії. На сучасному етапі розвитку галузі дедалі більше підприємств стикаються з труднощами, пов’язаними із значними операційними витратами, великими часовими затримками, низькою передбачуваністю переміщення вантажів та відсутністю автоматизованих інструментів для маршрутизації. Стандартні навігаційні сервіси, попри свою популярність, не враховують специфіку бізнес-процесів, зокрема необхідність оптимізувати доставку відразу до кількох точок, враховувати обмеження дорожньої інфраструктури для вантажних транспортних засобів, а також вибудовувати оптимальний порядок завезення товарів. Сформована ідея проєкту покликана заповнити цю нішу та запропонувати інструмент, який враховує реальні потреби компаній, що здійснюють регулярні перевезення меблів або інших габаритних товарів.

Основна концепція системи полягає у використанні геоінформаційних технологій та алгоритмів оптимізації для автоматичного формування маршруту доставки з урахуванням усіх можливих чинників, що можуть вплинути на ефективність логістичного процесу. На відміну від традиційних рішень, система орієнтується не лише на географічну відстань, а й на дорожні умови, обмеження для вантажного транспорту, реальну завантаженість вулиць та загальні характеристики міського середовища. Це дозволяє формувати три типи маршрутів — найкоротший, найшвидший та “комфортний” — які відрізняються математичними критеріями оптимізації та зорієнтовані на різні стратегічні потреби користувача.

Найкоротший маршрут визначає шлях мінімальної довжини, що є необхідним у ситуаціях, коли компанія прагне мінімізувати пробіг транспортного засобу та витрати на паливо. Найшвидший маршрут розраховується з урахуванням даних сервісу TomTom Traffic, завдяки чому система може реагувати на актуальні дорожні події: затори, уповільнення руху, аварії та інциденти на дорогах. “Комфортний”

маршрут застосовує додаткові ваги, що зменшують пріоритет складних ділянок дорожньої мережі, таких як нерівні дорожні покриття, вузькі вулиці, різкі повороти або ділянки, на яких часто спостерігаються затори. Таким чином, система здатна запропонувати альтернативні шляхи, які покращують якість транспортування та зменшують ризики для вантажу.

Важливим елементом ідеї є інтеграція з внутрішніми даними компанії, зокрема з базою складів, що дозволяє системі визначати оптимальну точку відправлення відповідно до міста клієнта. Це підвищує точність логістичного планування та дає змогу оптимізувати навантаження на складську інфраструктуру. Окрім того, система підтримує автоматичне геокодування адрес, що виключає помилки, пов'язані з некоректним введенням локацій користувачем, та дозволяє з високою точністю відповідати положенню об'єктів на карті.

Окремою перевагою проєкту є можливість детального відстеження замовлення. Користувач отримує доступ до візуалізованої мапи маршруту, інформації про етапи доставки, орієнтовного часу прибуття та статусу перевезення. Це сприяє підвищенню рівня довіри клієнтів та покращує взаємодію між компанією та її споживачами. Для внутрішніх працівників компанії реалізується можливість перегляду структури замовлень, їх статусів, маршрутизації та оптимізованої послідовності доставки. Веб-інтерфейс дозволяє операторам швидко формувати нові завдання, контролювати виконання попередніх та своєчасно реагувати на непередбачувані ситуації.

Ідея проєкту відкриває можливість для використання системи не лише меблевими компаніями, але й іншими суб'єктами господарської діяльності. Зокрема, платформа може застосовуватися службами доставки техніки, будівельними магазинами, логістичними компаніями, підприємствами з дрібними кур'єрськими перевезеннями та будь-якими установами, що регулярно здійснюють переміщення товарів або обладнання. Система має гнучку архітектуру та може адаптуватися до вимог різних бізнес-сфер за рахунок змін алгоритмічних параметрів та можливості інтеграції з внутрішніми корпоративними платформами.

Важливою перевагою проєкту є його здатність працювати виключно на основі відкритих даних та доступних API, що робить систему придатною до масштабування

на інші міста чи країни. Це також значно знижує вартість впровадження та підтримки, оскільки не потребує закупівлі окремих картографічних ліцензій або дорогих сервісів. У разі потреби можливе підключення додаткових модулів, таких як прогнозування часу доставки на основі машинного навчання або автоматизоване планування маршрутів для кількох транспортних засобів.

Особливої уваги заслуговує відмінність проєкту від конкурентних рішень. Поширені сервіси навігації, зокрема Google Maps, Waze чи Apple Maps, не пропонують можливості побудови багатоточкових маршрутів із оптимізацією їх послідовності, а також не орієнтовані на специфіку вантажного транспорту. Платні логістичні платформи достатньо дорогі, орієнтовані на великі компанії й не пропонують індивідуального налаштування під галузеві вимоги меблевого бізнесу. Запропонована система поєднує функціональність, доступність і зосередженість на потребах малого та середнього бізнесу, що забезпечує її конкурентну перевагу на ринку.

Узагальнюючи, ідея стартап-проєкту полягає у створенні комплексної цифрової платформи, яка забезпечує автоматизацію повного циклу логістичної доставки — від формування замовлення до побудови оптимального маршруту та інформування клієнта про його статус. Такий підхід дає можливість зменшити витрати, підвищити ефективність управління логістикою, мінімізувати людський фактор і запропонувати сучасне рішення, здатне адаптуватися до змін ринку. Усе це робить платформу перспективною для комерціалізації та подальшого розвитку як повноцінного стартап-продукту.

5.2 Аналіз технологічних можливостей реалізації ідей проєкту

Технологічні можливості відіграють ключову роль у визначенні життєздатності будь-якого стартап-проєкту, особливо якщо його функціональність пов’язана з обробкою просторових даних, оптимізацією логістичних маршрутів, інтеграцією з зовнішніми сервісами та взаємодією з базами даних. У випадку розробки інтелектуальної системи маршрутизації для меблевої компанії особлива увага приділяється надійності інструментів, їх здатності масштабуватися, сумісності та доступності для розробника. Проведений аналіз свідчить про те, що комплексний

підхід із використанням сучасних веб-технологій, бібліотек для роботи з графами, геоінформаційних сервісів та хмарних рішень на повну забезпечує технологічну здійсненність проєкту та відповідає потребам ідеї.

Основою функціонування системи є серверна частина, побудована на Node.js із застосуванням фреймворку Express. Такий вибір обґрунтовується кількома важливими чинниками. По-перше, Node.js дозволяє створювати високопродуктивні сервери завдяки неблокуючій моделі вводу-виводу, що забезпечує ефективну роботу з великою кількістю одночасних запитів. У контексті логістичної системи, де користувачі можуть одночасно перевіряти статуси замовлень, створювати нові заявки або запитувати маршрути, це має суттєве значення. По-друге, Express надає мінімалістичний, але водночас гнучкий інтерфейс для визначення маршрутизації та логіки обробки запитів, що спрощує розробку та підтримку проєкту. Крім того, екосистема Node.js включає численні модулі, які можуть знадобитися у процесі реалізації, зокрема для валідації даних, авторизації, документації API або оптимізації продуктивності.

