НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Інформаційна система формування маршрутів подорожей»

Виконав:
студент IV курсу, групи ІС-12
Манов Дмитро Володимирович _____

Керівник:
Доцент, к.т.н., доцент каф. ІСТ
Деведжіогуллари Алла Вікторівна _____

Рецензент:
Доцент кафедри ОТ, к.т.н., доцент
Роковий Олександр Петрович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту
Манову Дмитру Володимировичу

1. Тема проєкту «Інформаційна система формування маршрутів подорожей», керівник проєкту Деведжіогуллари Алла Вікторівна, к.т.н., доцент, затверджені наказом по університету від «23» травня 2025 р. № 1705-с.

2. Термін подання студентом проєкту: 9 червня 2025 р.

3. Вихідні дані до проєкту: мова програмування Python, редактор коду Visual Studio Code, фреймворк Flask

4. Зміст пояснювальної записки:

1. Опис предметної області
2. Аналіз існуючих рішень
3. Формування вимог до системи
4. Вибір технологій розробки
5. Розробка інформаційної системи
6. Математичне забезпечення
7. Тестування системи

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо):

1. Діаграма компонентів
2. Діаграма послідовностей
3. Діаграма потоків даних
4. Структурна схема

6. Дата видачі завдання: 6 березня 2025 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Аналіз предметної області	14 квітня 2025	
2	Огляд існуючих аналогів	17 квітня 2025	
3	Формування цілей та задач розробки	21 квітня 2025	
4	Розробка програмного рішення для збору даних	29 квітня 2025	
5	Розробка алгоритму	6 травня 2025	
6	Розробка веб-інтерфейсу	11 травня 2025	
7	Тестування готового проєкту	17 травня 2025	
8	Оформлення інструкцій користувача	24 травня 2025	
9	Подання ДП на основний захист	9 червня 2025	

Студент Дмитро МАНОВ

Керівник Алла ДЕВЕДЖІОГУЛЛАРИ

АНОТАЦІЯ

Манов Д.В. Інформаційна система формування маршрутів подорожей. – Київ: КПІ ім. Ігоря Сікорського, 2025. Проєкт містить 68 с. тексту, 12 рисунків, 8 таблиць, посилання на 24 літературні джерела, 5 додатків.

A*, DIJKSTRA, OPENROUTESERVICE, ВИТРАТА ПАЛЬНОГО, ІНФОРМАЦІЙНА СИСТЕМА, КУЛЬТУРНІ ТОЧКИ, МАРШРУТИЗАЦІЯ, МУРАШИНИЙ АЛГОРИТМ, ОПТИМІЗАЦІЯ МАРШРУТУ.

Об’єктом дослідження є процес автоматизованого планування маршрутів подорожей.

Метою дипломного проєкту є формування оптимальних маршрутів для подорожей.

У межах проєкту реалізовано три режими маршрутизації: короткий (за мінімальною довжиною маршруту), економний (за мінімальною витратою пального), культурний (із максимальною кількістю пам’яток уздовж маршруту). Система побудована на базі мови програмування Python з використанням Flask, інтеграції з OpenRouteService API, fueleconomy.gov API, а також бібліотек osmnx, networkx та інших.

Маршрутизація реалізована на основі класичних алгоритмів A* і Dijkstra для оптимізації за витратою пального та довжиною шляху, а також алгоритму колонії мурах для знаходження шляху за критерієм кількості культурних точок (POI).

Інформаційна система має зручний вебінтерфейс, дозволяє користувачеві задавати вхідні параметри, переглядати карту маршруту, порівнювати результати за метриками.

Результати розробки можуть бути використані в туристичних сервісах, логістичних платформах або в освітніх цілях для вивчення задач багатокритеріальної маршрутизації.

SUMMARY

Manov D.V. Information System for Travel Route Planning. – Kyiv: Igor Sikorsky Kyiv Polytechnic Institute, 2025. – 68 p., 12 figures, 8 tables, 24 sources, 5 appendices.

A*, ANT COLONY OPTIMIZATION, CULTURAL POINTS OF INTEREST, DIJKSTRA, FUEL CONSUMPTION, INFORMATION SYSTEM, OPENROUTESERVICE, ROUTE OPTIMIZATION, ROUTING.

The object of this study is the process of automated travel route planning.

The goal of the bachelor's project is to develop develop optimal travel routes.

The system supports three routing modes: shortest, economic and cultural (maximizing POIs). It is implemented in Python using Flask, OpenRouteService API, fueleconomy.gov API, and supporting libraries like osmnx, networkx.

Routing is based on A* and Dijkstra's algorithms for distance/fuel metrics and ant colony optimization for POI-based cultural routing.

The information system provides a user-friendly web interface that allows users to enter input parameters, view the route on an interactive map, and compare results by selected metrics.

The project's results can be applied in tourism, logistics, or educational fields.

№ рядка	Формат	Позначення	Найменування	Кіл. аркушів	№ екз.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІС12.210БАК.005 ПЗ	Інформаційна система формування	68		
6			маршрутів подорожей.			
7			Пояснювальна записка			
8						
9	А3	ІС12.210БАК.005 Д1	Інформаційна система формування	1		
10			маршрутів подорожей.			
11			Структурна схема			
12						
13	А3	ІС12.210БАК.005 Д2	Інформаційна система формування	1		
14			маршрутів подорожей.			
15			Діаграма компонентів			
16						
17	А3	ІС12.210БАК.005 Д3	Інформаційна система формування	1		
18			маршрутів подорожей.			
19			Діаграма потоків даних			
20						
21	А3	ІС12.210БАК.005 Д4	Інформаційна система формування	1		
22			маршрутів подорожей.			
23			Діаграма послідовностей			
24						
25						
26						
27						
28						

					ІС12.210БАК.005 ТП		
Зм.	Лист	№ докум.	Підпис				
Розробив		Манов			Інформаційна система формування маршрутів подорожей. Відомість дипломного проєкту	Літ.	Арк. 1 / Аркушів 1
Перевірив		Деведжіогуллари				Т	
Затв.					КПІ ім. Ігоря Сікорського Група ІС-12		

Пояснювальна записка
до дипломного проєкту
на тему: «Інформаційна система формування маршрутів подорожей»

Київ – 2025 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 3
ВСТУП 5
1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ 7
1.1 Опис процесу діяльності 7
1.2 Постановка задачі 8
1.2.1 Призначення системи 8
1.2.2 Цілі та задачі розробки 9
2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ 11
3 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ 15
3.1 Вимоги до системи в цілому 15
3.1.1 Функціональні вимоги 15
3.1.2 Нефункціональні вимоги 16
3.1.3 Вимоги до тестування 17
3.1.4 Вимоги до технічного забезпечення 17
4 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ 19
4.1 Вибір та обґрунтування технологій розробки 19
4.2 Мова програмування та середовище розробки 19
4.3 Вебфреймворк та архітектура 20
4.4 Бібліотеки та пакети 22
4.5 API та зовнішні сервіси 23
4.6 Інтерфейс користувача 25
5 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ 28
5.1 Структура системи 28
5.2 Функціональна модель системи 29
5.2.1 Побудова графа дорожньої мережі 29

5.2.2 Інтеграція даних POI 31
5.2.3 Обчислення витрати пального 32
5.2.4 Реалізація алгоритмів маршрутизації 34
5.3 Вивід результатів та інтерфейс 38
6 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ 41
6.1 Змістовна постановка задачі 41
6.2 Математична постановка задачі 43
6.3 Обґрунтування методу розв’язання 45
6.4 Опис методу розв’язання 47
7 ТЕСТУВАННЯ СИСТЕМИ 52
7.1 Мета тестування 52
7.2 Об'єкти тестування 53
7.3 Результати тестування 54
7.4 Інтерфейс та приклади використання 57
7.5 Порівняльний аналіз результатів маршрутів 61
ВИСНОВКИ 65
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 67
ДОДАТОК А 69

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

A* – алгоритм пошуку найкоротшого шляху з евристикою

ACO – Ant Colony Optimization

API – Application Programming Interface

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

ORS – OpenRouteService

POI – Points of Interest

UI – User Interface

VS Code – Visual Studio Code

ІС – інформаційна система

ВСТУП

У сучасних умовах бурхливого розвитку цифрових технологій особливої актуальності набувають автоматизовані інформаційні системи. Вони дозволяють швидко отримувати доступ до даних, обробляти інформацію й приймати рішення з урахуванням численних факторів. Одним із напрямів, де такі системи демонструють високу ефективність, є побудова маршрутів подорожей. Сучасний темп життя, розмаїття варіантів поїздок і потреба в персоналізованих рішеннях обумовлюють потребу в зручних, гнучких і інформативних інструментах для планування поїздок.

На сьогоднішній день існує низка онлайн-сервісів, які забезпечують базову побудову маршрутів. Втім, більшість із них зосереджені лише на класичних задачах пошуку найкоротшого шляху за відстанню або часом. Ці системи часто не враховують економічні фактори, такі як витрата пального, або туристичну привабливість маршруту (наявність цікавих локацій уздовж шляху). Як наслідок, користувач змушений витрачати значний час на пошук і аналіз даних вручну, що знижує ефективність і якість планування подорожі.

Метою даного дипломного проєкту є створення повноцінної інформаційної системи для побудови маршрутів подорожей, яка надає користувачеві можливість обирати між різними режимами маршрутизації. Реалізовано три основні режими побудови маршруту: найкоротший (оптимізація за довжиною), економний (мінімізація витрат пального) та культурний (максимізація кількості точок інтересу POI). Для перших двох реалізовано класичні алгоритми A* та Dijkstra, а для останнього – адаптований алгоритм колонії мурах із багатокритеріальним підходом, що дозволяє уникнути надмірно довгих та нефункціональних маршрутів.

Користувач взаємодіє із системою через україномовний вебінтерфейс, у якому задає початкову і кінцеву точку маршруту, обирає тип маршруту та вводить дані про транспортний засіб (або середню витрату пального вручну). Результати виводяться у вигляді карти з візуалізацією маршруту та нанесеними точками інтересу. Хоча система не містить модулів для побудови графіків або прямого порівняння альтернативних маршрутів, вона дозволяє інтуїтивно і швидко оцінити виб-

раний варіант за візуальними метриками.

Об’єктом дослідження у дипломному проєкті є процес автоматизованого формування маршрутів подорожей з урахуванням множини факторів.

Предметом дослідження виступають методи багатокритеріальної маршрутизації на графах, побудова та аналіз геопросторових даних, а також архітектурні рішення при створенні інтерактивних веборієнтованих систем планування маршрутів.

Основні задачі, які було вирішено під час розробки системи:

– аналіз предметної області та визначення основних потреб користувача;

– огляд існуючих інструментів для побудови маршрутів та виявлення їхніх обмежень;

– формалізація технічних і функціональних вимог до нової системи;

– вибір і реалізація відповідних алгоритмів маршрутизації;

– створення архітектури застосунку, модульна розробка та налагодження;

– тестування системи на прикладах маршрутів у Києві;

– створення супровідної документації та інструкцій для користувача.

Дипломний проєкт складається зі вступу, шести основних розділів, висновків, списку використаних джерел і додатків. Загальний обсяг пояснювальної записки становить понад 60 сторінок. Графічна частина включає структурні та логічні діаграми (компонентів, послідовностей, потоків даних, схем взаємодії).

У підсумку, розроблена інформаційна система є прикладом ефективного застосування алгоритмів маршрутизації у реальній задачі планування подорожей. Вона може бути корисною для туристичних сервісів, міських навігаційних платформ або в освітньому процесі як приклад проєктування інтелектуальних систем.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис процесу діяльності

Інформаційна система формування маршрутів подорожей, розроблена в межах дипломної роботи, дозволяє користувачеві будувати персоналізовані маршрути в межах міста Києва за трьома основними критеріями, а саме, найкоротший шлях, найекономніший за витратою пального та маршрут з найбільшою кількістю культурних об’єктів. Вся взаємодія користувача із системою здійснюється через зручний вебінтерфейс.

Користувач на головній сторінці вводить початкову та кінцеву точки маршруту (у вигляді адрес або координат), обирає режим маршруту, а також вводить або обирає модель автомобіля для визначення витрат пального. У разі відсутності даних про авто, витрату пального можна вказати вручну. Після натискання кнопки «Побудувати маршрут», система відправляє запит до бекенду.

На серверній частині система здійснюється наступне: за допомогою OpenRouteService API проводиться геокодування введених користувачем адрес, після чого формується граф дорожньої мережі Києва за допомогою бібліотеки osmnx, до графа додаються ваги ребер відповідно до трьох критеріїв, таких як, довжина, витрати пального, кількість POI, точки інтересу (POI) отримуються з Overpass API, аналізуються та прив’язуються до відповідних ребер графа. Для культурного маршруту додатково виконується нормалізація ваг для реалізації багатокритеріальної оптимізації, яка потрібна для запобігання створення занадто великого та нереалістичного маршруту. Після чого запускається відповідний алгоритм маршрутизації: A* та Dijkstra для найкоротшого та економного, або колонія мурах (ACO) для культурного маршруту. У випадку перших двох маршрутів система визначає кращий алгоритм за кращим результатом, а якщо результат однаковий – за швидшим алгоритмом.

Після побудови маршруту результати передаються у фронтенд, де відображаються на інтерактивній мапі. Користувач бачить маршрут на карті, у вигляді текстового списку команд (де і куди повертати), а також отримує узагальнену інфор-

мацію: довжину маршруту, час у маршруті, приблизну витрату пального та кількість культурних точок уздовж шляху. У культурному режимі POI також візуально нанесені на карту.

Таким чином, система виконує повний цикл маршрутизації, а саме, від введення вихідних параметрів до побудови та візуалізації результату. Увесь обчислювальний процес відбувається локально, із зверненням до зовнішніх сервісів через API. Система не зберігає історії маршрутів і не потребує встановлення бази даних. Основна перевага – можливість побудови гнучких, адаптивних маршрутів за потребами користувача з врахуванням декількох реальних критеріїв одночасно.

1.2 Постановка задачі

1.2.1 Призначення системи

Інформаційна система формування маршрутів подорожей призначена для автоматизованого створення оптимальних маршрутів між заданими точками на основі індивідуальних параметрів користувача, таких як довжина маршруту, орієнтовна витрата пального та щільність культурно-історичних об’єктів (POI) на шляху. Основне призначення системи – надати користувачеві гнучкий, інтуїтивно зрозумілий інструмент, який дозволяє не лише швидко отримати маршрут, але й обрати його відповідно до конкретної мети подорожі, а саме, заощадження пального, мінімізація часу, або культурне насичення.

Завдяки реалізації у вигляді вебдодатку з інтерактивним інтерфейсом, система забезпечує легкий доступ з будь-якого пристрою, що підтримує браузер, та не потребує встановлення додаткового програмного забезпечення. Користувач може самостійно задати всі необхідні параметри, а результат отримає у вигляді наочної мапи з інформацією про відстань, витрати та кількість точок інтересу.

Наукова цінність полягає у поєднанні методів маршрутизації з багатокритеріальною оптимізацією, де в рамках культурного режиму використовується алгоритм колонії мурах з урахуванням трьох нормалізованих критеріїв. Такий підхід дозволяє уникнути переобтяжених маршрутів і забезпечити баланс між практичні-

стю та культурною цінністю подорожі. Крім того, система демонструє застосування геоінформаційних технологій та API інтеграцій для створення розширюваного інструменту, що може бути основою для майбутніх досліджень і вдосконалень у сфері інтелектуальних транспортних систем та туристичної навігації .

1.2.2 Цілі та задачі розробки

Основною метою розробки є створення гнучкої, модульної та розширюваної інформаційної системи, яка дає змогу будувати індивідуалізовані маршрути, що допоможе користувачу отримувати маршрути з урахуванням параметрів таких як витрата пального, довжина маршруту, а також кількість культурних точок.