Другим важливим компонентом є база даних MongoDB, яка обрана як основний інструмент для зберігання внутрішніх даних системи. Цей рушій баз даних має документно-орієнтовану структуру, що дозволяє зберігати інформацію у форматі JSON-подібних об’єктів. Такий підхід є гнучким та природним для зберігання замовлень, що можуть містити різні поля залежно від типу доставки, додаткових вимог клієнта або обсягу товару. Наприклад, адреси доставки представляють собою масив рядків, дані про склади містять координати, а записи замовлень повинні включати стан, часові мітки, дані клієнта та ознаки доставки вантажівкою. MongoDB легко підтримує змінні структури документів без необхідності змінювати схему, що робить систему адаптивною до нових функціональних вимог у майбутньому. Крім того, масштабованість MongoDB дозволяє збільшувати обсяги даних без суттєвих змін у конфігурації інфраструктури, що є необхідним для стартапу, який може швидко рости.

Окремого аналізу потребує технологія, за допомогою якої реалізовано модуль маршрутизації. Для його побудови застосовано Python разом із бібліотекою OSMnx,

що забезпечує можливість завантаження дорожніх графів на основі даних OpenStreetMap. Це дає змогу отримувати актуальну й детальну карту проїзних шляхів із характеристиками кожного ребра: довжиною, допустимою швидкістю руху, типом дорожнього покриття, кількістю смуг, можливими обмеженнями для вантажного транспорту тощо. Додатково використано бібліотеку NetworkX, яка надає інструменти для побудови графів, виконання пошуку найкоротших шляхів, побудови вагових функцій та роботи з алгоритмами оптимізації. Поєднання цих двох інструментів дозволяє не лише знайти найкоротший маршрут, а й будувати швидкісні та комфортні маршрути завдяки зміні функції ваги. Python у цьому контексті є оптимальним вибором, оскільки має велику кількість наукових бібліотек, простий у використанні та дозволяє створювати окремий мікросервіс, який може працювати незалежно від основної частини застосунку.

Наявність доступних інструментів для отримання даних про дорожню ситуацію також є важливою складовою технологічної спроможності проєкту. Зокрема, для визначення реального стану руху використано API TomTom Traffic, яке надає дані про швидкість руху на відрізках дороги, рівень заторів, індекс завантаженості та інформацію про інциденти. Цей сервіс підтримує безплатний тарифний план із певними обмеженнями, що робить його доступним для стартапів. Оскільки система маршрутизації використовує лише один виклик API на кожну логічну ділянку, це забезпечує економне використання квоти запитів. Частина даних кешується, що додатково зменшує навантаження на сторонній сервіс та підвищує швидкість роботи модуля.

Загалом, технології, необхідні для реалізації стартапу, вже існують, є повністю доступними та не потребують створення нових низькорівневих рішень. Проєкт опирається на відкриті набори даних, безплатні API, бібліотеки з відкритим кодом та інструменти, які широко застосовуються у промислових застосунках. Це свідчить про технічну здійсненність ідеї. Інтеграція між компонентами забезпечується за допомогою HTTP-запитів між Node.js-сервером і Python-модулем, що є стандартним підходом у розподілених системах. Згодом, при зростанні навантаження, модуль

маршрутизації може бути винесений у окремий контейнер або хмарний сервіс, а API-шар — розширений балансувальниками навантаження.

Умови доступності технологій також є сприятливими. Усі інструменти, що використовуються, мають відкритий код або безплатні плани використання. Для розробника не виникає потреби у придбанні дорогого програмного забезпечення або ліцензій. Окрім цього, обрані технології є добре задокументованими і мають значну спільноту, що спрощує процес впровадження та усунення потенційних проблем. Також існує можливість використання численних хмарних платформ для розгортання інфраструктури, що відкриває шлях до масштабування та стабільного функціонування системи.

Отже, аналіз технологічних можливостей реалізації проєкту підтверджує, що всі необхідні інструменти є доступними, сумісними та здатними забезпечити реалізацію передбаченої функціональності системи. Сучасні веб-технології, наявність бібліотек для роботи з дорожніми графами, доступ до актуальних даних про трафік та можливість масштабування на хмарні платформи роблять проєкт технологічно доцільним і придатним до впровадження як стартап-рішення.

5.3 Аналіз ринкових можливостей запуску стартап-проєкту

Ринкові можливості є визначальним чинником для успішного запуску будь-якого стартапу, і системи, що стосуються логістики, як правило, мають особливо високий потенціал через стрімкий розвиток електронної комерції та зростаючу потребу у швидкій доставці товарів. У випадку інтелектуальної системи маршрутизації для меблевих компаній ринок має низку сприятливих характеристик, які формують реальну можливість комерційного впровадження розробленого рішення. Логістичний сектор перебуває у фазі цифрової трансформації, компанії активно шукають способи мінімізувати витрати на пальне, оптимізувати час доставлення, скоротити черги замовлень і підвищити ефективність водіїв. Усе це створює високий попит на технології автоматизованої маршрутизації, особливо на ті, які адаптовані до галузевої специфіки та враховують реальні обмеження, такі як тип транспортного засобу, габарити меблів, складну географію міста або щільність дорожнього руху.

Застарілі методи побудови маршрутів (ручне планування, використання стандартних навігаційних програм без інтеграції в бізнес-процеси, відсутність автоматизованого визначення оптимальної черги адрес) вже не відповідають потребам сучасного ринку. Компанії, що займаються доставленням меблів, стикаються з низкою істотних викликів: необхідністю підбирати маршрут під великогабаритний транспорт, уникати вузьких вулиць та одностороннього руху, враховувати часові піки трафіку, обмеження доступу для вантажівок у центральних частинах міст, а також правильно розподіляти замовлення між кількома складами. Усе це робить класичні навігаційні рішення, такі як Google Maps чи Waze, недостатньо ефективними, оскільки вони не мають механізмів автоматизації логістичних задач, не підтримують оптимізацію множини адрес, а в більшості випадків не враховують індивідуальні фінансові чи технічні параметри роботи компанії. Таким чином, на ринку існує очевидна потреба в спеціалізованому рішенні, що поєднує внутрішню логіку оптимізації із зовнішніми картографічними сервісами та здатне працювати в реальному часі.