Для досягнення цієї мети необхідно вирішити наступні задачі:

а) провести аналіз предметної області, існуючих рішень та сформувати вимоги до майбутньої системи;

б) побудувати граф дорожньої мережі на основі OpenStreetMap для міста Києва із застосуванням бібліотеки osmnx;

в) інтегрувати зовнішні API:

1) OpenRouteService API – для геокодування та початкової маршрутизації;

2) Overpass API – для отримання POI (музеї, театри, пам’ятки культури тощо);

3) fueleconomy.gov API – для отримання середньої витрати пального за моделлю авто;

г) реалізувати алгоритми побудови маршруту:

1) A* та Dijkstra – для оптимізації за мінімальною відстанню або витратою пального;

2) ACO (алгоритм колонії мурах) – для пошуку маршруту з максимальним охопленням POI з урахуванням довжини та витрат;

г) реалізувати вебінтерфейс на Flask, HTML/CSS з можливістю вибору параметрів маршруту, виведення карти та POI;

д) забезпечити багатокритеріальність шляхом нормалізації даних і побудови зваженої метрики для АСО;

е) протестувати систему в межах міста Києва на прикладах реальних маршрутів, оцінити швидкодію, точність і зручність інтерфейсу.

Таким чином, система поєднує в собі алгоритмічні, геоінформаційні та вебтехнології, що забезпечують побудову маршрутів з урахуванням кількох різних, але взаємозалежних факторів. Це дозволяє формувати гнучкі, ефективні та релевантні до потреб користувача маршрути подорожей.

Висновки до розділу 1

У даному розділі було представлено опис предметної області, в межах якої функціонує система. Детально розглянуто процес планування подорожей, особливості діяльності користувача, який взаємодіє із системою, та роль інформаційних технологій у спрощенні цього процесу.

Постановка задачі включала визначення цілей створення системи, формування переліку задач, які потрібно вирішити в межах розробки, а також опис функціональності, яка реалізується в системі. Було показано, що система враховує декілька важливих критеріїв при побудові маршруту, зокрема довжину, витрати пального та культурні точки, що забезпечує її практичну цінність для користувача.

Сформульовані вимоги до архітектури, алгоритмів, засобів візуалізації та джерел даних створюють основу для подальшого конструювання технічного, математичного та програмного забезпечення. Отримані результати доводять актуальність і доцільність створення подібної системи в умовах зростаючого попиту на індивідуалізовані цифрові сервіси для подорожей.

Таким чином, система поєднує в собі алгоритмічні, геоінформаційні та вебтехнології, що забезпечують побудову маршрутів з урахуванням кількох різних, але взаємозалежних факторів. Це дозволяє формувати гнучкі, ефективні та релевантні до потреб користувача маршрути подорожей.

2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Сьогодні на ринку інформаційних технологій існує безліч програмних продуктів і вебсервісів, які дозволяють користувачам будувати маршрути для різноманітних потреб – від туристичних подорожей до щоденних поїздок містом. Ці сервіси широко застосовуються у сфері навігації, логістики, громадського транспорту, кур'єрських служб і персонального планування маршрутів. Більшість із них мають зручний інтерфейс і забезпечують базовий функціонал – прокладання маршруту між двома точками з урахуванням дорожньої ситуації. Однак при більш детальному аналізі виявляється, що більшість існуючих рішень не враховують важливих додаткових критеріїв, які мають значення для користувачів – таких як витрата пального, наявність пам'яток або культурно значущих об'єктів (POI), та можливість комбінованого аналізу кількох параметрів одночасно.

У цьому розділі проведено огляд найпоширеніших навігаційних і туристичних систем, із зазначенням їхніх функціональних переваг та обмежень, щоб підкреслити потребу у створенні більш гнучкої й адаптивної системи маршрутизації. Ось деякі з них:

– Google Maps;

– OpenRouteService;

– Sygic Travel;

– Waze.

Google Maps – один із найпопулярніших сервісів маршрутизації, який пропонує побудову маршрутів для різних видів транспорту: автомобілів, пішоходів, велосипедистів і громадського транспорту. Він інтегрується з іншими сервісами Google, підтримує голосову навігацію, відображає затори та дорожні події в реальному часі [1].

Переваги:

– висока точність картографічних даних;

– регулярне оновлення інформації в режимі онлайн;

– кілька варіантів маршруту з урахуванням трафіку;

– інтеграція з екосистемою Google (пошта, календар, пошук тощо);

– підтримка мобільних пристроїв.

Недоліки:

– неможливість врахування витрати пального при побудові маршруту;

– відсутність режиму культурної маршрутизації або фільтрації за POI;

– обмежені можливості оптимізації за кількома критеріями одночасно;

– відсутність врахування індивідуальних обмежень користувача.

OpenRouteService (ORS) – це потужна open-source платформа, побудована на основі даних OpenStreetMap. Вона підтримує різні режими маршрутизації, включно з маршрутами для велосипедистів, людей з обмеженими можливостями, вантажного транспорту тощо. Особливою перевагою ORS є доступність через API, що дозволяє розробникам створювати власні інтеграції та рішення. ORS поєднує open source підхід з відкритими стандартами, що забезпечує гнучкість інтеграції [2].

Переваги:

– відкритий вихідний код і безкоштовне використання;

– широкі можливості API (пошук, маршрути, ізохрони, інструкції);

– адаптація до різних типів транспортних засобів і потреб;

– можливість розширення під проєктні задачі;

– активна підтримка спільноти розробників.

Недоліки:

– складність використання для пересічного користувача без технічної підготовки;

– відсутність нативної підтримки POI у маршрутах;

– обмежена реалізація багатокритеріального підходу без додаткової обробки.

Sygic Travel – це туристичний застосунок, орієнтований на користувачів, які хочуть створити маршрут із культурно-пізнавальним наповненням. Застосунок дозволяє планувати поїздки по містах і країнах, додавати до маршруту готелі, пам’ятки, музеї та визначати орієнтовну тривалість кожної зупинки [3].

Переваги:

– зручне візуальне планування подорожі;

– велика база POI з фотографіями та описами;

– можливість попереднього перегляду маршруту на мапі;

– підтримка режимів офлайн-доступу.

Недоліки:

– відсутність розрахунку витрати пального або її оцінки;

– більшість розширених функцій є платними;

– обмежений контроль над логікою формування маршруту.

Waze – це мобільний застосунок, який працює на основі принципу краудсорсингу, користувачі самостійно надсилають дані про дорожню ситуацію (затори, аварії, поліцейські пости тощо) [4]. Це дозволяє оперативно оновлювати маршрути, уникаючи заторів та небажаних ділянок.

Переваги:

– актуальні дані про ситуацію на дорогах у реальному часі;

– побудова оптимального маршруту з урахуванням заторів;

– інтерактивність і активна участь спільноти водіїв;

– простий і ефективний інтерфейс.

Недоліки:

– неможливість врахування культурної цінності маршрутів;

– відсутність системи оцінювання POI або рекомендацій на маршруті;

– слабка підтримка туристичних функцій;

– не передбачено кастомізації цілей маршруту.

Проаналізувавши найпопулярніші існуючі сервіси, можна зробити висновок, що більшість із них орієнтовані на виконання однотипних, спрощених задач маршрутизації. Вони зручно вирішують повсякденні навігаційні потреби, але мають низку істотних обмежень, які роблять їх недостатньо ефективними в умовах складного або культурно насиченого планування подорожі. Жоден з оглянутих сервісів не пропонує поєднання оптимізації за витратою пального, довжиною маршруту та кількістю туристичних точок у рамках єдиного інтерфейсу. Це вказує на потребу в розробці нової, більш гнучкої системи, яка б поєднувала класичну маршрутизацію з культурними та економічними аспектами, дозволяючи користувачу обирати тип

маршруту залежно від своїх уподобань і завдань подорожі.

Висновки до розділу 2

У цьому розділі було проведено систематичний аналіз наявних сервісів маршрутизації, таких як Google Maps, OpenRouteService, Sygic Travel, Waze. Вивчено їхні функціональні можливості, переваги та обмеження з точки зору кінцевого користувача. Особливу увагу приділено відсутності підтримки багатокритеріальної маршрутизації, неможливості обрахунку витрати пального, а також слабкій інтеграції з культурно-туристичними об’єктами (POI).

Отже, на основі аналізу сформульовано чітку потребу у створенні нової інформаційної системи формування маршрутів подорожей. Така система повинна бути гнучкою, адаптивною, із зручним вебінтерфейсом, здатною враховувати кілька критеріїв одночасно, та інтегрувати сучасні алгоритми оптимізації та зовнішні API. Вона має заповнити існуючу функціональну нішу, забезпечивши не лише навігаційний інструмент, а й інтелектуального помічника для туристів і мандрівників, орієнтованого на якісне планування подорожі.

3 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ

3.1 Вимоги до системи в цілому

Після детального аналізу предметної області, порівняння наявних рішень, а також врахування технічних і функціональних можливостей реалізованого програмного продукту, сформульовано розширений набір вимог до розроблюваної інформаційної системи формування маршрутів подорожей. Ці вимоги охоплюють функціональну частину, нефункціональні характеристики, технічні вимоги, а також вимоги до тестування. Основною метою формування вимог є забезпечення високої якості, зручності використання, гнучкості та точності побудови маршрутів для користувачів з різними цілями – від мінімізації витрат до туристичних потреб.

3.1.1 Функціональні вимоги

Система повинна забезпечувати можливість побудови маршруту між початковою і кінцевою точками, введеними користувачем, з урахуванням заданих параметрів, включаючи тип маршруту, транспортний засіб, витрати пального та інші фактори.

Інтерфейс повинен забезпечувати вибір одного з трьох основних режимів маршрутизації:

– найкоротший, мінімізація довжини маршруту;

– економний, мінімізація витрати пального на основі технічних характеристик авто;

– культурний, максимізація кількості точок інтересу (POI) з урахуванням витрат і довжини маршруту.

Інтерфейс має дозволяти вводити координати або адреси початку й завершення маршруту, вибирати модель автомобіля або вказувати витрату пального вручну.

Всі результати повинні виводитися на інтерактивній карті з можливістю перегляду POI, позначених на маршруті.

Для культурного режиму має використовуватись багатокритеріальний підхід, який комбінує нормалізовані значення довжини, витрат пального та кількості POI у формулу вагової метрики.

Алгоритми побудови маршруту повинні автоматично обиратись системою залежно від обраного користувачем режиму, а саме, A* та Dijkstra – для короткого та економного маршруту, ACO (алгоритм колонії мурах) – для культурного маршруту з багатокритеріальністю.

API fueleconomy.gov повинне використовуватись для автоматичного отримання інформації про витрати пального, якщо користувач обирає автомобіль за маркою, моделлю та роком випуску.

Система повинна повертати коротке текстове резюме маршруту, що включає, довжину маршруту, розрахункову витрату пального, кількість POI.

Система повинна підтримувати можливість зміни будь-якого параметра користувачем без необхідності повного оновлення сторінки.

Усі поля повинні перевірятись на коректність вводу з виведенням відповідних повідомлень про помилки.

Система повинна забезпечувати швидке перемикання між режимами без потреби повторного введення всіх параметрів.

3.1.2 Нефункціональні вимоги

Інтерфейс має бути україномовним, інтуїтивно зрозумілим і підтримувати локалізацію.

Система повинна мати адаптивний дизайн, оптимізований для перегляду як на настільних ПК, так і на мобільних пристроях.

Час обробки запиту не повинен перевищувати 5 секунд для стандартного запиту в межах одного міста.

Архітектура програмного забезпечення має бути модульною, окремі модулі для маршрутизаторів, API-клієнтів, обробки POI, візуалізації карт тощо.

Перевірка правильності вводу даних повинна виконуватись на стороні кліє-

нта та сервера для підвищення стабільності та безпеки.

Система має працювати локально без залежності від централізованої бази даних.

Система повинна мати можливість масштабування, а саме, підтримка нових режимів маршрутизації, інтеграції з іншими сервісами, підключення нових джерел POI.

Вебінтерфейс повинен відповідати сучасним стандартам UI/UX – чітке меню, логічна структура переходів, візуальні підказки.

У разі помилок (відсутність API-зв'язку, проблеми з геокодуванням, відсутність маршруту) система має надавати зрозумілі пояснення та рекомендації для виправлення ситуації.

Забезпечення безпеки – система не повинна зберігати персональні дані користувачів, усі обчислення мають бути тимчасовими, без логування запитів.

3.1.3 Вимоги до тестування

Система повинна бути протестована на маршрутах у межах різних районів Києва з різною довжиною, складністю та кількістю POI.

Повинно бути проведено функціональне тестування всіх трьох режимів маршрутизації (короткий, економний, культурний). Необхідно перевірити коректність обробки помилок при відсутності зв'язку з API або некоректному введенні даних. Результати маршрутизації повинні бути верифіковані вручну для перевірки відповідності реальним маршрутам на карті. Система повинна стабільно працювати при зміні параметрів запиту без перезавантаження сторінки.

Тестування UI повинно включати перевірку адаптивності інтерфейсу на різних пристроях (ПК, планшет, смартфон).

Повинно бути здійснено тестування продуктивності, а саме, час відповіді, стабільність при одночасному запуску декількох сесій.

Здійснюється тестування коректності візуалізації результатів та правильності нанесення POI на карту.

3.1.4 Вимоги до технічного забезпечення

Система повинна запускатись на локальному середовищі розробки або звичайному ПК з ОС Windows/Linux/MacOS. Необхідно мати встановлене середовище розробки Python (версія 3.10 або вище), а також залежності: Flask, osmnx, networkx, folium, requests, dotenv, heapq, time, random, math.

Для інтерактивного інтерфейсу необхідний веббраузер із підтримкою HTML5 та JavaScript (Google Chrome, Firefox, Edge).

Система потребує інтернет-доступу для отримання даних з API.

Висновки до розділу 3

У цьому розділі було сформульовано розгорнутий набір вимог до інформаційної системи формування маршрутів подорожей. Визначено ключові функціональні вимоги, які охоплюють підтримку трьох основних режимів маршрутизації (найкоротший, економний, культурний), використання зовнішніх API, побудову маршруту на основі введених користувачем параметрів, а також інтерактивну візуалізацію результатів. Особливу увагу приділено багатокритеріальному підходу, що реалізується у культурному режимі за допомогою алгоритму колонії мурах.

У межах нефункціональних вимог окреслено вимоги до архітектури, інтерфейсу, адаптивності дизайну, часу обробки запитів, масштабованості та безпеки. Також було визначено вимоги до тестування функціональних можливостей системи та стабільності її роботи, включно з перевіркою продуктивності, обробкою помилок та UI-тестуванням. Надано специфікацію до технічного забезпечення, необхідного для локального розгортання системи, з можливістю її подальшого перенесення на сервер або контейнерне середовище.

Сформульовані вимоги є основою для побудови ефективної, гнучкої та зручної у використанні інформаційної системи, яка дозволяє задовольнити запит сучасного користувача на персоналізоване планування маршрутів. Їх чітке дотримання забезпечує якість, масштабованість і відповідність розробки поставленим задачам.

4 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

4.1 Вибір та обґрунтування технологій розробки

У процесі створення інформаційної системи формування маршрутів подорожей було здійснено ретельний аналіз сучасних технологій, інструментів і мов програмування для вибору тих рішень, які найкраще відповідали функціональним вимогам, вимогам до продуктивності, масштабованості, простоти інтеграції та підтримки з боку спільноти. Обрані технології дозволили реалізувати ефективну, зручну в розгортанні та адаптивну до розширення систему.