Серед ключових ринкових можливостей варто виділити динамічний ріст ринку електронної комерції, збільшення обсягів доставок великогабаритних товарів і активний розвиток малого та середнього бізнесу у сфері виробництва і продажу меблів. Ці тенденції створюють стабільний попит на інструменти, що допомагають оптимізувати логістичні витрати. Меблеві компанії, на відміну від служб доставки дрібних товарів, мають складнішу специфіку: маршрути містять мало точок, але вони значно рознесені географічно; товар потребує акуратного транспортування; транспорт обмежений у маневреності; а логістичні помилки призводять до значних фінансових втрат. На цьому фоні будь-яке рішення, що дозволяє скорочувати пробіг транспорту хоча б на 10–15 %, дає бізнесу відчутну економію, що формує сприятливі умови для впровадження систем автоматичної маршрутизації.

Конкурентне середовище на ринку логістичного програмного забезпечення є доволі розвиненим, однак воно складається переважно з великих універсальних систем, таких як MyRouteOnline, Route4Me, Onfleet або OptimoRoute, які пропонують багатофункціональні рішення, але їх використання є платним, складним для малих

бізнесів або недостатньо спеціалізованим. Усі ці сервіси виконують роль широкого інструменту для різноманітних галузей, але не враховують особливості меблевої логістики: габарити товарів, типи транспортних засобів, необхідність поєднувати роботу зі складами та їхнім розташуванням, а також можливість інтеграції в існуючу CRM або внутрішню систему обліку замовлень. Крім того, їхня вартість може бути високою для локальних компаній. Це відкриває конкурентну нішу для вузькогалузевого рішення, яке буде доступним, адаптивним та здатним працювати на рівні конкретної компанії, а не як універсальна віддалена платформа.

Аналіз структури попиту свідчить про те, що потенційними клієнтами проєкту можуть бути як великі меблеві компанії, так і локальні магазини та склади, а також логістичні служби, що займаються транспортуванням великогабаритних товарів. Додатково до цього, технологія може знайти застосування у суміжних сферах, зокрема у доставці побутової техніки, будівельних матеріалів, металоконструкцій або у службах перевезення корпоративної техніки. Для кожного такого сегмента важливою є економічна ефективність, скорочення часу доставлення та автоматизація розрахунків, що дозволяє мінімізувати людський фактор. Враховуючи можливість гнучкого налаштування параметрів маршрутизації, система може бути швидко адаптована до вимог різних клієнтських груп, що розширює місткість ринку і підвищує шанси на комерційний успіх.

Важливою складовою аналізу ринкових можливостей також є оцінка загроз і ризиків. Однією з основних загроз є залежність від зовнішніх картографічних сервісів, які можуть змінювати умови доступу або обмежувати використання безплатних тарифів. Додатковим ризиком є висока конкуренція в сегменті логістичного програмного забезпечення, де нові рішення з’являються регулярно. Проте вузька спеціалізація проєкту та його прив’язка до конкретної галузі значно знижують загрозу прямої конкуренції. Ще одним ризиком є потенційно низька цифрова грамотність малих підприємств, однак простота інтерфейсу системи, мінімалізм у взаємодії та можливість швидкого навчання користувачів дозволяють нівелювати цей недолік.

5.4 Розроблення ринкової стратегії проєкту

Розроблення ринкової стратегії є ключовим етапом у плануванні комерційного впровадження стартап-проєкту, оскільки саме правильне визначення цільових груп споживачів та обраний формат виходу на ринок впливають на темпи зростання, конкурентоспроможність та фінансові результати. Інтелектуальна система оптимізації маршрутів для меблевих компаній належить до сегменту B2B-рішень, що накладає певні особливості на підхід до маркетингової діяльності. Для таких проєктів критично важливим є вибір не масового ринку, а чітко окреслених сегментів, які мають конкретну, виражену потребу в оптимізації логістичних процесів. У цьому контексті ринкова стратегія має враховувати специфіку меблевої галузі, динаміку її розвитку, готовність компаній до цифрової трансформації та рівень конкуренції серед технологічних рішень, що виконують подібні функції.

На етапі визначення цільових клієнтських сегментів були виокремлені такі основні групи потенційних користувачів: меблеві компанії, що здійснюють доставлення власної продукції; логістичні служби, орієнтовані на великогабаритні перевезення; магазини побутової техніки та будівельних матеріалів; локальні транспортні служби з невеликими автопарками. Серед цих груп найбільш перспективною є категорія меблевих компаній, оскільки саме їхня діяльність найтісніше пов’язана з потребою у точному розрахунку маршрутів, оптимізації навантаження на транспорт та зменшенні витрат. Доставка меблів суттєво відрізняється від стандартної логістики через габарити товару, обмеження щодо типу транспорту, необхідність планування підйому на поверх чи роботи у зоні з ускладненим під’їздом, а також високий рівень відповідальності за збереження продукції. Для таких підприємств запропонована система може стати не просто додатковим інструментом, а ключовим елементом усієї логістичної інфраструктури.

З огляду на визначену структуру споживачів, проєкт обирає стратегію диференційованого маркетингу, оскільки його функціонал може бути адаптований під різні сегменти ринку, а кожен із них має власні специфічні вимоги. Для меблевих компаній акцент робиться на оптимізації маршрутів, зниженні витрат та підвищенні точності часу доставлення. Для логістичних служб акцент буде зміщено на

підвищення пропускної здатності автопарку, зменшення простоїв та автоматизацію обробки замовлень. Для роздрібних магазинів з доставкою великогабаритних товарів ключовою перевагою стане інтеграція системи маршрутизації з внутрішніми CRM або складськими системами, що дозволить автоматично обробляти запити й зменшувати навантаження на персонал. Усі ці напрямки передбачають різну маркетингову комунікацію, що підтверджує доцільність диференційованої стратегії.

Побудова ринкової стратегії включає аналіз конкурентних переваг, які мають бути чітко артикульовані в процесі виходу на ринок. Основною конкурентною перевагою проєкту є спеціалізація на меблевій логістиці, що забезпечує кращу точність маршрутизації порівняно з універсальними сервісами. Друга перевага полягає в адаптації маршруту до параметрів вантажівки, доступності доріг, дорожніх обмежень та чинного трафіку, що реалізується через інтеграцію з TomTom Traffic API та власним алгоритмічним модулем. Третя перевага полягає у використанні інтелектуальних методів побудови оптимального порядку доставки (TSP-оптимізація, 2-opt, модель кількох метрик маршруту), що дозволяє зменшити пробіг та час роботи водіїв. Така спеціалізація формує унікальну ціннісну пропозицію, що дає змогу чітко позиціонувати систему на ринку.

При розробленні ринкової стратегії особлива увага приділяється визначенню каналів просування. Враховуючи специфіку B2B-сектору, ефективними будуть такі канали: пряма комунікація з меблевими фабриками та магазинами, участь у галузевих виставках, публікації у професійних онлайн-спільнотах, таргетована реклама для локального бізнесу, партнерства зі складами та логістичними центрами. Система може бути інтегрована із сервісами, які вже використовують меблеві компанії, такими як CRM, ERP або внутрішні системи управління замовленнями. На рисунку 5.1 (потрібно вставити схему каналів просування) може бути представлена структурна схема маркетингової екосистеми, яка описує взаємозв’язки між компанією, клієнтами, партнерами та каналами просування.

Важливою складовою ринкової стратегії є підхід до формування ціни. Для стартапу доцільно використовувати модель підписки (subscription), що передбачає щомісячну оплату за використання сервісу залежно від обсягу функціональності або

кількості маршрутів. Такий підхід є зручним для клієнтів і забезпечує прогнозовані фінансові надходження для проєкту. Додатково можуть бути запропоновані пакети для малих компаній із невеликою кількістю доставок, а також корпоративні пакети для великих мереж меблевих магазинів. Модель freemium або пробний період у 14 днів допоможуть пришвидшити залучення нових користувачів і створити позитивний досвід взаємодії із системою.

Після визначення цільових груп, каналів просування та ціноутворення формується стратегія виходу на ринок. Вона передбачає три етапи: початковий — запуск MVP та тестування рішення на обмеженій кількості локальних клієнтів; проміжний — масштабування на кілька великих міст України та розширення функціональності; фінальний — вихід на міжнародний ринок, насамперед у країни Східної Європи, де також є попит на оптимізацію доставки великогабаритної продукції. Ця стратегія дає змогу поступово нарощувати функціонал системи, виявляти слабкі місця, коригувати маркетинговий підхід та підвищувати якість сервісу на основі реальних сценаріїв використання.

5.5 Маркетингова програма стартап-проєкту

Маркетингова програма стартап-проєкту визначає комплекс заходів, спрямованих на вихід інтелектуальної системи логістичної оптимізації на ринок, формування попиту, утримання клієнтів та побудову конкурентоспроможного бренду. Її завданням є інтеграція результатів попереднього аналізу — ідентифікованих потреб цільових сегментів, технологічних можливостей реалізації та конкурентного середовища — у цілісну систему ринкових дій. Розроблена програма враховує особливості B2B-сегменту, циклічність меблевого ринку, цифрову трансформацію логістичних процесів та підвищення запиту на оптимізацію витрат у складних економічних умовах.

Одним із ключових елементів маркетингової програми є концепція товару, яка визначає функціональну цінність системи для користувача та формує основу ринкового позиціонування. Продукт позиціонується як сучасна інтелектуальна платформа для оптимізації процесів доставки меблів і великогабаритних товарів. До її складу входить модуль побудови маршрутів з урахуванням дорожніх обмежень,

трафіку, характеристик транспортного засобу та черговості точок доставки; інтегрований механізм розрахунку часу прибуття; система відстеження замовлень; механізм оптимізації черги доставки; а також аналітичний модуль для оцінювання ефективності логістики. У межах концепції продукту визначаються ключові переваги, серед яких: спеціалізація на меблевих доставках, точність маршрутизації, можливість використання декількох метрик побудови маршрутів, інтеграція з Python-модулем оптимізації та зручна клієнтська частина. Усі ці характеристики формують основу унікальної пропозиції, яка відрізняє продукт від конкурентів.

Наступним компонентом є концепція збуту, яка визначає спосіб доведення продукту до кінцевого споживача. Враховуючи B2B-специфіку, основним каналом збуту виступає пряма комунікація з потенційними клієнтами через демонстраційні зустрічі, онлайн-презентації, консультаційні дзвінки та корпоративні продажі. Додатковими каналами стають email-маркетинг, партнерські інтеграції з меблевими виробниками, а також участь у професійних заходах. Продаж здійснюється за моделлю підписки (subscription). Нижчі пакети включають базову маршрутизацію, тоді як корпоративні тарифи розширюються додатковими можливостями, такими як індивідуальна конфігурація, аналітика, пріоритетна підтримка та інтеграція зі складськими або ERP-системами. На рисунку 5.2 (необхідно вставити схему моделі збуту) може бути представлено загальну логіку взаємодії між стартапом, каналами збуту та споживачами.

Концепція просування визначає комплекс заходів, спрямованих на формування попиту, підвищення впізнаваності та стимулювання користувачів до переходу на платну версію. Просування ґрунтується на поєднанні цифрових і традиційних методів комунікації. Основою стають інформаційні матеріали про ефективність оптимізації логістики, демонстраційні відео з побудови реальних маршрутів, кейси використання та відгуки перших клієнтів. Застосовуються корпоративні сторінки у соцмережах, таргетована реклама для підприємств, які займаються доставкою великогабаритних товарів, та прямі звернення до компаній, що вже мають власні доставки. Також доцільним є створення партнерських відносин з платформами, які агрегують логістичні сервіси, та участь у спеціалізованих виставках і форумах. Комплекс

просування представлений у вигляді багаторівневої моделі, що включає створення бренду, інформування, залучення користувачів, утримання та формування довготривалої лояльності.

Важливою частиною маркетингової програми є початковий аналіз можливостей ціноутворення, який ґрунтується на принципі відповідності між користю для клієнта та обсягом ресурсів, що витрачає компанія на роботу сервісу. Враховуючи економічну ситуацію в Україні, оптимальним рішенням є багаторівнева тарифна система. Базовий тариф призначений для малих підприємств з невеликою кількістю доставок і обмеженим автопарком. Середній тариф орієнтований на меблеві компанії середнього масштабу, які потребують регулярної маршрутизації та стабільного функціоналу. Преміальні тарифи орієнтовані на мережеві компанії з можливістю розширення функціоналу під індивідуальні потреби. Таким чином забезпечується охоплення різних сегментів та підвищення гнучкості моделі монетизації. Додатково може бути впроваджена пробна безкоштовна версія на обмежений період, що сприятиме залученню нових клієнтів.

Маркетингова програма також передбачає структуру діяльності з роботи з клієнтами, оскільки B2B-рішення потребують ретельного супроводу. На цьому етапі створюється механізм навчання нових користувачів роботі із системою, збір зворотного зв'язку, покращення функціональності відповідно до потреб клієнтів та оперативна технічна підтримка. Це є важливим фактором підвищення лояльності та зниження ймовірності відмови від сервісу. На рисунку 5.3 (потрібно вставити блок-схему роботи з клієнтами) може бути представлено життєвий цикл взаємодії користувача із системою, від моменту ознайомлення до формування довготривалого партнерства.

Упровадження маркетингової програми спрямоване не лише на отримання перших клієнтів, але й на довготривале масштабування проєкту. Оскільки система заснована на сучасних алгоритмах оптимізації та інтеграції з дорожніми сервісами, її функціональність може бути розширена на інші галузі, зокрема на доставку побутової техніки, матеріалів, інтернет-магазини та сервісні компанії. Це дозволить створити

платформу, яка виходить за межі меблевої логістики і може перетворитися на універсальний сервіс оптимізації маршрутів.