4.2 Мова програмування та середовище розробки

Для реалізації серверної частини інформаційної системи формування маршрутів подорожей було обрано мову програмування Python. Цей вибір зумовлений низкою факторів, які роблять Python одним із найзручніших та найефективніших інструментів для створення прототипів, реалізації алгоритмічної логіки та обробки просторових даних. Python має простий, лаконічний синтаксис, що дозволяє зменшити обсяг коду, підвищити його читабельність і спростити процес розробки. Це особливо важливо для інженерних і наукових застосунків, де критичним є швидке впровадження та тестування алгоритмів.

Python підтримує розробку об'єктно-орієнтованих, процедурних і функціональних програм, що робить його гнучким у виборі архітектурного підходу. У контексті даного проєкту це дозволило поєднати логіку побудови маршруту, обробку геоданих, взаємодію з API та реалізацію вебінтерфейсу в єдиній технологічній екосистемі. Розробку системи здійснено мовою Python версії 3.11 [5].

Python має велику екосистему бібліотек, що безпосередньо підтримують функціональність, необхідну в проєкті:

– бібліотека osmnx забезпечує завантаження графа дорожньої мережі міста з OpenStreetMap;

– networkx дозволяє зручно маніпулювати графовими структурами;

– math, heapq, random, time забезпечують необхідну обробку чисел, пріоритетів, ітерацій та часових вимірів;

– Flask інтегрується як вебфреймворк безпосередньо в Python, дозволяючи уникати необхідності розділення середовищ і мов.

Ще однією ключовою перевагою Python є наявність потужної спільноти розробників і доступної документації, що значно пришвидшує вирішення технічних питань, дозволяє швидко знаходити приклади коду, плагіни, адаптувати готові рішення до власних задач. Завдяки цьому розробка проєкту велася швидкими ітераціями з постійним покращенням якості рішення.

Як середовище розробки було обрано Visual Studio Code (VS Code) – потужний, безкоштовний та кросплатформений редактор коду, який підтримує всі необхідні інструменти для розробки на Python. Серед його ключових переваг можна виділити гнучку підтримку плагінів, інтеграцію з системами контролю версій (Git), вбудовану підтримку налагодження, форматування коду, автодоповнення, візуальну індикацію синтаксичних і логічних помилок.

VS Code забезпечив можливість створення структурованого та масштабованого середовища розробки, в якому окремі модулі системи могли бути організовані у вигляді незалежних компонентів. Це спростило управління кодом, дало змогу паралельно працювати з бекендом, інтерфейсом та логікою API-взаємодії.

Таким чином, вибір Python як мови програмування у поєднанні з VS Code як середовища розробки, повністю виправдав себе. Було досягнуто високої швидкості розробки, стабільності, розширюваності та інтегрованості всіх компонентів системи у єдине цілісне рішення. Це забезпечило основу для подальшого вдосконалення та масштабування інформаційної системи маршрутизації.

4.3 Вебфреймворк та архітектура

Для реалізації вебінтерфейсу, серверної логіки та обробки API-запитів було обрано мікрофреймворк Flask, що працює на мові Python. Основною причиною такого вибору є простота впровадження, модульність і висока швидкість розробки.

Flask забезпечує достатню гнучкість та масштабованість, дозволяючи створювати повноцінні вебдодатки з чітким розподілом обов'язків між модулями. Серверна частина системи реалізована за допомогою мікрофреймворку Flask [6].

Однією з основних переваг Flask є його мінімалістичність. У базовій версії фреймворк надає лише найнеобхідніше, а все інше додається у вигляді окремих розширень. Це дозволяє розробнику самостійно формувати структуру додатку під специфіку конкретного проєкту, не обмежуючись жорсткими рамками. У нашому випадку це було важливо, оскільки архітектура системи передбачає модульність та можливість підключення додаткових сервісів (API для маршрутів, POI, паливо). Flask дозволяє швидко реалізувати REST API для вебінтерфейсу [7].

Flask підтримує шаблонізатор Jinja2, систему маршрутизації, механізми обробки запитів POST/GET, сесій, авторизації, а також легко інтегрується з іншими Python-бібліотеками. Це дало змогу створити RESTful API, що забезпечує повноцінну взаємодію між фронтендом і бекендом. Обробка HTTP-запитів для побудови маршруту, передачі координат, обчислення витрат пального або культурної цінності маршруту, все це виконується через чітко визначені ендпоїнти, реалізовані у Flask.

Код системи організований у вигляді незалежних модулів:

– маршрутизатори (A*, Dijkstra, ACO);

– генератор графа на основі osmnx;

– модуль обробки POI (Overpass API);

– API-клієнти для зовнішніх сервісів;

– генератор інструкцій маршруту;

– логіка форми та сторінок відображення (Flask + шаблони).

Такий підхід дозволяє розширювати або змінювати функціональність кожного модуля без потреби в істотних змінах у всій системі. Наприклад, можна замінити алгоритм пошуку або додати новий тип маршруту без впливу на логіку побудови форми чи відображення карти. Окрім того, Flask добре підходить для локального розгортання та тестування. Він не потребує складної конфігурації серверного середовища та може працювати у віртуальному середовищі Python, що значно

спрощує налагодження. Завдяки використанню Flask було досягнуто високої продуктивності розробки, чіткого розмежування відповідальності компонентів та можливості подальшого масштабування системи.

Усі ці переваги зробили Flask оптимальним вибором для реалізації веборієнтованої інформаційної системи побудови маршрутів.

4.4 Бібліотеки та пакети

Для реалізації всіх основних функціональних модулів системи було використано набір спеціалізованих бібліотек, які забезпечують ефективну роботу з графами, обробку геоданих, виконання алгоритмічних обчислень і підтримку системних налаштувань.

Osmnx – дозволяє завантажувати граф дорожньої мережі міста Києва з OpenStreetMap, виконувати попередню обробку, фільтрацію та збереження геоданих. Бібліотека також має інструменти для візуалізації графа та підтримує роботу з геометричними атрибутами доріг. Її використання дозволяє автоматизувати побудову топологічної структури карти, що значно знижує час попередньої підготовки даних. Для завантаження графів дорожньої мережі використано osmnx [8].

Networkx – використовується для побудови, обробки та аналізу графів. Саме з її допомогою реалізовано базові алгоритми маршрутизації, як-то A*, алгоритм Dijkstra та адаптований алгоритм колонії мурах. Бібліотека дозволяє присвоювати ваги ребрам, виконувати ітераційні обчислення, знаходити найкоротші шляхи, а також зручно візуалізувати та перевіряти результати обчислень.

Heapq – модуль стандартної бібліотеки Python, що використовується для реалізації черги з пріоритетом. Це необхідно для ефективного виконання пошуку в алгоритмах A* і Dijkstra, де важливо швидко вибирати вершину з найменшою поточною вартістю. Його інтеграція дозволила значно прискорити час виконання алгоритмів пошуку.

Random, time, math – набір допоміжних бібліотек, що забезпечують генерацію випадкових чисел (наприклад, для початкової ініціалізації колонії мурах), за-

міри продуктивності, математичні розрахунки та інші технічні задачі, необхідні в алгоритмах маршрутизації.

Dotenv – дозволяє безпечно зберігати конфіденційні налаштування (зокрема API-ключі) у вигляді змінних середовища. Це полегшує розгортання системи в різних середовищах (тестовому, локальному, у Docker) без зміни основного коду та підвищує безпеку розробки.

Вибір зазначених бібліотек був обумовлений їхньою сумісністю з іншими компонентами проєкту, стабільністю роботи, продуктивністю та наявністю розширеної документації. Усі інструменти активно підтримуються спільнотою, що забезпечує можливість гнучкого масштабування та подальшого розвитку системи. Їх використання дозволило зосередитись безпосередньо на розробці логіки маршрутизації та оптимізації, а не на низькорівневій реалізації базових функцій.

4.5 API та зовнішні сервіси

Для забезпечення повноцінного функціонування інформаційної системи формування маршрутів подорожей була реалізована інтеграція з кількома зовнішніми API. Їх використання дало змогу значно розширити можливості системи без необхідності розробки власних сервісів для збору географічної, туристичної та технічної інформації. Завдяки цим API система отримує доступ до достовірних, актуальних та багатогранних даних, що дозволяє реалізувати складну логіку маршрутизації, яка враховує різноманітні фактори, від географічного положення до характеристик транспортного засобу й культурної цінності об’єктів.

OpenRouteService API – це основне джерело геокодування та побудови базових маршрутів у системі. ORS забезпечує потужні інструменти для аналізу доступності [9]. Однією з ключових переваг ORS є його здатність швидко та точно перетворювати введені користувачем адреси у координати широти та довготи, що критично важливо для коректної побудови графа дорожньої мережі. Крім того, API надає можливість створювати маршрути з урахуванням типу транспортного засобу, а також надає дані про відстань, тривалість подорожі, геометрію шляху та інші ме-

тадані. Додатковою перевагою є підтримка численних режимів маршрутизації (наприклад, пішохідний, велосипедний, автомобільний). Причина вибору ORS полягає у високій стабільності його роботи, підтримці глобального покриття, відкритому доступі, сумісності з Python, широкій документації та підтримці GeoJSON-формату, який зручно інтегрується з картографічними бібліотеками. Геокодування адрес і маршрутизацію реалізовано через OpenRouteService API [10].

Overpass API – використовується для отримання точок інтересу (POI) уздовж маршруту, таких як музеї, архітектурні пам’ятки, історичні об’єкти тощо. Його головна перевага полягає у здатності здійснювати гнучкі та складні запити до бази OpenStreetMap, дозволяючи отримувати лише релевантну інформацію. Зокрема, використовується фільтрація за тегами типу «tourism=museum», «historic=monument», що дозволяє адаптувати вибір об’єктів під різні типи маршрутів. Отримані POI додаються до структури графа, де використовуються як вагові коефіцієнти при побудові маршруту у культурному режимі. POI отримуються з OpenStreetMap через Overpass API [11]. Вибір Overpass API обґрунтовано відкритістю, частим оновленням бази даних, простотою обробки JSON-відповідей, а також сумісністю з Python та бібліотекою requests. Його використання значно підвищує наукову цінність системи, оскільки дозволяє реалізувати багатокритеріальну маршрутизацію на основі культурного насичення шляху.

Fueleconomy.gov API – слугує джерелом достовірних даних про витрату пального для автомобілів різних марок, моделей та років випуску. Його головною перевагою є наявність офіційних даних, отриманих на основі тестів Агентства з охорони довкілля США (EPA), що забезпечує високу точність і достовірність показників. Інформація використовується для розрахунку витрат пального в економному режимі побудови маршруту. Це дозволяє користувачеві враховувати не лише відстань, а й реальні витрати ресурсу, що особливо актуально з огляду на зростання цін на паливо та екологічні міркування. Вибір API обумовлений простим механізмом інтеграції, підтримкою REST-запитів, безкоштовним доступом, структурованим JSON-виводом та зручністю обробки результатів у середовищі Flask.

Причини вибору саме цих API полягають у їхній відкритості, стабільності,

високій точності даних, широкому спектрі функціоналу, доступності та сумісності з архітектурою обраної системи. Всі вони дозволяють будувати ефективну, масштабовану та гнучку логіку маршрутизації з урахуванням як класичних (довжина, час), так і додаткових (POI, витрата пального) параметрів. Їхнє використання спростило реалізацію складних обчислювальних і пошукових задач, зробивши систему більш динамічною та адаптивною до запитів користувачів.

Інтеграція з цими зовнішніми сервісами є ключовим компонентом інформаційної системи. Вона забезпечує динамічну побудову маршрутів, адаптовану до реальних умов та індивідуальних параметрів користувача. Крім того, вона підвищує рівень точності та прикладної користі системи, відкриваючи можливість для її подальшого використання в туристичних, освітніх або аналітичних цілях.

4.6 Інтерфейс користувача

Інтерфейс користувача є однією з найважливіших складових інформаційної системи формування маршрутів подорожей, адже саме він забезпечує взаємодію між користувачем і програмною логікою. У рамках дипломного проєкту було реалізовано інтуїтивно зрозумілий, зручний та адаптивний вебінтерфейс, що дозволяє користувачу взаємодіяти із системою в кілька кліків. Обрано саме вебформат через його універсальність, достатньо лише сучасного браузера, що робить систему доступною з будь-якого пристрою, не потребуючи встановлення додаткового ПЗ.

Інтерфейс побудовано на основі стандартних вебтехнологій: HTML5, CSS, а також шаблонізатора Jinja2, що інтегрований у фреймворк Flask [12]. Цей підхід дозволяє ефективно генерувати динамічні сторінки з урахуванням параметрів, які вводить користувач. HTML5 відповідає за структуру документу, CSS за візуальне оформлення й адаптивність, а Jinja2 за інтеграцію змінних і умов логіки прямо в шаблони. Обрана зв'язка технологій забезпечує високу гнучкість та розширюваність інтерфейсу без ускладнення архітектури.

Головна форма розміщена на стартовій сторінці. Вона дозволяє користувачеві:

– ввести адресу початкової точки маршруту;

– ввести адресу кінцевої точки маршруту;

– вибрати тип маршруту (найкоротший, економний або культурний);

– обрати марку, модель та рік випуску автомобіля або вказати витрату пального вручну.

Такий рівень налаштування параметрів дозволяє персоналізувати результат, зробити його максимально релевантним до потреб користувача. Наприклад, якщо користувач бажає зекономити пальне, він може вибрати економний режим та ввести точні параметри свого авто, що дозволяє враховувати витрати не лише за відстанню, а й за реальним споживанням пального.

Після надсилання форми, на сторінці результатів динамічно відображається:

– інтерактивна мапа з маршрутом;

– інформаційна панель із тривалістю, довжиною маршруту та витратою пального;

– підрахунок кількості пам’яток для культурного маршруту;

– інструкції у вигляді покрокових вказівок (наприклад: «поверніть праворуч на вулицю...»);

– елемент керування для запуску нового запиту.

Відображення маршруту реалізовано за допомогою бібліотеки Folium, яка базується на Leaflet.js. Цей інструмент був обраний завдяки повній сумісності з Python та можливості виводити GeoJSON-дані безпосередньо з backend-частини. Folium дозволяє створити інтерактивну карту з підтримкою зумування, прокручування, клацання по маркерам (POI), що значно покращує зручність сприйняття результату. Маршрут відображається кольоровою лінією, POI – маркерами з підказками, а також чітко позначені стартова і кінцева точки.

Інтерфейс було реалізовано україномовним, оскільки цільовою аудиторією є локальні користувачі. Усі підписи, інструкції та підказки написані українською мовою, що робить систему доступною для широкого кола користувачів, включаючи освітні заклади, органи місцевого самоврядування та туристичні сервіси. Крім цього, локалізований інтерфейс створює зручні умови для поширення системи в

українському інфопросторі.

Інтерфейс користувача тісно пов’язаний із серверною логікою, що дозволяє сформувати повноцінний цикл взаємодії від заповнення форми до обробки даних та відображення результатів. Розділення обов’язків між фронтендом і бекендом дозволяє гнучко масштабувати систему, оновлювати окремі компоненти (наприклад, змінити картографічну бібліотеку або розширити шаблон виводу інструкцій) не змінюючи ядро програми. У майбутньому інтерфейс легко адаптується до мобільних пристроїв, або інтегрується у десктопні додатки чи PWA-рішення, зберігаючи загальну структуру й принципи взаємодії.

Завдяки простоті, гнучкості, локалізації та логічній структурі, інтерфейс забезпечує ефективну, комфортну й результативну взаємодію користувача з інформаційною системою побудови маршрутів подорожей.