Висновки до розділу

У межах п’ятого розділу було сформовано цілісне бачення стартап-проєкту, що ґрунтується на результатах розробки інтелектуальної веб-системи оптимізації логістики доставки меблевої продукції. Аналіз ідеї проєкту підтвердив наявність значного ринкового запиту на інструменти, здатні зменшити витрати на перевезення, покращити точність планування маршрутів, мінімізувати простої транспорту та підвищити якість взаємодії з клієнтами. Встановлено, що запропонована система має широкий спектр потенційних застосувань у сфері доставки великогабаритних товарів: від меблевих магазинів і виробників до сервісних компаній, перевізників та торгово-логістичних підприємств. Порівняння з конкурентами продемонструвало, що ключовою відмінністю проєкту є спеціалізація саме на особливостях меблевої логістики, інтеграція з алгоритмами оптимізації, забезпечення багатометричної побудови маршрутів (за відстанню, часом, комфортністю) та використання трафікових сервісів для точнішого прогнозування часу доставки. Аналіз технологічних можливостей показав, що проєкт повністю здійсненний технічно завдяки доступності сучасних веб-технологій, JavaScript- та Python-фреймворків, бібліотек для роботи з геоданими, а також відкритих картографічних сервісів. Підтверджено, що всі необхідні технології доступні розробнику, а обрана архітектура є здатною до масштабування та інтеграції з додатковими сервісами у разі зростання кількості користувачів. Ринковий аналіз продемонстрував значні можливості для запуску проєкту, оскільки потреба в цифровізації логістики зростає, а підприємства прагнуть зменшити витрати та підвищити ефективність управління своїми доставками. Визначено цільові групи користувачів, їхні потреби й очікування, а також фактори, що сприяють ринковому впровадженню проєкту, і бар’єри, які необхідно враховувати під час просування. На основі проведеного аналізу було сформовано маркетингову програму, яка включає концепцію продукту, модель збуту, підхід до просування та початкову стратегію ціноутворення. Запропонована програма орієнтована на B2B-ринок, передбачає використання прямого маркетингу,

партнерських інтеграцій, демонстраційних презентацій та гнучкої тарифної моделі. Узагальнюючи, можна стверджувати, що стартап-проєкт має високий потенціал комерціалізації, оскільки відповідає актуальним потребам ринку, демонструє конкурентні переваги, володіє унікальною цінністю для клієнтів та має технологічну базу, достатню для подальшого розвитку і масштабування.

ВИСНОВКИ

У дипломній роботі було виконано комплексне дослідження та реалізовано інтелектуальну веб-систему оптимізації логістики доставки замовлень меблевої компанії. Проведено аналіз предметної області та визначено основні проблеми сучасної меблевої логістики: неефективність ручного планування маршрутів, значні витрати часу й ресурсів, складність урахування дорожніх особливостей українських міст та потреб клієнтів. На основі огляду існуючих рішень сформульовано чітку постановку задачі та обґрунтовано актуальність створення спеціалізованої системи, здатної будувати оптимальні маршрути з врахуванням трафіку, стану доріг та обмежень вантажного транспорту.

Сформовано інформаційну модель, що охоплює замовлення, дані клієнтів, адреси доставки та інформацію про склади. Також розроблено концептуальну модель, яка відображає взаємозв’язки між усіма компонентами системи та забезпечує цілісність і узгодженість процесу обробки даних.

Побудовано графову модель дорожньої мережі міст на базі OpenStreetMap та розроблено набір вагових функцій, що включають довжину, швидкість руху, час проходження, коефіцієнти комфортності та обмеження інфраструктури. Реалізовано алгоритмічний апарат маршрутизації, який поєднує модифікований A*, поворотні штрафи, урахування трафіку TomTom та евристичний пошук оптимального порядку відвідування адрес за алгоритмом «найближчого сусіда» та покращенням методом 2-opt.

Досліджено ринкові можливості впровадження системи, визначено її конкурентні переваги та сформовано ціннісну пропозицію. Система орієнтована на меблеві магазини, логістичні компанії та підприємства, що здійснюють багатоточкові доставки. Аналіз ринку підтвердив високу актуальність цифровізації логістики в Україні, а також попит на рішення, що дозволяють заощаджувати час і ресурси. На основі цього сформовано ринкову стратегію, визначено цільові сегменти та окреслено основні канали просування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Christopher M. Logistics & Supply Chain Management. 6th ed. Harlow: Pearson, 2023. 338 p.;
2. Taillard E. D. Design of Heuristic Algorithms for Hard Optimization: With Python Codes for the Travelling Salesman Problem. Cham: Springer, 2023. 287 p.;
3. Liu Y. Supply Chain Analytics: Concepts, Techniques and Applications. Cham: Springer, 2023. 380 p.;
4. Sierpiński G., Naumann S., Macioszek E. Transport Systems and Urban Logistics. Cham: Springer, 2024. 207 p.;
5. Alharbi I., Ben-Romdhane H. Advances in Computational Logistics and Supply Chain Analytics. Jeddah/Tunis: Springer, 2024. 196 p.;
6. Nominatim OpenStreetMap Search Service. URL: https://nominatim.org;
7. TomTom Traffic Index 2023. URL: https://www.tomtom.com/traffic-index;
8. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2nd ed. Sebastopol: O'Reilly Media, 2018. 316 p.;
9. Boeing G. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. Computers, Environment and Urban Systems. 2017, Vol. 65, pp. 126–139.;
10. Talbi E. Metaheuristics: From Design to Implementation. Hoboken : John Wiley & Sons, 2009. 624 p.;
11. Aggarwal C. Neural Networks and Deep Learning 2nd ed. Springer, 2023. 553 p.;
12. Ghiani G., Laporte G., Musmanno R. Introduction to Logistics Systems Management. 2nd ed. Chichester: Wiley, 2013. 480 p.;
13. Barceló J. Fundamentals of Traffic Simulation. Springer, 2010. 459 p.;
14. Node.js documentation. URL: https://nodejs.org/docs/latest/api/;
15. Express documentation. URL: https://expressjs.com/en/5x/api.html;
16. MongoDB documentation. URL: https://www.mongodb.com/docs/;
17. Cassel L., Gauld A. Python Projects Willey, 2014. 384 p.
18. Python documentation. URL: https://docs.python.org/3/;

19. NetworkX documentation. URL: https://networkx.org/documentation/;
20. Leaflet documentation. URL: https://leafletjs.com/reference.html.

ДОДАТКИ

ДОДАТОК А

package.json

```
{
  "name": "furni",
  "version": "1.0.0",
  "main": "app.js",
  "scripts": {
    "dev": "nodemon ./src/app.js --legacy-watch ./src",
    "start": "node ./src/app.js"
  },
  "author": "Olga Karnafel",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "axios": "^1.9.0",
    "cors": "^2.8.5",
    "dotenv": "^16.5.0",
    "express": "^5.1.0",
    "joi": "^17.13.3",
    "mongoose": "^8.15.1",
    "nodemailer": "^6.10.0",
    "nodemailer-express-handlebars": "^6.1.2"
  },
  "devDependencies": {
    "nodemon": "^3.1.9"
  }
}
```

app.js

```
const express = require("express")
const mongoose = require("mongoose");
const cors = require("cors");
const {orderRouter} = require("./routers/order.router");
const {DB_URL} = require("./configs/config");
const app = express()
app.use(cors())
app.use(express.json());
app.use(express.urlencoded({extended: true}));
app.use("/orders", orderRouter)
app.use((err, req, res, next) => {
    const status = err.status || 500;
    return res.status(status).json({
        message: err.message,
        status: err.status,
    });
});
const dbConnect = async () => {
    let dbCon = false;
    while (!dbCon) {
        console.log("Connecting to database")
        await mongoose
            .connect(DB_URL)
            .then(() => {
                dbCon = true
                console.log("Successfully connected to database")
            })
            .catch(async (err) => {
```

```
            console.log(`Error connecting to database: ${err} \nWait 3 seconds`)
            await new Promise(resolve => setTimeout(resolve, 3000))
        })
    }
}
const PORT = 3000;
app.listen(PORT, async () => {
    await dbConnect()
    console.log(`Server has started on PORT ${PORT}`)
})
```

order.controller.js

```
const {orderService} = require("../services/order.service");

class OrderController {
    async create(req, res, next) {
        try {
            const createdOrder = await orderService.create(req.body);

            return res.json(createdOrder)
        } catch (e) {
            next(e)
        }
    }

    async findById(req, res, next) {
        try {
            const {orderId} = req.params;
            const order = await orderService.findById(orderId);

            return res.json(order)
        } catch (e) {
            next(e)
        }
    }

    async findRoute(req, res, next) {
        try {
            const {orderId} = req.params;
            const order = await orderService.findRoute(orderId);

            return res.json(order)
        } catch (e) {
            next(e)
        }
    }
}

const orderController = new OrderController();

module.exports = {orderController}
```

order.service.js

```
const axios = require("axios");
const {Order} = require("../models/Order.model");
const {ApiError} = require("../errors/api.error");
const {Warehouse} = require("../models/Warehouse.model");
const {emailService} = require("./email.service");
const {EEmailActions} = require("../enums/email.enum");
class OrderService {
    async create(data) {
        try {
            const order = await Order.create(data);
            const context = {
```

```
            id: order._id,
            name: data.name,
            companyName: data.companyName,
            phone: data.phone,
            email: data.email,
            city: data.city,
            addresses: data.addresses.toString(),
            message: data.message,
            truck: data.truck ? "Так" : "Ні"
        }
        await emailService.sendMail(data.email, EEmailActions.ORDER_CREATED, context)
        return order
    } catch (e) {
        throw new ApiError(e.message, e.status)
    }
}
async findById(id) {
    try {
        return await Order.findById(id)
    } catch (e) {
        throw new ApiError(e.message, e.status)
    }
}
async findRoute(id) {
    try {
        let {addresses, city, truck} = await Order.findById(id);
        const vehicleStart = await Warehouse.findOne({city: city});
        const res = await axios.post(
            "http://127.0.0.1:5000/route",
            {
                start: {
                    label: "start",
                    coords: vehicleStart.coords,
                    city
                },
                addresses,
                truck
            }
        );
        return res.data
    } catch (e) {
        throw new ApiError(e.message, e.status)
    }
}
}
const orderService = new OrderService();
module.exports = {orderService}
```

order.router.js

```
const {Router} = require("express");

const {commonMiddleware} = require("../middlewares/common.middleware");
const {orderMiddleware} = require("../middlewares/order.middleware");
const {OrderValidator} = require("../validators/order.validator");
const {orderController} = require("../controllers/order.controller");

const router = Router();

router.post(
    "/",
    commonMiddleware.isBodyValid(OrderValidator.create),
    orderController.create
)
```

```
router.post(
    "/:orderId",
    commonMiddleware.isIdValid("orderId"),
    commonMiddleware.isBodyValid(OrderValidator.getOrder),
    orderMiddleware.isOrderExist("orderId"),
    orderMiddleware.checkPhoneNumber("orderId"),
    orderController.findById
)
router.post(
    "/track/:orderId",
    commonMiddleware.isIdValid("orderId"),
    commonMiddleware.isBodyValid(OrderValidator.getOrder),
    orderMiddleware.isOrderExist("orderId"),
    orderMiddleware.checkPhoneNumber("orderId"),
    orderController.findRoute
)

const orderRouter = router;

module.exports = {orderRouter}
```

order.middleware.js

```
const {ApiError} = require("../errors/api.error");
const {Order} = require("../models/Order.model");

class OrderMiddleware {
    isOrderExist(idField) {
        return async (req, res, next) => {
            try {
                const order = await Order.findById(req.params[idField]);
                if (!order) {
                    throw new ApiError("Order not found", 404)
                }

                next()
            } catch (e) {
                next(e)
            }
        }
    }

    checkPhoneNumber (idField) {
        return async (req, res, next) => {
            try {
                const order = await Order.findById(req.params[idField]);

                const {phone} = req.body;

                if (phone !== order.phone) {
                    throw new ApiError("Unknown phone number", 401)
                }

                next()
            } catch (e) {
                next(e)
            }
        }
    }
}

const orderMiddleware = new OrderMiddleware();

module.exports = {orderMiddleware}
```

ДОДАТОК Б

main.py

```
from flask import Flask, request, jsonify
from route_utils_v6 import compute_full_route
app = Flask(__name__)
@app.route("/", methods=["GET"])
def api():
    return jsonify("Hello World!!!")
@app.route("/route", methods=["POST"])
def route():
    data = request.get_json()
    full_route = compute_full_route(
        data["start"],
        data["addresses"],
        data.get("truck", False)
    )
    print(full_route)
    return jsonify(full_route)
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

route_utils_v6.py

```
import math
import requests
import osmnx as ox
import networkx as nx
from functools import lru_cache
from networkx.algorithms.simple_paths import shortest_simple_paths
TOMTOM_KEY = "key"
NOMINATIM_URL = "https://nominatim.openstreetmap.org/search"
CELL_SIZE = 0.01
TRAFFIC_SIGNAL_DELAY_SEC = 20.0
print("→ Loading Lviv graph…")
G_base = ox.graph_from_place("Львів", network_type="drive")
def enrich_graph(G):
    avg_speeds = {
        "motorway": 90,
        "trunk": 80,
        "primary": 70,
        "secondary": 60,
        "tertiary": 50,
        "residential": 30,
        "service": 20,
    }
    for u, v, k, d in G.edges(keys=True, data=True):
        length = d.get("length", 1.0)
        hw = d.get("highway", "")
        if isinstance(hw, list):
            hw = hw[0]
        if "maxspeed" in d:
            try:
                speed = float(str(d["maxspeed"]).split()[0])
            except:
                speed = 40
        else:
            speed = avg_speeds.get(hw, 40)
        d["speed_kmh"] = speed
        d["travel_time"] = length / (speed * 1000 / 3600)
        surface = d.get("surface", "asphalt")
        try:
```