Висновки до розділу 4

У результаті аналізу та впровадження обраних технологій можна зробити висновок, що всі компоненти, а саме, мова програмування, середовище розробки, фреймворк, бібліотеки, зовнішні сервіси та інтерфейс були обрані свідомо та обґрунтовано. Їхня сукупність забезпечила реалізацію функціональної, адаптивної та зручної інформаційної системи для побудови маршрутів подорожей. Обрані технології сприяють гнучкій підтримці, простому розширенню та можливості інтеграції додаткових сервісів у майбутньому. Крім того, застосовані рішення дозволили забезпечити високу якість обробки геоданих, оптимальну маршрутизацію та зручність користувача при роботі з інтерфейсом. Усі технологічні компоненти показали себе надійними у процесі реалізації, що підтверджує їх доцільність і придатність для задач подібного типу.

5 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

5.1 Структура системи

У цьому розділі детально описується процес створення, налаштування та впровадження інформаційної системи формування маршрутів подорожей. Основна увага приділяється поетапному опису технічної реалізації кожного з модулів системи, а також тому, як відбувається взаємодія між ними на рівні обміну даними, API-викликів та відображення результатів. Важливим аспектом є розкриття архітектурних рішень, структуризації коду, форматів збереження даних і механізмів масштабованості, які дозволяють легко розширювати систему або адаптувати її під нові задачі.

До складу системи входить кілька взаємопов’язаних блоків, а саме, блок завантаження та побудови графа дорожньої мережі, блок обробки об'єктів інтересу (POI), модуль обчислення витрати пального на основі зовнішнього API, ядро маршрутизаторів з реалізацією трьох алгоритмів (A*, Dijkstra, ACO), а також вебінтерфейс, який слугує засобом взаємодії користувача з функціоналом системи. Кожен з цих блоків виконує окрему роль, але водночас тісно взаємодіє з іншими частинами, формуючи цілісну структуру.

У розділі послідовно розглянуто:

– способи побудови графа на основі відкритих геоданих (OpenStreetMap);

– методи інтеграції зовнішніх сервісів, таких як Overpass API та fueleconomy.gov;

– деталі реалізації кожного алгоритму маршрутизації відповідно до вибраних критеріїв;

– технологічні особливості фронтенд- і бекенд-частини системи.

Також охоплено моменти, пов’язані з ефективністю системи, зокрема збереження обчисленого графа у форматі GraphML, застосування нормалізації ваг для багатокритеріальної маршрутизації, обробку запитів користувача через форму, генерацію інструкцій та вивід інтерактивної карти. Окрему увагу приділено створенню інтерфейсу користувача, який дозволяє легко взаємодіяти з системою, буду-

вати маршрути на основі індивідуальних параметрів і отримувати візуалізований результат.

Усі частини системи були реалізовані з урахуванням попередньо визначених вимог, дотримання принципів модульності та можливості масштабування. Це дозволяє легко оновлювати окремі компоненти, впроваджувати нові джерела даних, або адаптувати систему для інших регіонів чи умов експлуатації.

Структурна схема системи в кресленику ІС12.210БАК.005 Д1 та діаграма компонентів в кресленику ІС12.210БАК.005 Д2.

Діаграма потоків даних наведена в кресленику ІС12.210БАК.005 Д3.

Також було створено діаграму послідовностей, яка була наведена в кресленику ІС12.210БАК.005 Д4.

5.2 Функціональна модель системи

5.2.1 Побудова графа дорожньої мережі

Побудова графа дорожньої мережі стала першим та одним із найважливіших етапів реалізації інформаційної системи формування маршрутів подорожей. Саме ця частина системи відповідає за базову топологію, по якій виконуються всі подальші обчислення від простої маршрутизації до складних багатокритеріальних оптимізацій. Для реалізації було обрано бібліотеку OSMnx, що є потужним інструментом для завантаження, обробки та візуалізації просторових даних з відкритого джерела OpenStreetMap. Вибір саме цієї бібліотеки зумовлений її гнучкістю, широкою підтримкою формату графів, а також зручністю у роботі з Python.

Процес починається з завантаження карти обраного регіону, в нашому випадку це місто Київ. Столиця була обрана з огляду на складну та розгалужену дорожню інфраструктуру, що робить її ідеальним полігоном для тестування алгоритмів маршрутизації, а також через наявність великої кількості культурних та туристичних об’єктів, які пізніше інтегруються в систему як точки інтересу (POI). За допомогою функцій OSMnx виконується автоматичне отримання графа за заданими координатами або адміністративними межами.

Граф будується як орієнтований, тобто кожен сегмент дороги має напрямок руху, що відповідає реальній організації дорожнього руху.

Вузлами графа є координати точок перетину або зміни напрямку дороги, а ребрами відрізки між ними. Це дозволяє точно моделювати реальну інфраструктуру. До кожного ребра додається атрибут довжини, обчислений автоматично на основі геометрії. Ця довжина надалі використовується як основна метрика в алгоритмах A* і Dijkstra, які обирають маршрут з урахуванням найменшої відстані або найнижчої ваги.

Особливу увагу було приділено розширюваності графа. До стандартної структури ребра (довжина, напрямок) були додані додаткові атрибути, витрата пального (fuel_weight) та щільність культурних об'єктів (poi_score). Це стало основою для багатокритеріальної маршрутизації, що враховує не лише відстань, а й інші значущі фактори. Для цього ребра графа модифікували, додаючи ваги на основі зовнішніх обчислень, наприклад, з API витрати пального або з Overpass API, що постачає дані про POI.

Готовий граф зберігається у форматі GraphML, що забезпечує зручність повторного використання без необхідності повторного завантаження та обробки геоданих. Цей формат також дозволяє у разі потреби редагувати граф вручну або переглядати його в сторонніх редакторах.

Для забезпечення точності було реалізовано фільтрацію елементів, видалено недоступні для руху сегменти (пішохідні зони, сервісні під'їзди, технічні дороги) та враховано лише ті шляхи, що відповідають стандартам міського автомобільного руху.

Узагальнюючи, побудований граф є універсальною базовою структурою системи, яка забезпечує її масштабованість, точність та продуктивність. Завдяки його модульності і підтримці різних типів ваг, від простих метричних до аналітичних, система готова до інтеграції нових джерел даних, підтримки різних типів транспортних засобів, а також до подальшого розвитку функціоналу інтелектуальної маршрутизації.

5.2.2 Інтеграція даних POI

Для побудови культурних маршрутів у системі критично важливо враховувати наявність об’єктів інтересу (Points of Interest – POI), що розташовані вздовж потенційного шляху. Саме ці об’єкти музеї, пам’ятки архітектури, історичні монументи, культурні центри, галереї, релігійні споруди тощо формують унікальну цінність маршруту для користувача, який орієнтується не лише на швидкість чи економічність, а й на змістовність подорожі. Тому інтеграція даних POI стала ключовим елементом розробки культурного режиму маршрутизації в системі.

З технічного боку реалізація цієї функціональності відбулася через підключення до Overpass API потужного інструменту, що надає доступ до гнучких запитів даних з OpenStreetMap. API дозволяє здійснювати пошук об’єктів за конкретними тегами в межах просторового буфера навколо заданих координат. У системі формуються запити, які охоплюють маршрутні сегменти з урахуванням фіксованого радіуса (наприклад, 500 метрів) і повертають об’єкти з тегами типу tourism=museum, historic=monument, amenity=place_of_worship тощо.

Після отримання даних система обробляє відповіді Overpass API та асоціює знайдені POI з відповідними відрізками графа. Для цього реалізовано підрахунок кількості об’єктів, що потрапляють у зону дії кожного ребра. Отримане значення додається до структури графа як новий атрибут ребра (poi_score), що репрезентує інформативність або культурну насиченість даного сегменту.

Зважаючи на те, що система реалізує багатокритеріальну маршрутизацію, значення POI не можуть використовуватись у сирому вигляді. Для цього було впроваджено механізм нормалізації, усі poi_score автоматично перетворюються у шкалу від 0 до 1, залежно від максимального значення в межах графа. Таке рішення забезпечує узгодженість метрик при побудові комбінованих функцій вартості, де також враховуються довжина маршруту (distance_weight) та витрата пального (fuel_weight).

Окрему увагу було приділено ефективності запитів до Overpass API. Щоб уникнути надмірного навантаження та затримок, реалізовано кешування POI-ре-

зультатів у межах певного регіону, що дає змогу використовувати повторно результати для близьких маршрутів. Також застосовано групування вузлів у батч-запити, що зменшує кількість звернень до API.

Інтеграція POI значно підвищила функціональність системи, зробивши її не лише технічною, але й змістовно орієнтованою. Користувач отримав змогу будувати маршрути з орієнтацією на зміст, а не лише на логістику, що особливо цінно для туристичних та освітніх застосунків. Маршрути з великою кількістю POI можуть бути пріоритетними при плануванні екскурсій, тематичних подорожей або культурних програм.

Крім того, вибір саме алгоритму колонії мурах (ACO) для реалізації культурного маршруту був зумовлений потребою в обробці декількох критеріїв одночасно. ACO дозволяє враховувати культурну насиченість маршруту поряд із його довжиною та витратою пального, забезпечуючи баланс між змістовністю та практичністю.

Таким чином, реалізація механізму інтеграції POI відкрила нові перспективи розвитку інформаційної системи. Надалі можливо розширити категорії об’єктів (наприклад, гастрономічні заклади, природні парки, технічні пам’ятки), дозволити користувачу самостійно обирати типи точок інтересу або ж впровадити індивідуальні сценарії подорожей на основі попередніх вподобань. Це створює потенціал для персоналізованих маршрутів, що значно підвищує прикладну цінність системи.

5.2.3 Обчислення витрати пального

У рамках реалізації економного режиму маршруту в інформаційній системі критично важливою складовою є точне та коректне обчислення витрати пального. Цей параметр є одним із ключових чинників при виборі оптимального шляху за критерієм мінімізації витрат. Його врахування дозволяє користувачеві не тільки побудувати маршрут, а й оцінити фінансову доцільність поїздки. Таким чином, цей функціональний блок має як прикладне значення для щоденного використання, так і навчальну цінність у контексті вивчення задач оптимізації.

Для отримання достовірних даних щодо середньої витрати пального було реалізовано інтеграцію з зовнішнім сервісом fueleconomy.gov. Це офіційне джерело інформації про технічні характеристики автомобілів у США, що працює через REST API та забезпечує доступ до актуальної бази даних для широкого спектру моделей. Користувач системи має змогу вибрати зі списку або ввести вручну марку, модель та рік випуску автомобіля. Після цього система надсилає запит до API й отримує значення середньої витрати пального в літрах на 100 км. У разі, якщо дані відсутні або користувач бажає використовувати власне значення, реалізовано альтернативний сценарій, ручне введення значення в спеціальному полі форми.

Після отримання відповідного значення витрати пального система виконує обчислення витрат на кожному сегменті маршруту. Кожне ребро графа дорожньої мережі має атрибут довжини в метрах, який попередньо конвертується в кілометри. Витрата пального у літрах розраховується як добуток довжини сегмента маршруту (в кілометрах) на середню витрату пального в літрах на 100 км, поділений на 100.

Цей процес повторюється для кожного сегмента маршруту, і результати акумулюються у загальне значення. В результаті користувач отримує не тільки маршрут, а й точну оцінку витрати пального по дорозі. Крім того, це значення може бути використано для додаткових розрахунків, наприклад для оцінки вартості подорожі з урахуванням ціни пального.

З технічного боку модуль реалізовано як окремий сервіс у структурі бекенду системи. Він відповідає за отримання параметрів автомобіля з форми, формування та відправку запиту до API, обробку відповіді, кешування результату (для запобігання надмірному навантаженню на зовнішній сервіс) та передачу обчисленого значення до модуля маршрутизації. Кешування реалізовано на рівні сесії, що дозволяє зберігати останні результати протягом обмеженого часу для повторного використання.

Обчислення витрати пального є не лише корисним функціоналом для користувача, але й важливим аналітичним інструментом у процесі побудови економних маршрутів. Це особливо актуально при реалізації алгоритмів, що враховують витрати. Також цей показник враховується в алгоритмі колонії мурах (ACO), що до-

зволяє враховувати витрату пального поряд з іншими критеріями (довжина маршруту, кількість культурних точок).

Таким чином, модуль обчислення витрати пального є повноцінною та критичною частиною інформаційної системи. Він не лише покращує точність побудови маршруту, а й підвищує прикладну користь системи для реального використання. Завдяки гнучкій структурі та можливості налаштування значень, цей модуль може бути адаптований до будь-яких умов експлуатації, включаючи підтримку електромобілів або розрахунків на основі CO_2-викидів у майбутніх версіях.

5.2.4 Реалізація алгоритмів маршрутизації

Інформаційна система формування маршрутів подорожей підтримує три основні режими маршрутизації, кожен з яких реалізовано за допомогою спеціалізованого алгоритму. Ці алгоритми працюють на основі графа дорожньої мережі, побудованої з використанням бібліотеки OSMnx. Вони враховують різні критерії, а саме, довжину маршруту, витрату пального та кількість об’єктів інтересу (POI) при обчисленні оптимального шляху. Вибір конкретного алгоритму залежить від обраного користувачем режиму маршруту.

Алгоритм A* застосовується для побудови найкоротшого маршруту за критерієм відстані та найекономнішим за критерієм витрати палива. Це один із найефективніших евристичних алгоритмів пошуку шляху, який поєднує фактичну відстань від стартової точки до поточної (g_n) з евристичною оцінкою відстані до цілі (h_n). A* є евристичним розширенням Dijkstra та використовує оцінку $f_n = g_n + h_n$ [13]. У цій системі як евристика використовується евклідова відстань між поточним вузлом і цільовим, що дозволяє досягати високої швидкості пошуку при збереженні точності результату. Завдяки алгоритму A* користувач отримує геометрично оптимальний маршрут. Як показано на рисунку 5.1 алгоритм A* працює на основі поєднання фактичної ваги шляху та евристичної оцінки відстані до цільового вузла. Це дозволяє ефективно орієнтувати пошук у графі, мінімізуючи кількість оброблених вершин.

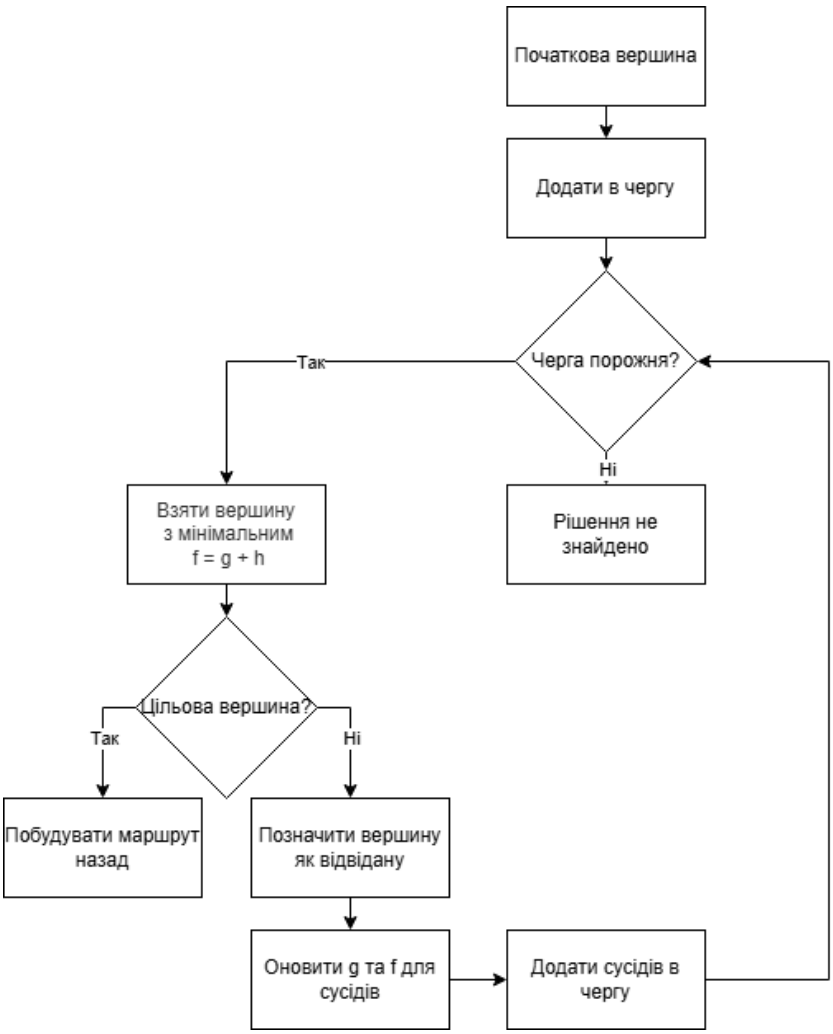

Рисунок 5.1 – Блок-схема алгоритму A*

Алгоритм Dijkstra також використовується для побудови найкоротшого маршруту за критерієм відстані та найекономнішим за критерієм витрати палива. На відміну від A*, у цьому випадку алгоритм не враховує евристику, а лише реальні значення ваг, присвоєні ребрам графа. У кожному ребрі графа зберігається атрибут fuel_weight або lenght, які відображають витрату пального на цьому сегменті з урахуванням довжини ділянки та середньої витрати авто або відстань. Алгоритм проходить усі можливі шляхи та обирає той, де сума ваг є найменшою. Алгоритм Dijkstra дозволяє знайти найкоротший шлях у графі з не негативними вагами [14]. У великих мережах Dijkstra демонструє високу продуктивність [15]. Рисунок 5.2 демонструє класичну реалізацію алгоритму Dijkstra в контексті поставленої задачі. Основна ідея полягає у поступовому розширенні множини вершин з відомою мінімальною вагою, що забезпечує гарантовану оптимальність знайденого шляху.