```
            lanes = int(d.get("lanes", 2))
        except:
            lanes = 2
        comfort_penalty = 1.0
        bad_surfaces = [
            "gravel", "compacted", "ground", "unpaved",
            "dirt", "sand", "fine_gravel",
            "cobblestone", "sett"
        ]
        if surface in bad_surfaces:
            comfort_penalty *= 10.0
        if lanes == 1:
            comfort_penalty *= 5.0
        if hw in ["service", "residential"]:
            comfort_penalty *= 3.0
        curvature = 1.0
        geom = d.get("geometry")
        if geom is not None:
            geom_len = geom.length
            x1, y1 = geom.coords[0]
            x2, y2 = geom.coords[-1]
            straight = math.dist((x1, y1), (x2, y2))
            if straight > 0:
                k = geom_len / straight
                if k > 1.1:
                    curvature = k
        comfort_penalty *= curvature
        d["comfort"] = length * comfort_penalty
        d["curvature_factor"] = curvature
        d["signal_delay"] = 0.0
    signal_nodes = [
        n for n, nd in G.nodes(data=True)
        if nd.get("highway") == "traffic_signals"
    ]
    for n in signal_nodes:
        for u, v, k, d in G.in_edges(n, keys=True, data=True):
            d["signal_delay"] += TRAFFIC_SIGNAL_DELAY_SEC
        for u, v, k, d in G.out_edges(n, keys=True, data=True):
            d["signal_delay"] += TRAFFIC_SIGNAL_DELAY_SEC
    return G
print("→ Enriching graph…")
G_base = enrich_graph(G_base)
TRUCK = {"weight": 10, "height": 3.5, "length": 8}
def filter_graph_for_truck(G):
    G2 = G.copy()
    remove = []
    for u, v, k, d in G2.edges(keys=True, data=True):
        if d.get("hgv") == "no":
            remove.append((u, v, k))
        if d.get("access") == "no":
            remove.append((u, v, k))
        for key, par in [("maxweight", "weight"), ("maxheight", "height"), ("maxlength", "length")]:
            if key in d:
                try:
                    if float(d[key]) < TRUCK[par]:
                        remove.append((u, v, k))
                except:
                    pass
    G2.remove_edges_from(remove)
    return G2
print("→ Applying truck filter…")
G_truck = filter_graph_for_truck(G_base)
def find_accessible_node(G, lon, lat):
```

```
    try:
        n = ox.distance.nearest_nodes(G, lon, lat)
        if len(list(G.neighbors(n))) > 0:
            return n
    except:
        pass
    return None
def cell_id(lat, lon):
    return int(lat / CELL_SIZE), int(lon / CELL_SIZE)
def edge_mid(u, v, d):
    if "geometry" in d and d["geometry"] is not None:
        g = d["geometry"]
        (x1, y1) = g.coords[0]
        (x2, y2) = g.coords[-1]
    else:
        x1, y1 = G_base.nodes[u]["x"], G_base.nodes[u]["y"]
        x2, y2 = G_base.nodes[v]["x"], G_base.nodes[v]["y"]
    return (y1 + y2) / 2, (x1 + x2) / 2  # (lat, lon)
@lru_cache(maxsize=5000)
def _cell_flow(i, j):
    if not TOMTOM_KEY:
        return 1.0
    lat = (i + 0.5) * CELL_SIZE
    lon = (j + 0.5) * CELL_SIZE
    url = "https://api.tomtom.com/traffic/services/4/flowSegmentData/relative0/10/json"
    params = {"key": TOMTOM_KEY, "point": f"{lat},{lon}"}
    try:
        r = requests.get(url, params=params, timeout=2)
        d = r.json().get("flowSegmentData", {})
        cur = d.get("currentSpeed", 40)
        free = d.get("freeFlowSpeed", 40)
        jam = d.get("jamFactor", 0)
        if cur <= 1:
            return 3.0
        return min(max((free / cur) * (1 + jam / 10), 1.0), 4.0)
    except:
        return 1.0
@lru_cache(maxsize=5000)
def _cell_incidents(i, j):
    if not TOMTOM_KEY:
        return 1.0
    lat = (i + 0.5) * CELL_SIZE
    lon = (j + 0.5) * CELL_SIZE
    half = CELL_SIZE / 2
    min_lat = lat - half
    max_lat = lat + half
    min_lon = lon - half
    max_lon = lon + half
    bbox = f"{min_lon},{min_lat},{max_lon},{max_lat}"
    url = "https://api.tomtom.com/traffic/services/5/incidentDetails"
    params = {"key": TOMTOM_KEY, "bbox": bbox, "fields": "severity"}
    try:
        r = requests.get(url, params=params, timeout=2)
        inc = r.json().get("incidents", [])
        if not inc:
            return 1.0
        count = len(inc)
        max_sev = max((i.get("severity", 1) for i in inc), default=1)
        return 1.0 + min(0.1 * count + 0.3 * max_sev, 2.0)
    except:
        return 1.0
def traffic_factor(lat, lon):
    i, j = cell_id(lat, lon)
```

```
    t = _cell_flow(i, j)
    p = _cell_incidents(i, j)
    return min(max(t * p, 1.0), 6.0)
def distance_weight(u, v, d):
    length = d.get("length", 1.0)
    lat, lon = edge_mid(u, v, d)
    tf = traffic_factor(lat, lon)
    curv = d.get("curvature_factor", 1.0)
    return length * (1 + (tf - 1) * 0.3) * (0.5 + 0.5 * curv)
def time_weight(u, v, d):
    base_tt = d.get("travel_time", 1.0)
    sig = d.get("signal_delay", 0.0)
    lat, lon = edge_mid(u, v, d)
    tf = traffic_factor(lat, lon)
    curv = d.get("curvature_factor", 1.0)
    return (base_tt * tf + sig) * (0.7 + 0.3 * curv)
def comfort_weight(u, v, d):
    base_c = d.get("comfort", 1.0)
    lat, lon = edge_mid(u, v, d)
    tf = traffic_factor(lat, lon)
    curv = d.get("curvature_factor", 1.0)
    return base_c * (1 + (tf - 1) * 0.4) * curv
def build_routes_between(lon1, lat1, lon2, lat2, truckMode, start_label=None, end_label=None):
    Gp = G_truck if truckMode else G_base
    s = find_accessible_node(Gp, lon1, lat1)
    e = find_accessible_node(Gp, lon2, lat2)
    if s is None or e is None:
        raise ValueError("Не знайдено доступних вузлів")
    K = 5
    candidates = []
    try:
        gen = shortest_simple_paths(Gp, s, e, weight=time_weight)
        for i, path in enumerate(gen):
            if i >= K:
                break
            candidates.append(path)
    except:
        path = nx.shortest_path(Gp, s, e, weight="length")
        candidates.append(path)
    def score(path):
        total_len = 0.0
        total_time = 0.0
        total_comf = 0.0
        for u, v in zip(path[:-1], path[1:]):
            ed = G_base.get_edge_data(u, v)
            if not ed:
                ed = G_base.get_edge_data(v, u)
                if not ed:
                    continue
            d = list(ed.values())[0]
            L = d.get("length", 1.0)
            tt = d.get("travel_time", 1.0)
            cf = d.get("comfort", 1.0)
            lat, lon = edge_mid(u, v, d)
            tf = traffic_factor(lat, lon)
            total_len += L
            total_time += tt * tf
            total_comf += cf * (1 + (tf - 1) * 0.4)
        return {"length": total_len, "time": total_time, "comfort": total_comf}
    metrics = [score(p) for p in candidates]
    def argmin(key):
        return min(range(len(metrics)), key=lambda i: metrics[i][key])
```