Рисунок 5.2 – Блок-схема алгоритму Dijkstra

Для реалізації культурного маршруту, що орієнтується на максимальне охоплення пам’яток культури, застосовується алгоритм колонії мурах. ACO – це евристичний метод, натхненний поведінкою реальних мурах, які знаходять оптимальні шляхи до джерела їжі. У нашій системі цей алгоритм адаптований до умов багатокритеріальної маршрутизації. Він одночасно враховує три основні параметри, довжину маршруту, витрату пального та культурну насиченість (poi_score). Кожна з цих метрик нормалізується та включається до комбінованої формули ваги ребра. Після цього група штучних «мурах» імітує проходження графа, поступово вдосконалюючи феромонні сліди та наближаючись до оптимального рішення. Природні метаевристики, зокрема колонія мурах, належать до натхненних природою алгоритмів [16].

Перевага ACO полягає в здатності шукати більш наближені рішення в умовах

великої кількості змінних. Такий підхід дозволяє формувати маршрути з високою кількістю POI, при цьому не жертвуючи надмірно довжиною або економічною ефективністю. Завдяки цій особливості ACO забезпечує унікальний користувацький досвід для мандрівників, які бажають відвідати якомога більше пам'яток. ACO імітує поведінку колонії мурах при пошуку їжі, що дозволяє реалізувати ефективну маршрутизацію за POI [17]. На рисунку 5.3 представлено узагальнену схему алгоритму колонії мурах, який передбачає ітеративну побудову рішень агентами (мурахами) з урахуванням накопиченого досвіду у вигляді феромонів та локальної евристики.

Рисунок 5.3 – Блок-схема алгоритму колонії мурах (ACO)

Отже усі алгоритми реалізовано у вигляді незалежних класів-модулів, які мають уніфікований інтерфейс виклику. Такий підхід дозволяє зберігати модульність архітектури, забезпечувати зручну підтримку коду та відкриває можливості для майбутнього розширення функціональності. Наприклад, у майбутньому можна до-

дати підтримку нового алгоритму (наприклад, генетичного або жадібного пошуку), не змінюючи основної логіки системи. Усі алгоритми використовують спільні компоненти, граф дорожньої мережі, обчислення витрат, нормалізацію метрик, інтерфейс побудови маршрутів та вивід результатів.

Таким чином, реалізація алгоритмів маршрутизації забезпечує гнучкість і функціональну повноту системи. Користувач може обрати маршрут відповідно до своїх пріоритетів: швидкість, економічність або культурна цінність. Такий підхід дозволяє охопити широке коло сценаріїв використання від туризму й побутових поїздок до логістичних задач і освітніх симуляцій. Завдяки підтримці багатокритеріального аналізу та розширюваної архітектури, система має значний потенціал для подальшого розвитку.

5.3 Вивід результатів та інтерфейс

Заключним етапом побудови маршруту в інформаційній системі є вивід результатів у зрозумілому, візуальному та зручному для користувача вигляді. Саме цей етап забезпечує інтуїтивну взаємодію користувача з результатами обчислень та дає змогу ефективно сприймати інформацію про побудований маршрут. Цей компонент реалізовано у вигляді вебінтерфейсу, який поєднує інтерактивну карту, динамічні елементи керування та текстову аналітику результатів.

Основна карта маршруту генерується з використанням бібліотеки Folium, потужного інструмента, побудованого на основі Leaflet.js, що дозволяє формувати інтерактивні мапи з підтримкою різних геооб’єктів. Цей вибір обумовлений простотою інтеграції з Flask і високою гнучкістю у візуалізації. Побудований маршрут виводиться на карту у вигляді кольорової лінії, що чітко позначає шлях від початкової до кінцевої точки. Кінцеві пункти маршруту позначаються спеціальними маркерами з підписами, що дозволяє користувачеві швидко зорієнтуватися у структурі подорожі. При необхідності додаткові шари можуть відображати транспортні вузли, пункти POI або зони з особливими умовами руху.

Окрім графічної візуалізації, система надає користувачу повний набір текс-

тової інформації про побудований маршрут. До неї належать:

– загальна довжина маршруту в кілометрах з точністю до десятих;

– приблизна тривалість подорожі в годинах і хвилинах, розрахована на основі середньої швидкості;

– загальна витрата пального в літрах (якщо активовано режим економії);

– кількість культурних об’єктів (POI), що потрапили в зону маршруту (в культурному режимі);

– детальна покрокова інструкція, яка містить назви вулиць, напрямки поворотів та орієнтири для водія чи мандрівника.

Для реалізації взаємодії між серверною частиною та інтерфейсом застосовується шаблонізатор Jinja2. Він дозволяє гнучко передавати дані з backend (на Flask) до HTML-сторінок і формувати відповідний контент у режимі реального часу. Таким чином, зміна будь-якого параметра у формі (режим маршруту, тип автомобіля, початкова/кінцева точка) миттєво впливає на відображення результату без потреби перезавантаження всієї сторінки. Це значно покращує користувацький досвід та робить систему зручною для щоденного застосування.

Ключовим елементом виводу результатів є інтерактивна карта, згенерована засобами бібліотеки Folium, яка базується на популярній JavaScript-бібліотеці Leaflet.js. Ця карта дозволяє не лише візуалізувати побудований маршрут, а й інтерактивно взаємодіяти з елементами карти, зокрема з точками інтересу (POI). Кожен POI, отриманий із Overpass API, виводиться на мапу у вигляді окремого маркера з підказкою, яка з’являється при наведенні або натисканні. У підказці відображається назва об’єкта, його тип (музей, пам’ятка архітектури, історична будівля тощо), що дозволяє користувачеві оцінити потенційні зупинки та внести корективи до плану поїздки. Розміщення POI здійснюється з урахуванням реальної топології графа дорожньої мережі, на основі якої будувався маршрут, що забезпечує просторову узгодженість усіх даних.

Інтерфейс системи реалізовано повністю українською мовою, що є важливим аспектом для локального користувача. Візуальне оформлення створено з використанням HTML/CSS із дотриманням сучасних стандартів адаптивного дизайну. Це

гарантує коректне відображення сторінки як на настільних ПК, так і на мобільних пристроях із різними розмірами екранів. Окрему увагу приділено зручності навігації, контрастності елементів, читабельності шрифтів та логічності розміщення блоків інформації.

Система також передбачає перемикання між різними режимами маршрутизації – найкоротшим, економічним та культурним – без необхідності повторного заповнення вхідних параметрів. Це реалізовано через інтерактивні кнопки, які дозволяють миттєво змінити критерії побудови маршруту, зберігаючи вже введені дані та оновлюючи карту й аналітику. Такий механізм робить порівняння варіантів маршруту швидким та зручним. Інтерфейс також включає кнопку «Почати новий маршрут», яка очищує форму та карту, забезпечуючи можливість повторного запиту без оновлення сторінки.

Таким чином, модуль виводу результатів не лише виконує роль візуалізатора даних, а й виступає ключовим компонентом, що поєднує аналітичні та практичні функції системи. Поєднання інтерактивної карти, текстових пояснень, адаптивного дизайну та динамічного оновлення виводить зручність користування на високий рівень і робить систему доступною для широкого кола користувачів.

Висновки до розділу 5

У результаті реалізації розділу було побудовано повнофункціональну програмну систему, що охоплює всі ключові етапи маршрутизації, від формування графа дорожньої мережі до обробки даних POI, обчислення витрати пального, реалізації алгоритмів побудови маршрутів та виведення результатів у зручному форматі. Побудова графа забезпечила гнучку основу для аналізу дорожньої інфраструктури міста. Інтеграція POI розширила можливості системи культурними критеріями, а модуль обчислення витрати пального дозволив оцінювати економічність маршрутів. Під час розробки також було виконано успішну інтеграцію з трьома зовнішніми API: OpenRouteService для геокодування адрес, Overpass API для отримання POI-об'єктів уздовж маршруту та FuelEconomy.gov API для автоматичного

визначення витрати пального за характеристиками автомобіля. Це дозволило підвищити рівень автоматизації системи та зменшити кількість ручного введення з боку користувача.

Реалізовані алгоритми A*, Dijkstra та мурашиної колонії (ACO) забезпечили широкий вибір маршрутних стратегій. Зокрема, A* та Dijkstra орієнтовані на найекономніший та найкоротший шляхи, тоді як ACO дозволяє враховувати культурні об’єкти та багатокритеріальність. Такий підхід дозволяє адаптувати систему під різноманітні запити користувачів.

Важливим досягненням стала реалізація повнофункціонального вебінтерфейсу, що поєднує карту, текстові результати, інструменти перемикання режимів та адаптивний дизайн. Це зробило систему придатною до щоденного використання та створило основу для її подальшого розвитку – зокрема, додавання модуля порівняння маршрутів, збереження історії запитів, підтримки мультимовності або підключення нових джерел POI.

Загалом, реалізована система продемонструвала здатність адаптуватися до різних типів задач та користувацьких потреб, створивши підґрунтя для подальшого розширення функціоналу, інтеграції нових джерел даних та масштабування на інші регіони.

6 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

6.1 Змістовна постановка задачі

Інформаційна система формування маршрутів подорожей призначена для побудови індивідуалізованих маршрутів між двома географічними точками, з урахуванням специфічних параметрів і потреб користувача. Основна мета системи – надати гнучкий інструмент, що дозволяє обирати тип маршруту залежно від пріоритету, мінімальна довжина, економія пального або культурна насиченість. Такий підхід значно розширює можливості стандартної маршрутизації, дозволяючи застосовувати інтелектуальні методи аналізу геоданих.

Система реалізує три ключові режими маршрутизації:

а) Найкоротший маршрут – оптимізується за мінімальною геометричною довжиною шляху. Підходить для випадків, коли головна мета швидко дістатися з пункту А до пункту Б.

б) Економний маршрут – орієнтований на мінімізацію витрат пального, що особливо актуально для користувачів із обмеженим бюджетом або в умовах підвищеної вартості пального. Тут критичною є коректна інтеграція API fueleconomy.gov, яке надає середні витрати пального для конкретного автомобіля.

в) Культурний маршрут – враховує не лише довжину та витрату пального, а й кількість культурно значущих об’єктів (POI), таких як музеї, театри, історичні пам’ятки. Для цього система взаємодіє з Overpass API, витягаючи точки інтересу та вбудовуючи їх у граф дорожньої мережі для багатокритеріальної оцінки.

Для реалізації цих режимів система повинна виконувати такі функції:

– приймати вхідні дані через інтерактивний вебінтерфейс (початкова й кінцева точки маршруту, тип авто або витрата пального, режим маршрутизації);

– виконувати геокодування введених адрес за допомогою OpenRouteService API;

– будувати граф дорожньої мережі міста Києва через бібліотеку OSMnx, що дозволяє отримати зважений граф із координатами та атрибутами кожного сегмента дороги;

– збирати дані про POI за допомогою Overpass API та інтегрувати їх у граф як ваговий параметр для культурного режиму;

– обчислювати витрату пального на кожному сегменті маршруту, використовуючи дані з API або вручну задане значення користувачем;

– запускати обраний алгоритм маршрутизації (A* для найкоротшого, Dijkstra для економного, ACO для культурного) на побудованому графі з відповідними метриками;

– виводити результати у вигляді інтерактивної мапи з маркерами POI, підсумковою статистикою маршруту (довжина, тривалість, витрати пального, POI) та покроковими інструкціями для користувача.

Таким чином, система вирішує комплексну задачу маршрутизації з урахуванням різнорідних критеріїв, поєднуючи класичні алгоритми пошуку шляху та сучасні вебтехнології. Завдяки багатофункціональності, вона є корисним інструментом для туристів, водіїв, аналітиків логістики та будь-яких користувачів, які бажають планувати маршрут не лише раціонально, а й змістовно. Такий підхід підвищує якість користувацького досвіду та дає змогу розширити функціональність у майбутньому, наприклад, через додавання нових метрик або підтримку інших міст.

6.2 Математична постановка задачі

Задача побудови маршруту в інформаційній системі формування маршрутів подорожей математично формалізується як задача пошуку оптимального шляху в орієнтованому зваженому графі $G = (V, E)$, де V множина вершин, що відповідають ключовим точкам дорожньої мережі, таким як перехрестя, зупинки або зміни напрямку руху, E множина орієнтованих ребер (дуг), що з'єднують пари вершин і представляють наявні шляхи пересування між ними з відповідними характеристиками.

Граф G створюється на основі просторових даних, отриманих з OpenStreetMap через бібліотеку OSMnx. Він зберігає повну структуру дорожньої мережі, включаючи географічні координати, довжину сегментів, типи доріг та інші

параметри. На основі цього графа виконуються всі обчислення маршрутів.

Кожному ребру $e_{ij} \in E$ призначається вага w_{ij}, яка визначає «вартість» переходу з вершини i до вершини j. Ця вага може відображати один з таких параметрів:

– d_{ij} – геометрична довжина сегмента, обчислена на основі координат вузлів;

– f_{ij} – орієнтовна витрата пального на цьому сегменті, що враховує тип автомобіля та середню витрату пального;

– poi_{ij} – кількість об'єктів інтересу (POI), пов'язаних із сегментом, таких як музеї, пам’ятки, театри або комбіновану метрику, що об'єднує всі вищезазначені фактори з урахуванням вагових коефіцієнтів.

Таким чином, залежно від вибраного користувачем режиму маршрутизації, система буде використовувати відповідну вагову функцію. Для однокритеріальних режимів (найкоротший або економний маршрут) вага визначається безпосередньо довжиною або витратою пального. Для культурного маршруту застосовується багатокритеріальна модель.