```
    idx_time = argmin("time")
    idx_dist = argmin("length")
    idx_comf = argmin("comfort")
    # ensure different routes where possible
    used = {idx_time}
    if idx_dist in used:
        for i in sorted(range(len(metrics)), key=lambda x: metrics[x]["length"]):
            if i not in used:
                idx_dist = i
                break
    used.add(idx_dist)
    if idx_comf in used:
        for i in sorted(range(len(metrics)), key=lambda x: metrics[x]["comfort"]):
            if i not in used:
                idx_comf = i
                break
    path_time = candidates[idx_time]
    path_dist = candidates[idx_dist]
    path_comf = candidates[idx_comf]
    def convert(path):
        arr = []
        for i, n in enumerate(path):
            item = {"coords": [G_base.nodes[n]["x"], G_base.nodes[n]["y"]]}
            if i == 0 and start_label:
                item["label"] = start_label
            if i == len(path) - 1 and end_label:
                item["label"] = end_label
            arr.append(item)
        return arr
    return {
        "time": convert(path_time),
        "distance": convert(path_dist),
        "comfort": convert(path_comf),
    }
@lru_cache(maxsize=2000)
def geocode_address(addr, city):
    r = requests.get(
        NOMINATIM_URL,
        params={"q": f"{addr}, {city}", "format": "json", "limit": 1},
        headers={"User-Agent": "logistics"},
        timeout=5
    )
    d = r.json()
    if not d:
        raise ValueError(f"Не знайдено адресу: {addr}")
    d = d[0]
    return [float(d["lon"]), float(d["lat"])]
def euc(p1, p2):
    return math.dist(p1, p2)
def dist_matrix(points):
    n = len(points)
    M = [[0]*n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            M[i][j] = euc(points[i]["coords"], points[j]["coords"])
    return M
def nearest_neighbor(M):
    n = len(M)
    un = set(range(1, n))
    route = [0]
    c = 0
    while un:
        nxt = min(un, key=lambda j: M[c][j])
```

```
        un.remove(nxt)
        route.append(nxt)
        c = nxt
    return route
def two_opt(route, M):
    def rlen(r):
        return sum(M[r[i]][r[i+1]] for i in range(len(r)-1))
    best = route[:]
    best_len = rlen(best)
    improved = True
    while improved:
        improved = False
        for i in range(1, len(best)-2):
            for j in range(i+1, len(best)-1):
                new = best[:]
                new[i:j+1] = reversed(new[i:j+1])
                new_len = rlen(new)
                if new_len < best_len:
                    best = new
                    best_len = new_len
                    improved = True
    return best
def compute_eta(route):
    if not route or len(route) < 2:
        return 0.0
    total = 0.0
    for i in range(len(route)-1):
        lon1, lat1 = route[i]["coords"]
        lon2, lat2 = route[i+1]["coords"]
        try:
            u = ox.distance.nearest_nodes(G_base, lon1, lat1)
            v = ox.distance.nearest_nodes(G_base, lon2, lat2)
        except:
            continue
        ed = G_base.get_edge_data(u, v)
        if not ed:
            ed = G_base.get_edge_data(v, u)
            if not ed:
                continue
        d = list(ed.values())[0]
        base_tt = d.get("travel_time", 1.0)
        sig = d.get("signal_delay", 0.0)
        lat_mid, lon_mid = edge_mid(u, v, d)
        tf = traffic_factor(lat_mid, lon_mid)
        curv = d.get("curvature_factor", 1.0)
        total += (base_tt * tf + sig) * (0.7 + 0.3 * curv)
    return total / 60.0  # minutes
def compute_distance(route):
    if not route or len(route) < 2:
        return 0.0
    total_m = 0.0
    for i in range(len(route) - 1):
        lon1, lat1 = route[i]["coords"]
        lon2, lat2 = route[i+1]["coords"]
        try:
            u = ox.distance.nearest_nodes(G_base, lon1, lat1)
            v = ox.distance.nearest_nodes(G_base, lon2, lat2)
        except Exception:
            continue
        edges = G_base.get_edge_data(u, v)
        if not edges:
            edges = G_base.get_edge_data(v, u)
            if not edges:
```

```
                continue
            data = list(edges.values())[0]
            L = data.get("length", 0.0)
            total_m += L
        return total_m / 1000.0  # в кілометри
    def compute_full_route(start, addresses, truckMode):
        jobs = []
        for i, a in enumerate(addresses):
            coords = geocode_address(a, start["city"])
            jobs.append({
                "label": f"{a}; Job{i+1}",
                "coords": coords
            })
        points = [start] + jobs
        M = dist_matrix(points)
        nn = nearest_neighbor(M)
        order = two_opt(nn, M)
        ordered = [points[i] for i in order]
        dist_route = []
        time_route = []
        comf_route = []
        for i in range(len(ordered)-1):
            p1 = ordered[i]
            p2 = ordered[i+1]
            seg = build_routes_between(
                p1["coords"][0], p1["coords"][1],
                p2["coords"][0], p2["coords"][1],
                truckMode,
                p1["label"], p2["label"]
            )
            for key, dest in [
                ("distance", dist_route),
                ("time", time_route),
                ("comfort", comf_route),
            ]:
                if seg[key]:
                    if dest:
                        dest.extend(seg[key][1:])
                    else:
                        dest.extend(seg[key])
        return {
            "distance": dist_route,
            "time": time_route,
            "comfort": comf_route,
            "distance_eta": compute_eta(dist_route),
            "time_eta": compute_eta(time_route),
            "comfort_eta": compute_eta(comf_route),
            "distance_length": compute_distance(dist_route),
            "time_length": compute_distance(time_route),
            "comfort_length": compute_distance(comf_route),
        }
```