Формально, задача полягає в знаходженні маршруту $P = \{v_s, \dots, v_s\}$, що з'єднує початкову вершину v_s із кінцевою вершиною v_t, та мінімізує сумарну вагу шляху згідно з формулою 6.1:

$$W_P = \sum w_{ij}, \tag{6.1}$$

де для всіх ij ∈ P

У випадку багатокритеріального підходу, вага ребра w_{ij} розраховується за узагальненою формулою 6.2:

$$w_{ij} = \alpha \times norm(d_{ij}) + \beta \times norm(f_{ij}) - \gamma \times norm(h_{ij}), \tag{6.2}$$

де $norm_x = \frac{(x - x_{min})}{(x_{max} - x_{min})}$, нормалізоване значення параметра в діапазоні [0,1];

α, β, γ – коефіцієнти ваг, що визначають пріоритет кожного критерію згідно з поточним режимом маршрутизації;

знак «−» перед нормалізованим значенням poi_{ij} використовується, оскільки система мінімізує цільову функцію, а збільшення кількості POI вважається перевагою.

Нормалізація усіх параметрів дозволяє привести різнотипні характеристики (довжина в метрах, витрата пального в літрах, кількість POI) до спільного масштабу, що забезпечує коректну багатокритеріальну агрегацію. У результаті цього підходу користувач отримує маршрут, що найкраще відповідає його запиту з точки зору швидкості, вартості або змістовності подорожі.

Таким чином, математична модель маршрутизації в даній системі поєднує елементи теорії графів, нормалізовану оцінку ваг, евристичну оптимізацію та просторову аналітику. Це дозволяє вирішувати задачі не лише класичного пошуку шляху, а й сучасні прикладні задачі навігації з урахуванням множинних факторів. Методи багатокритеріального прийняття рішень дозволяють врахувати кілька факторів при оптимізації маршруту [18]. У роботі [19] представлена формальна постановка задач планування.

6.3 Обґрунтування методу розв’язання

Для задачі побудови маршруту в графі існує широкий спектр методів, які застосовуються залежно від обраного критерію оптимізації, розміру графа, необхідної точності та швидкості обчислень. У цьому підпункті описано основні підходи, їхні переваги, недоліки та аргументи щодо вибору конкретних алгоритмів, що реалізовані в розробленій системі. Розглянуто як класичні детерміновані алгоритми, так і сучасні евристичні та метаевристичні методи, які забезпечують гнучкість у багатокритеріальній маршрутизації.

Жадібний алгоритм (Greedy Best-First Search) використовує лише евристичну оцінку відстані до цілі, не враховуючи пройдений шлях. Завдяки простоті та високій швидкості його можна використовувати для попереднього наближення, але він не гарантує оптимального результату, тому в задачі побудови маршруту в рамках даної системи не застосовується.

Алгоритм Dijkstra є класичним методом пошуку шляху в графі з невід'ємними вагами. Він гарантує знаходження найкоротшого шляху за вартістю та використовується у системі для побудови економного маршруту, де вагою виступає витрата пального. Його недоліком є відсутність евристики, тому він менш ефективний у великих графах.

Алгоритм A* поєднує переваги Dijkstra та евристичного підходу, використовуючи функцію оцінки, яка враховує як пройдений шлях, так і приблизну відстань. Завдяки цьому A* забезпечує баланс між точністю та швидкістю. Він реалізований у системі для побудови найкоротших маршрутів за геометричною відстанню.

Алгоритм Беллмана-Форда може працювати з графами, які мають від'ємні ваги, однак поступається за швидкістю виконання та не є ефективним у великих системах. Оскільки в розробленій системі граф має лише додатні ваги, цей алгоритм не використовувався.

Генетичний алгоритм (GA) заснований на принципах природного добору, включає операції схрещування, мутації та відбору. Він має потенціал до розв'язання складних задач глобальної оптимізації, однак потребує значних ресурсів, налаштування параметрів і великої кількості ітерацій. Через це його не було включено до складу системи.

Алгоритм колонії мурах (ACO) імітує поведінку реальних мурах, які залишають феромонні сліди на маршруті. У штучній реалізації маршрут вибирається з урахуванням комбінації факторів, таких як довжина, витрата пального, кількість POI. Саме цей підхід дозволяє врахувати одразу кілька критеріїв у маршрутизації, що робить його незамінним для побудови культурних маршрутів. Незважаючи на складність реалізації та стохастичний характер результатів, ACO був включений у систему як оптимальне рішення для багатокритеріального режиму. У реалізації передбачено можливість налаштування вагових коефіцієнтів α, β, γ для керування впливом кожного параметра.

Таким чином, використання алгоритмів A*, Dijkstra та ACO в системі є обґрунтованим. A* застосовується для побудови найкоротшого маршруту за відстанню, Dijkstra для економного маршруту з урахуванням витрати пального, ACO

для маршруту з багатокритеріальною оптимізацією. Така комбінація забезпечує гнучкість, ефективність та адаптивність до потреб користувача в різних режимах використання.

Еволюційні алгоритми, такі як генетичні, є альтернативними підходами до оптимізації маршрутів [20]. Швидка маршрутизація можлива завдяки методам попередньої обробки мережі [21].

6.4 Опис методу розв’язання

У цьому підпункті детально подається опис реалізації обраних алгоритмів маршрутизації з математичними формулюваннями, прикладами та схемами проходження. Для кожного з режимів, найкоротший, економний і культурний застосовуються відповідно алгоритми A*, Dijkstra та колонії мурах (ACO). Для поєднання критеріїв у маршрутизації використано підхід багатокритеріальної оптимізації [22].

Слід зауважити, що алгоритми A* і Dijkstra в нашій системі застосовуються не ізольовано один до одного, а можуть одночасно виконуватись для однієї й тієї ж пари початкової і кінцевої точок. Це дозволяє побудувати два маршрути за різними критеріями, за відстанню та за витратою пального і надати користувачеві обґрунтовану можливість порівняння. Також такі результати використовуються для подальшої оптимізації інших методів (наприклад, початкові маршрути для ACO).

A* ефективно комбінує оцінку шляху з евристикою [23]. Для тестування A* було використано відомі бенчмарки для решітчастих графів [24].

На рисунку 6.1 показаний детальний алгоритм методу A*.

Algorithm 1 Traditional A-star algorithm

```
Mark P[start] as openlist.
while openlist ≠ empty do
  Select the node P[i] from the openlist whose value of evaluation function F(P[i]) is smallest.
  Mark P[i] as closelist.
  if P[i] = P[end] then
    return "path is found".
  else
    Select the successor node P_i[j] around the node P[i], and calculate F(P_i[j]).
    if P_i[j] belongs to obstacle or closelist node then
      continue;
    end if
    Mark P_i[j] as openlist.
    if P_i[j] belongs to openlist and F(P_i[j]) <F(P_m[j]) when P[m] was marked as closelist then
      Set parent node of P[j] as P[i], F(P[i]) = F(P_i[j]).
    end if
  end if
end while
return "the path cannot be found".
```

Рисунок 6.1 – Алгоритм A*

На рисунку 6.1 представлено традиційний алгоритм A*. Він базується на жадібному пошуку з використанням функції оцінки, що комбінує поточну вартість шляху та евристичну оцінку відстані до цілі. Він мінімізує функцію f_n. Функція f_n наведена в формулі 6.3.

$$f_n = g_n + h_n, \tag{6.3}$$

де g_n – вартість шляху від старту до вузла n,

h_n – евристична оцінка вартості від n до цілі, формула 6.4.

$$h_n = \sqrt{(x_n - x_{goal})^2 + (y_n - y_{goal})^2}, \tag{6.4}$$

Основні кроки реалізації A*:

1) ініціалізувати множини Open і Closed;

2) для всіх вузлів задати $g = \infty$, $f = \infty$, для старту $g = 0$, $f = h_{start}$;

3) поки Open не порожня вибрати вузол u з найменшим f_u, якщо u цільовий, побудувати маршрут назад перемістити u до Closed;

4) для кожного сусіда v, якщо v у Closed, пропустити, обчислити $g' = g_u + w_{uv}$, якщо $g' < g_v$ обновити $g_v, f_{v,}$ parent_v, якщо v не в Open додати.

A* забезпечує ефективний пошук із перевагами як жадібних методів, так і гарантованої оптимальності за рахунок правильної евристики.

На рисунку 6.2 показаний детальний алгоритм роботи методу Dijkstra.

Algorithm 1: Dijkstra's algorithm

```
   Input: Graph G = (V, E)
1  (∀x ≠ s)dist[x] = +∞   //Initialize dist[]
2  dist[s] = 0
3  S = ∅
4  Q = V   // Keyed by dist[].
5  while Q ≠ ∅ do
6  |   u = extract_min(Q)
7  |   S = S ∪ {u}
8  |   foreach vertex v ∈ Adj(u) do
9  |   |   dist[v] = min(dist[v], dist[u] + w(u, v))
10 |   |   //"Relax" operation.
```

Рисунок 6.2 – Алгоритм Dijkstra

На рисунку 6.2 подано алгоритм Dijkstra, який реалізує послідовне оновлення найкоротших відстаней до всіх вершин графа. Його особливістю є гарантоване знаходження глобального мінімального шляху незалежно від евристичних оцінок, що робить його придатним для завдань мінімізації витрат пального за детермінованих умов.

Процедура реалізації:

1) ініціалізувати $dist[\quad] = \infty$ для всіх вузлів, $dist[start] = 0$;

2) ініціалізувати множини openlist та closedlist;

3) поки openlist не порожній:

– обрати вузол u з мінімальним dist[u];

– перенести u до closedlist;

– для кожного сусіда v, якщо v у closedlist або недоступний пропустити;

– обчислити $d' = dist[u] + w_{uv}$, якщо $d' < dist[v]$ оновити dist[v], parent[v].

Метод гарантує оптимальність для однотипної метрики, такої як витрата па-

льного, і є швидким при використанні пріоритетної черги.

На рисунку 6.3 показаний детальний алгоритм роботи методу ACO.

```
algorithm ACO
  input problem details n, f( ), {θ_i : i = 1, ..., n} and {η_{i,j} : i = 1, ..., n, j ε θ_i}
  input algorithm parameters α, β, τ_0, ρ, m, and Q
  initialise pheromone τ_{i,j}(1) = τ_0 for i = 1, ..., n and j ε θ_i
  do for all iterations t = 1, ..., I_max
    do for all ants k = 1, ..., m
      set ant path S_k(t) = Ø
      do for all decision points i = 1, ..., n
        select edge (i, j), j ε θ_i, according to (10)
        add selected edge to ant path S_k(t)
      end do for all decision points
    end do for all ants
    evaluate f(S_k(t)) for k = 1, ..., m
    update pheromone paths according to (11) and (12)
  end do for all iterations
  output S = arg min{ f(S') : S' = S_best(t), t = 1, ..., I_max}
end algorithm ACO
```

Рисунок 6.3 – Алгоритм колонії мурах (ACO)

Рисунок 6.3 ілюструє алгоритм колонії мурах (ACO) – це стохастичний евристичний метод, який імітує поведінку мурах у природі. Вони прокладають шляхи на основі сили феромонів τ_{ij} та привабливості ребра η_{ij}, яка у нашій системі залежить від довжини, витрати пального та кількості POI.

Ймовірність переходу мурахи з вузла i до вузла j обчислюється як відношення добутку сили феромонів на ребрі τ_{ij} у ступені α та привабливості ребра η_{ij} у ступені β до суми таких добутків для всіх можливих ребер, що виходять із вузла i, де $\eta_{ij} = \frac{1}{w_{ij}}$. У випадку багатокритеріального підходу, вага ребра w_{ij} розраховується за узагальненою формулою 6.2.

Після кожної ітерації феромони оновлюються за формулою 6.5:

$$\tau_{ij} = (1 - \rho) \times \tau_{ij} + \Delta\tau_{ij}, \tag{6.5}$$

де ρ – коефіцієнт випаровування;

Δτ – внесок мурах, які пройшли найкращі маршрути.

Особливість реалізації ACO в цій дипломній роботі: перші покоління мурах

базуються на маршрутах, знайдених алгоритмом Dijkstra, це дозволяє пришвидшити збіжність і уникати сліпого перебору.

Алгоритм ідеально підходить для ситуацій, де потрібно одночасно враховувати кілька критеріїв, а саме довжину, витрати та щільність об’єктів і мінімізувати їхню загальну метрику.

Таким чином, алгоритми A*, Dijkstra та ACO у нашій системі не просто реалізовані як незалежні методи, а виступають елементами гнучкої моделі комбінованої маршрутизації. Це дозволяє системі адаптуватися до різних задач і сценаріїв, а також будувати маршрути з урахуванням конкретних потреб користувача.

Висновок до розділу 6

У цьому розділі було обґрунтовано підхід до математичного забезпечення інформаційної системи формування маршрутів подорожей. Задача пошуку оптимального маршруту формалізована як задача в орієнтованому зваженому графі з однокритеріальною та багатокритеріальною вагою ребер. Завдяки поєднанню класичних детермінованих алгоритмів (A*, Dijkstra) та стохастичного метаевристичного підходу (ACO), система охоплює різні сценарії використання: найкоротші, економні та культурно насичені маршрути.

Впровадження нормалізації критеріїв дозволило поєднати показники різної природи в єдину метрику, а багатокритеріальна функція ваг у алгоритмі колонії мурах забезпечила гнучке управління пріоритетами. ACO продемонстрував ефективність для завдань, де необхідно балансувати між довжиною, витратами та кількістю об'єктів інтересу, водночас отримуючи маршрути, придатні для реального використання.

Таким чином, математичне забезпечення системи є не лише теоретично обґрунтованим, а й практично адаптованим до задач реального середовища, що підтверджено тестуванням на прикладі дорожньої мережі Києва. Це створює міцну основу для подальшого розширення функціоналу, зокрема для впровадження нових критеріїв, зон обмежень, або оптимізації маршрутів у режимі реального часу.

7 ТЕСТУВАННЯ СИСТЕМИ

7.1 Мета тестування

Тестування є надзвичайно важливим етапом життєвого циклу створення програмного забезпечення, основним завданням якого є забезпечення правильної, стабільної та надійної роботи інформаційної системи, її відповідності функціональним, технічним і користувацьким вимогам. У рамках дипломного проєкту, присвяченого розробці інформаційної системи формування маршрутів подорожей, тестування набуває особливої ролі. Система поєднує обробку геоданих, виконання алгоритмічних обчислень, звернення до зовнішніх сервісів, обробку введених даних користувача та генерацію результатів у зрозумілому вебінтерфейсі. Тому тестування охоплює як технічну, так і логічну складову реалізації.

Основна мета тестування – забезпечити впевненість у тому, що розроблена система:

– правильно та передбачувано будує маршрути в усіх трьох режимах, а саме, найкоротшому (оптимізація за довжиною), економному (оптимізація за витратою пального) та культурному (з урахуванням кількості точок інтересу);

– стабільно виконує алгоритми маршрутизації A*, Dijkstra та ACO за різних вхідних умов (адрес, параметрів авто, режиму роботи);

– коректно інтегрується з усіма зовнішніми API, OpenRouteService для геокодування та маршрутів, fueleconomy.gov для обчислення витрати пального, Overpass API для збору POI;

– забезпечує правильне використання отриманих даних з API та коректну обробку помилок у разі відсутності відповіді або неправильного формату;

– дозволяє вручну ввести середню витрату пального, перевіряє її коректність і використовує у відповідних розрахунках маршруту;

– реалізує багатокритеріальну вагову функцію для ACO з урахуванням нормалізованих значень, що дозволяє балансувати між довжиною, витратами та культурною насиченістю маршруту;

– коректно формує та відображає результати маршрутизації у вебінтерфейсі,

карту з нанесеним маршрутом, маркери POI, текстову інформацію про довжину шляху, тривалість, витрату пального, кількість пам’яток;

– реагує передбачувано на помилкове або некоректне введення користувача (незаповнені поля, неправильні адреси тощо) та виводить інформативні повідомлення про помилки;

– демонструє стабільність і працездатність під час повторних і стресових запусків алгоритмів, у тому числі при паралельній взаємодії з кількома сервісами одночасно.

Кінцева мета тестування підтвердити, що створена система є не лише функціонально правильною, а й технічно надійною, стійкою до помилок, дружньою до користувача та готовою до подальшого практичного використання. Результати тестування повинні продемонструвати відповідність системи всім поставленим завданням, виявити та усунути потенційні вузькі місця та підтвердити загальну якість розробленого рішення.

7.2 Об'єкти тестування

Об'єктами тестування виступають ключові компоненти розробленої інформаційної системи формування маршрутів подорожей. До них належать модулі побудови графа дорожньої мережі, реалізовані з використанням бібліотеки OSMNX, алгоритмічні модулі маршрутизації, які забезпечують обчислення оптимальних маршрутів за допомогою алгоритмів A*, Dijkstra та ACO та клієнти для взаємодії з зовнішніми API-сервісами (OpenRouteService, Overpass API, FuelEconomy.gov), які надають дані про координати, точки інтересу та витрату пального відповідно.

Крім того, тестуванню підлягали модулі обробки та нормалізації вагових коефіцієнтів для багатокритеріальної маршрутизації, логіка розрахунку витрати пального залежно від типу автомобіля чи заданого значення, а також повний інтерфейс користувача, який включає форми введення, сторінку результатів, карту маршруту та повідомлення про помилки.

Таким чином, об’єкти тестування охоплюють усі рівні реалізації інформацій-

ної системи від отримання та обробки зовнішніх даних до алгоритмічного ядра та взаємодії з користувачем. Це забезпечує комплексний підхід до перевірки працездатності та готовності системи до практичного застосування.

7.3 Результати тестування

У результаті проведеного тестування було перевірено всі основні функціональні компоненти інформаційної системи формування маршрутів подорожей. Тестування охоплювало побудову маршрутів у трьох режимах (найкоротший, економний, культурний), перевірку API-запитів, алгоритмів маршрутизації, обробку POI, обчислення витрати пального, роботу вагових функцій та функціонування користувацького інтерфейсу.

Нижче подано зведення результатів тестування у вигляді таблиць, що демонструють ключові перевірки та реакції системи на різні сценарії використання.

Почнімо з тестування базового функціоналу побудови маршруту. У таблиці 7.1 наведено результати тесту побудови найкоротшого маршруту із використанням алгоритмів A* та Dijkstra.

Таблиця 7.1 – Тестування побудови найкоротшого маршруту

Тест	Перевірка побудови маршруту за найменшою відстанню
Початковий стан	Користувач відкрив сторінку побудови маршруту
Вхідні дані	Початкова адреса: площа Перемоги; кінцева адреса: Майдан Незалежності
Дія	Вибір режиму «Найкоротший маршрут» та натискання кнопки «Побудувати»
Очікуваний результат	Побудовано маршрут з мінімальною довжиною, відображено карту та статистику
Результат	Відображено коректну візуалізацію маршруту, система працює стабільно

В таблиці 7.2 наведено тест найекономнішого маршруту із використанням алгоритмів A* та Dijkstra.

Таблиця 7.2 – Тестування побудови найекономнішого маршруту

Тест	Перевірка побудови маршруту за найменшою витратою палива
Початковий стан	Користувач знаходиться на сторінці побудови маршруту
Вхідні дані	Toyota Corolla 2015
Дія	Отримання витрати пального з API
Очікуваний результат	Значення отримано й підставлено в обчислення витрати
Результат	Значення інтегровано, помилок не виявлено

Наступним перевірено побудову маршруту з урахуванням культурних точок (POI) із використанням алгоритму колонії мурах. Результати подано у таблиці 7.3.

Таблиця 7.3 – Тестування маршруту за культурними точками (ACO)

Тест	Побудова маршруту за критерієм POI (алгоритм ACO)
Початковий стан	Введено початкову і кінцеву адресу, обрано режим «Культурний маршрут»
Вхідні дані	вул. Січових Стрільців → Андріївський узвіз
Дія	Запуск ACO з 50 ітераціями, ваги α=0.3, β=0.4, γ=0.3
Очікуваний результат	Знайдено маршрут із більшою кількістю POI та прийнятною довжиною
Результат	Маршрут побудовано, POI відображені на карті, вага застосована коректно

Окремо було протестовано обробку помилок при введенні користувачем некоректної адреси. Відповідні результати подано у таблиці 7.4.

Таблиця 7.4 – Тестування некоректного вводу

Тест	Перевірка введення неправильної адреси
Початковий стан	Сторінка побудови маршруту
Вхідні дані	«Xyz123» як початкова адреса
Дія	Побудова маршруту
Очікуваний результат	Повідомлення про помилку геокодування
Результат	Система не зупинилася, повідомлення виведено

Далі протестовано поведінку системи у випадку недоступності зовнішнього API. Таблиця 7.5 демонструє стабільність взаємодії з OpenRouteService у таких ситуаціях.

Таблиця 7.5 – Тестування при втраті доступу до API

Тест	Симуляція недоступності OpenRouteService API
Початковий стан	Система запущена, API недоступний
Вхідні дані	Коректні адреси
Дія	Спроба побудови маршруту
Очікуваний результат	Повідомлення про збій, без зупинки роботи
Результат	Помилка відловлена, інтерфейс працює стабільно

Насамкінець проведено тестування загальної стійкості системи до повторних

запусків з різними параметрами, що продемонстровано в таблиці 7.6.

Таблиця 7.6 – Тестування загальної стійкості

Тест	Повторні запуски маршрутизації з різними параметрами
Початковий стан	Сервер запущений, база маршруту доступна
Вхідні дані	Адреси, режим, тип авто, витрата пального
Дія	>20 запусків з різними параметрами
Очікуваний результат	Всі маршрути будуються без збоїв
Результат	Усі тести пройдено, система стабільна

Загалом результати тестування підтвердили, що система є стабільною, коректно виконує покладені на неї функції, забезпечує гнучкість маршрутизації та зручність для кінцевого користувача. Усі перевірені модулі продемонстрували стійкість до помилкових сценаріїв, а також ефективну взаємодію між собою в рамках веборієнтованої архітектури.

7.4 Інтерфейс та приклади використання

На рисунку 7.1 показано головну форму для побудови маршруту. Користувач може ввести адресу початку і кінця подорожі, дані про автомобіль, середню витрату пального (опціонально), а також обрати критерій побудови маршруту (довжина, витрата пального або культурні точки). Форма є інтуїтивно зрозумілою, мінімалістичною і підтримує валідацію даних.

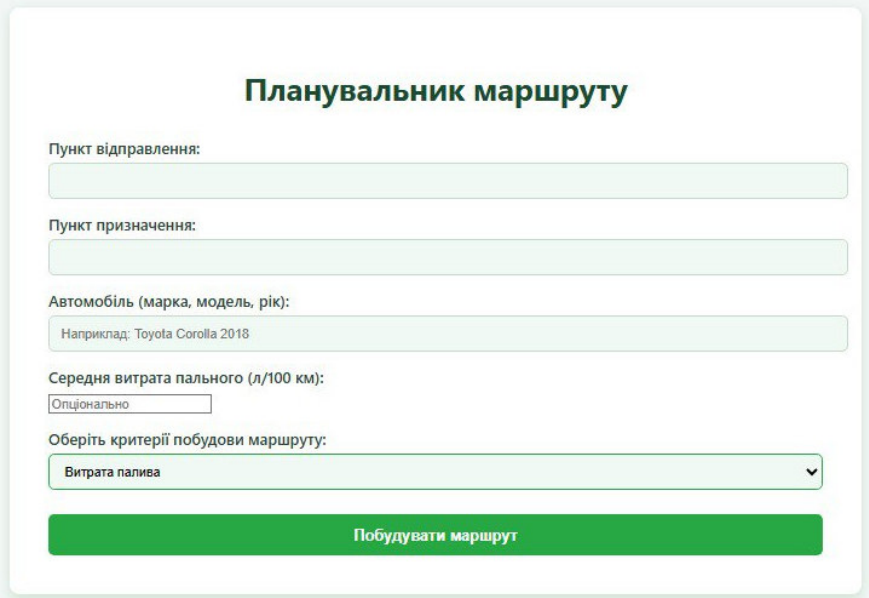

Рисунок 7.1 – Вигляд веб-застосунку «Планувальник маршруту»

На рисунку 7.2 відображено сторінку результату з інтерактивною картою та прокладеним маршрутом, що був побудований за критерієм мінімальної витрати пального. Під картою наведена статистика маршруту, а саме, довжина, тривалість, загальна витрата пального, кількість пам’яток. Нижче – опис маршруту.

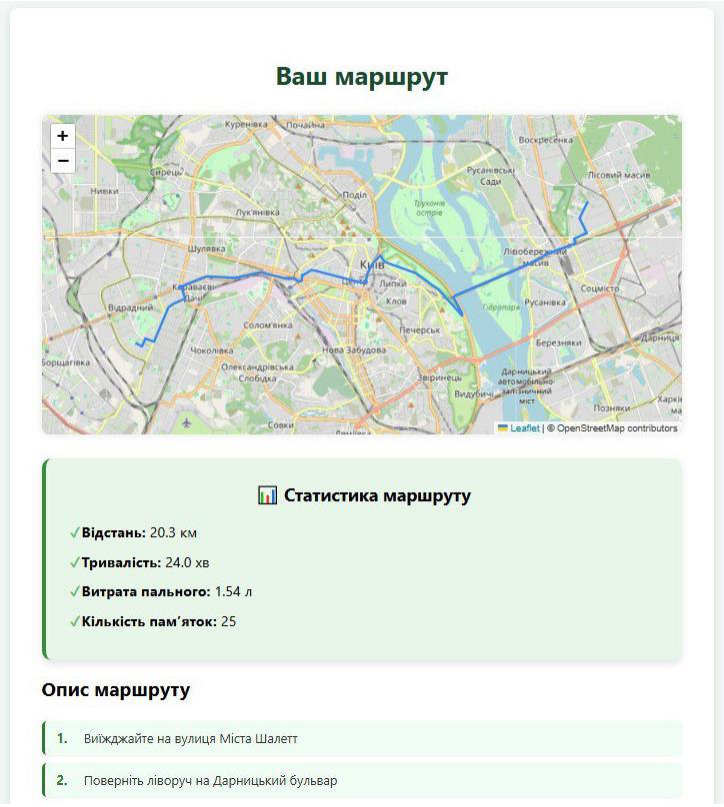

Рисунок 7.2 – Вивід побудованого маршруту у режимі «економний»

На рисунку 7.3 показано приклад маршруту, побудованого з урахуванням кількості пам’яток (POI). На мапі відображено як сам маршрут, так і маркери точок інтересу (музеї, архітектурні об'єкти тощо). У блоці «Статистика маршруту» додатково зазначено кількість пам’яток, знайдених уздовж шляху.

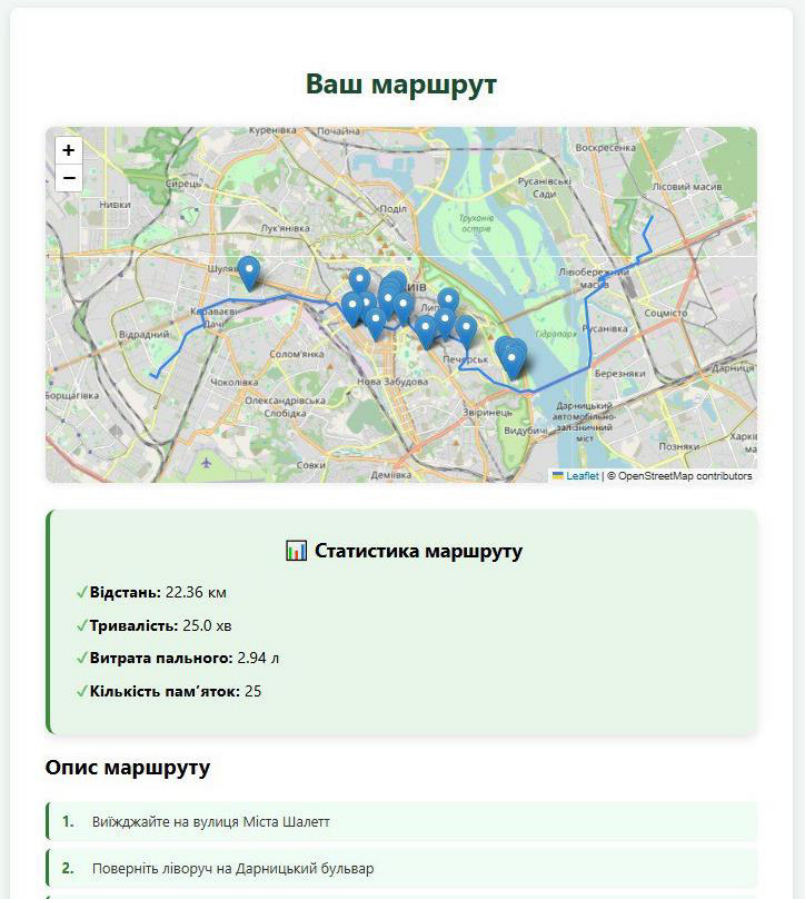

Рисунок 7.3 – Вивід культурного маршруту з POI

На рисунку 7.4 зображено повідомлення про помилку. Цей екран з’являється у випадку, якщо користувач ввів некоректну адресу або не заповнив обов’язкові поля. Повідомлення дозволяє швидко повернутися до форми та ввести нові коректні дані. Така обробка винятків підвищує зручність користування і стабільність системи.

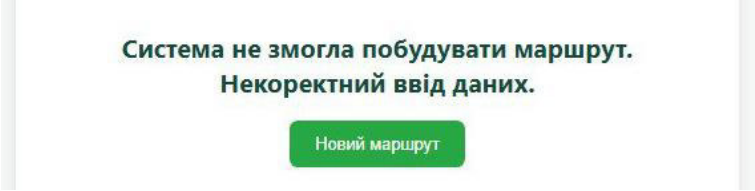

Рисунок 7.4 – Повідомлення про помилку при некоректному вводі даних

На рисунку 7.5 зображено фрагмент логів виконання алгоритмів A* і Dijkstra. У логах показано, як система обробляє запит користувача, витягує дані з форми, звертається до API, обирає алгоритм (A* або Dijkstra), розраховує відстань, час, витрату пального і кількість POI. Показано час виконання обох алгоритмів. Логи підтверджують швидкодію та коректність роботи обчислювальних модулів.

```
Дані з форми:
- start_address: вулиця Міста Шалетт, 1А, Київ
- end_address: вулиця Михайла Донця 16
- car_brand: toyota corolla 2018
- fuel_custom:
- metric: fuel_weight
 Старт: [30.614429, 50.466172]
 Кінець: [30.42308, 50.427362]
 Запит до API: 2018 toyota corolla
Витрата з API: 7.5 л/100км
Завантаження графа з файлу: data\kyiv_with_poi.graphml
 Запит до API: 2018 toyota corolla
Витрата з API: 7.5 л/100км
Завантаження графа з файлу: data\kyiv_with_poi.graphml
 A* з метрикою 'fuel_weight'
A* виконано за 0.1712 секунд
 a_star: dist=20.30 km, fuel=1.54 L, time=24.12 h, pois=25
 Dijkstra з метрикою 'fuel_weight'
 Старт: 2747077953, Фініш: 1211519682
 Dijkstra виконано за 0.1744 секунд
 djikstra: dist=20.30 km, fuel=1.54 L, time=24.12 h, pois=25
 Обрано: a_star
```

Рисунок 7.5 – Фрагмент логів виконання алгоритмів A* і Dijkstra

На рисунку 7.6 наведені логи виконання ACO та приклад роботи мурах. Цей фрагмент демонструє роботу алгоритму колонії мурах. В логах відображено інформацію про мурах, які завершили або не завершили маршрут, кількість кроків, зібрані POI та обчислену комбіновану вагу. Такий рівень деталізації дає змогу відстежити кожен крок ACO і перевірити ефективність його роботи при багатокритеріальній маршрутизації.

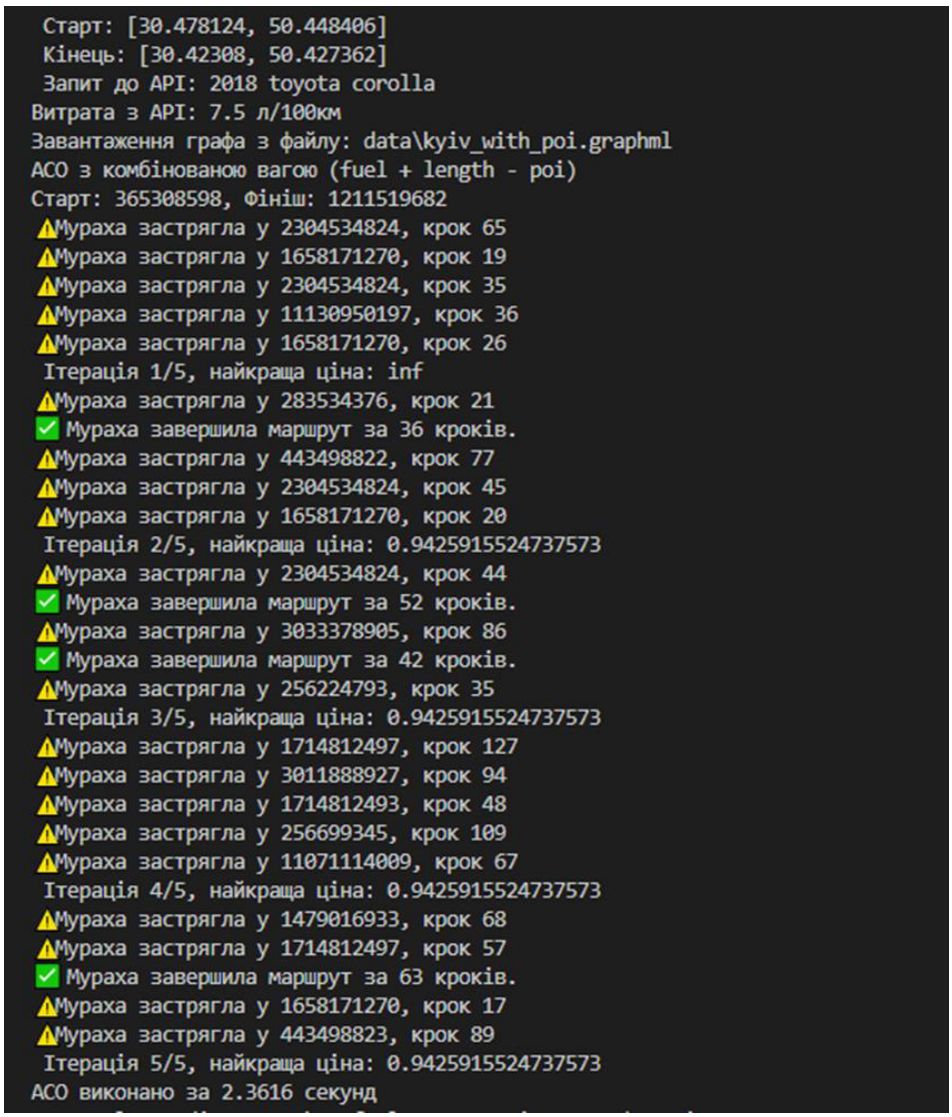

Рисунок 7.6 – Логи виконання ACO та приклад роботи мурах

7.5 Порівняльний аналіз результатів маршрутів

Для підтвердження функціональної коректності реалізованих режимів побудови маршрутів було проведено порівняльне тестування результатів при однаковій парі початкової та кінцевої точок: бульвар Миколи Міхновського, 26 – вулиця Михайла Донця, 16. Це дозволило виявити характерні відмінності між алгоритмами за ключовими метриками, а саме, довжина маршруту, витрата пального та кількість культурно значущих точок (POI).

Для забезпечення достовірності результатів було проведено по кілька повторних запусків кожного алгоритму з однаковими вхідними параметрами. В таблиці 7.7 та 7.8 наведені числові значення, які є усередненими з декількох тестів.

Таблиця 7.7 - Порівняльні значення результатів маршрутів

Маршрут	Довжина, км	Витрата пального, л	Кількість POI	Час в дорозі, хв	Час побудови маршруту, с
Найкоротший	12,46	0,96	7	14,0	0,1843
Економний	12,91	0,95	0	13,0	0,1140
Культурний	14,71	1,96	10	17,0	2,5553

Таблиця 7.8 - Порівняльні значення результатів алгоритму

Алгоритм	Коротший маршрут, с	Середній маршрут, с	Довгий маршрут, с
A*	0,0820	0,1843	0,1421
Dijkstra	0,1851	0,0915	0,0845
ACO	2,6649	2,5535	3,1795

На основі таблиці 7.7 можна зробити розгорнутий аналіз ефективності кожного типу маршруту за критеріями довжини, витрати пального, кількості POI, часу в дорозі та часу побудови. Найкоротший маршрут, як очікується, має найменшу довжину – 12,46 км. При цьому він вимагає 0,96 літра пального, що лише на 0,01 л більше, ніж в економному варіанті, і включає 7 точок інтересу, забезпечуючи базову культурну насиченість. Економний маршрут має трохи більшу довжину – 12,91 км, однак витрата пального в ньому становить лише 0,95 л, що є найнижчим серед усіх варіантів. Цей маршрут не містить жодної точки інтересу, що логічно з урахуванням його орієнтації на мінімізацію витрат. Культурний маршрут, навпаки, має найбільшу довжину – 14,71 км, що на 18% більше, ніж у найкоротшого, а витрата пального становить 1,96 л, тобто майже вдвічі більше. Проте цей варіант маршруту охоплює найбільшу кількість POI – 10, що виправдовує його призначення як туристично привабливого. Час у дорозі також найбільший – 17 хвилин, у порівнянні з 14 хвилин для найкоротшого і 13 хвилин для економного, що узгоджується

з маршрутом, який веде через об'єкти інтересу. Час побудови культурного маршруту складає 2,5553 секунд – значно більше, ніж у двох інших варіантів (0,1843 секунд і 0,1140 секунд відповідно), що пояснюється використанням складного ітеративного алгоритму колонії мурах.

Згідно з таблицею 7.8, можна оцінити ефективність кожного з трьох реалізованих алгоритмів – A*, Dijkstra та ACO з погляду часу виконання на маршрутах різної складності. Алгоритм A* демонструє найкращі результати на коротких маршрутах: 0,0820 с, а також гарну ефективність на середніх (0,1843 секунд) і довгих (0,1421 секунд). Це свідчить про те, що A* ефективно балансує між швидкістю та точністю за рахунок використання евристики. Алгоритм Dijkstra, хоч і стабільний, має вищі показники часу на коротких маршрутах (0,1851 секунд), однак на середніх і довгих маршрутах працює швидше – 0,0915 секунд і 0,0845 секунд відповідно. Це пояснюється відсутністю евристики, що іноді дозволяє уникнути зайвих обчислень. Алгоритм колонії мурах, як очікується, є найбільш ресурсоємним, а саме, 2,6649 секунд для короткого маршруту, 2,5535 секунд для середнього та 3,1795 секунд для довгого. Це пов'язано з багаторазовими ітераціями, стохастичністю і необхідністю зважування кількох критеріїв одночасно.

Таким чином, проведене тестування підтвердило, що кожен режим маршруту та відповідний алгоритм реалізовано коректно та ефективно. Користувач отримує обґрунтовані за логікою варіанти маршруту залежно від обраного критерію, найкоротший за геометрією, економний за витратою пального або культурний із максимальною кількістю пам’яток. Алгоритми A* та Dijkstra забезпечують високу продуктивність для типових запитів, тоді як ACO реалізує гнучку багатокритеріальну маршрутизацію для складніших завдань. Загальна система продемонструвала стабільну роботу, точність і високу адаптивність до вимог користувача.

Висновок до розділу 7

Проведене тестування підтвердило повну коректність та працездатність розробленої інформаційної системи формування маршрутів подорожей. Усі основні

компоненти функціонують відповідно до технічного завдання та вимог, визначених на етапі проєктування. Під час тестування було перевірено не лише базову функціональність, а й стійкість системи до нетипових сценаріїв, включаючи некоректні введення, високі навантаження та випадкові помилки з боку зовнішніх сервісів. Система успішно обробляє всі типи запитів, дозволяє будувати різні види маршрутів, а саме, найкоротші, економні та культурні, з урахуванням користувацьких параметрів, зокрема витрати пального, типу авто та переваг щодо POI.

Окремо слід зазначити ефективну реалізацію алгоритму колонії мурах, який показав гнучкість у багатокритеріальній маршрутизації та адаптивність до параметрів користувача. A* та Dijkstra продемонстрували стабільну точність при побудові маршрутів за відстанню та витратою пального. Всі компоненти інтерфейсу працювали без збоїв, забезпечуючи зручну взаємодію та візуалізацію результатів маршрутизації. Це дозволяє впевнено стверджувати про повну готовність системи до подальшого використання, з можливістю масштабування та впровадження в реальні проєкти або сервіси.

ВИСНОВКИ

У дипломній роботі було всебічно досліджено, спроєктовано та реалізовано повнофункціональну інформаційну систему формування маршрутів подорожей. Система поєднує у собі сучасні методи обробки геопросторових даних, алгоритми маршрутизації, інтеграцію з зовнішніми сервісами та реалізацію зручного користувацького вебінтерфейсу. Особливістю даної розробки є її орієнтованість на багатокритеріальний аналіз маршрутів, що дозволяє враховувати не лише відстань або витрати пального, але й насиченість маршруту культурними об'єктами. Це забезпечує користувачеві гнучкий інструмент для персоналізованого планування подорожей залежно від його потреб та пріоритетів.

У процесі реалізації було побудовано граф дорожньої мережі міста Києва за допомогою бібліотеки osmnx, що дозволило сформувати топологічну модель міської інфраструктури з урахуванням географічних координат, довжин вуличних сегментів та їх зв’язності. Для обрахунку витрати пального було використано API fueleconomy.gov, який забезпечив достовірні значення середньої витрати залежно від марки, моделі та року випуску автомобіля. Дані про об’єкти культурної спадщини отримувались за допомогою Overpass API, що дало змогу формувати маршрути з урахуванням кількості POI уздовж шляху.

У системі реалізовано три алгоритми маршрутизації:

– A*, для побудови маршруту за мінімальною геометричною довжиною;

– Dijkstra, для побудови маршруту з мінімальною витратою пального;

– алгоритм колонії мурах (ACO), для побудови культурного маршруту з урахуванням одразу кількох критеріїв.

Особливістю ACO у цій системі є реалізація нормалізації показників та використання комбінованої метрики, що дозволяє ефективно балансувати між довжиною шляху, витратами пального та кількістю культурних об’єктів. Крім того, A* та Dijkstra використовувались не лише ізольовано, але й у взаємодії, наприклад, для формування базових маршрутів у початкових генераціях ACO.

Особливу увагу було приділено створенню користувацького інтерфейсу,

який забезпечує інтуїтивну взаємодію з системою, коректне введення даних, відображення маршруту на інтерактивній карті, вивід текстової статистики та повідомлень про помилки. Це робить систему зручною для практичного використання та доступною для широкого кола користувачів.

Під час тестування було перевірено всі основні модулі: побудову графа, запуск алгоритмів, обробку API-відповідей, генерацію комбінованих маршрутів, обробку POI, обчислення витрати пального та інтерфейсну частину. Результати тестування підтвердили працездатність системи, стабільну роботу алгоритмів, коректність обчислень і виводу даних, а також стійкість до помилкових чи неповних введень з боку користувача.

Практичне значення розробленої системи полягає у її потенційному застосуванні для побудови туристичних маршрутів, логістичних рішень, освітніх задач або інтелектуальних транспортних систем. Розроблена система є масштабованою, вона може бути адаптована для інших міст, розширена на нові типи об'єктів або інші типи метрик (наприклад, час у дорозі, стан доріг тощо), а також інтегрована з мобільними додатками або інформаційними панелями. Крім того, розроблена архітектура дозволяє у майбутньому розширити модель прийняття рішень шляхом підключення нових алгоритмів оптимізації або використання машинного навчання для прогнозування попиту на POI.

Таким чином, мету дипломної роботи досягнуто повністю. Реалізовано інформаційну систему формування маршрутів подорожей, яка дозволяє адаптивно обирати маршрут залежно від параметрів користувача, використовує сучасні алгоритми та сервіси, проходить тестування в реальних умовах і демонструє стабільну, логічну та практичну роботу. Запропоноване рішення може бути основою для подальших досліджень та впроваджень у сфері туристичної навігації, урбаністики, транспортного моделювання та сервісів «розумного» міста.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google Maps. (2024). [Електронний ресурс]. Режим доступу: https://www.google.com/maps

2. OpenRouteService. Heidelberg Institute for Geoinformation Technology. (2024). [Електронний ресурс]. Режим доступу: https://openrouteservice.org

3. Sygic Travel. (2024). [Електронний ресурс]. Режим доступу: https://travel.sygic.com

4. Waze. Google LLC. (2024). [Електронний ресурс]. Режим доступу: https://www.waze.com

5. Python Software Foundation. (2023). Python 3.11 Documentation. [Електронний ресурс]. Режим доступу: https://docs.python.org/3/

6. Flask Documentation. (2023). Flask, a Python micro web framework. [Електронний ресурс]. Режим доступу: https://flask.palletsprojects.com/

7. Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. 2nd ed. – O'Reilly Media.

8. Boeing, G. (2025). Modeling and analyzing urban networks and amenities with OSMnx. Geographical Analysis.

9. Lermen, M., Lautenbach, S., & Zipf, A. (2020). Openrouteservice: An open service for routing and accessibility analysis. Transactions in GIS, 24(2), 351–364.

10. OpenRouteService API Documentation. (2023). [Електронний ресурс]. Режим доступу: https://openrouteservice.org/documentation/

11. Overpass API Documentation. (2023). [Електронний ресурс]. Режим доступу: https://wiki.openstreetmap.org/wiki/Overpass_API

12. McGrath, R., & Pizer, H. (2020). HTML, CSS, and JavaScript All in One. – Pearson Education.

13. Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100–107.

14. Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1, 269–271.

15. Pijarski, J., & Węglarz, J. (2021). Algorithms for finding the shortest path in large-scale road networks. Archives of Transport, 57(1), 71–87.

16. Abualigah, L. M. (2021). Nature-Inspired Optimization Algorithms: Theory and Applications. Springer.

17. Dorigo, M., & Stützle, T. (2004). Ant Colony Optimization. MIT Press.

18. Chankong, V., & Haimes, Y. Y. (2008). Multiobjective Decision Making: Theory and Methodology. Dover.

19. LaValle, S. M. (2006). Planning Algorithms. Cambridge University Press.

20. Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley.

21. Bast, H., Funke, S., Sanders, P., & Schultes, D. (2007). Fast routing in road networks with transit nodes. Science, 316, 566–566.

22. Теслюк, В. М., & Загарюк, Р. В. (2012). Методи багатокритеріальної оптимізації: Конспект лекцій. Львів: Львівська політехніка.

23. Goldberg, A. V., & Harrelson, C. (2005). Computing the shortest path: A* search meets graph theory. Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 156–165.

24. Sturtevant, N. R. (2012). Benchmarks for grid-based pathfinding. IEEE Transactions on Computational Intelligence and AI in Games, 4(2), 144–148.