Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматики
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

WEB-ПЛАТФОРМА ДЛЯ ВИБОРУ ТУРИСТИЧНОГО МАРШРУТУ

Виконав: студент 4 курсу, групи 3КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Романюк Д.О.
(прізвище та ініціали)

Керівник:
к.т.н., доцент каф. КН Колодний В.В.
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент:
к.т.н., доцент каф. АІІТ Барабан М.В.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри Ярový Á.A.
«06» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» травня 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Романюку Дмитру Олександровичу

1. Тема роботи: WEB-платформа для вибору туристичного маршруту.

Керівник роботи: Колодний Володимир Володимирович, доцент кафедри КН.
затверджені наказом вищого навчального закладу від «20» березня 2025 року №

2. Термін подання студентом роботи 30 травня 2025 р.

3. Вихідні дані до роботи: Клієнтська та серверна частина WEB-платформи для вибору туристичного маршруту органайзера - 1 од.; кількість сутностей в базі даних - 4 од; мова програмування Python.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): визначення об'єкта, предмета, мети та задачі подальшого дослідження; аналіз предметної області, існуючих систем для планування маршрутів; проектування програмних модулів WEB-додатку для вибору туристичного маршруту; програмна реалізація модулів WEB-додатку для вибору туристичного маршруту

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслен блок-схема алгоритму роботи модулів побудови туристичного маршруту; ER-діагра розробленої бази даних; загальний вигляд інтерфейсу WEB-додатку;

1. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Колодний В.В., доцент каф. КН		

2. Дата видачі завдання 05 травня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вступ; Аналіз вимог і вибір алгоритмів маршрутизації;	05.05.2025	08.05.2025	
2	Проектування клієнт-серверної архітектури;	09.05.2025	12.05.2025	
3	Розробка функціональної схеми системи;	12.05.2025	14.05.2025	
4	Вибір інструментальних засобів; Реалізація модуля маршрутизації;	15.05.2025	17.05.2025	
5	Тестування платформи та аналіз продуктивності;	18.05.2025	26.05.2025	
6	Висновки;	27.05.2025	30.05.2025	
7	Оформлення додатків	01.06.2025	04.06.25	

Студент _____ Романюк Д.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Колодний В.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 92 сторінок формату А4, на яких є 21 рисунків, 3 таблиці, список використаних джерел містить 30 найменувань.

Бакалаврська кваліфікаційна робота є повним циклом розробки WEB-платформи для планування туристичних маршрутів. У цій бакалаврській роботі розроблено систему яка забеспечує інтерактивне планування туристичних маршрутів із використанням таких сучасних технологій як HTML/CSS/JavaScript для інтерфейсу, Google Maps/Places та Вікіпедії для отримання геологічних даних, та серверною частиною на Node.js із PostgreSQL (ODBC). Платформа спрямована на підвищення ефективності планування маршрутів шляхом їх оптимізації.

Ключові слова: WEB-платформа, туристичний маршрут, вебдодаток, Google Maps API, PostgreSQL, адаптивний дизайн, українізація, програмування.

ABSTRACT

The Bachelor's qualification thesis consists of 92 A4 pages, containing 21 figures, 3 tables, and a list of references with 30 entries.

The Bachelor's qualification thesis represents a complete cycle of developing a web platform for planning tourist routes. Within this thesis, a system has been developed that provides interactive tourist route planning using modern technologies such as HTML/CSS/JavaScript for the interface, Google Maps/Places, and Wikipedia for obtaining geographical data, along with a server-side component built on Node.js with PostgreSQL (ODBC). The platform is aimed at improving the efficiency of route planning through their optimization.

Keywords: web platform, tourist route, web application, Google Maps API, PostgreSQL, adaptive design, Ukrainization, programming.

ЗМІСТ

ВСТУП

Актуальність. Сучасний туризм переживає значні зміни завдяки цифровим технологіям, які трансформують підходи до планування подорожей. Велика частина туристів використовує WEB-платформи для пошуку інформації, але стикається з проблемами розпорошеної інформації, логістичних труднощів, недостатньої персоналізації послуг та застарілої інформації, що ускладнює планування та знижує задоволення від подорожей.

Метою дослідження є розширення функціональних можливостей програмного забезпечення для вибору туристичних маршрутів.

Об'єктом дослідження є процес створення WEB-платформи для вибору туристичних маршрутів.

Предметом дослідження є програмні засоби створення WEB-платформи для вибору туристичних маршрутів.

Задачі

- Аналіз проблеми організації туристичних маршрутів
- Аналіз відомих рішень для в галузі планування туристичних маршрутів
- Проектування архітектури WEB-платформи для вибору туристичного маршруту
- Програмна реалізація WEB-платформи для вибору туристичного маршруту
- Тестування WEB-платформи для вибору туристичного маршруту

Практичною цінністю є створення WEB-платформи для вибору туристичних маршрутів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ WEB-ПЛАТФОРМИ ДЛЯ ВИБОРУ ТУРИСТИЧНОГО МАРШРУТУ

1.1 Аналіз проблем при організації туристичних маршрутів

Туризм у сучасному світі зазнає кардинальних трансформацій під впливом цифрових технологій, які змінюють підходи до планування подорожей. Сучасні мандрівники дедалі частіше покладаються на WEB-ресурси для пошуку інформації про туристичні об’єкти, транспорт, проживання, місцеві події та додаткові послуги, що підкреслює необхідність створення зручних, функціональних і універсальних платформ. WEB-платформи для вибору туристичних маршрутів дозволяють об’єднати різноманітні дані в одному місці, значно спрощуючи процес підготовки до подорожі, зменшуючи витрати часу та мінімізуючи ризик помилок. Проте організація туристичних маршрутів залишається складним і багатогранним процесом через численні виклики, з якими стикаються мандрівники. Ці проблеми включають розпорошеність інформації, складність логістичного планування, обмежену персоналізацію, застарілі дані, труднощі з доступом до локальних рекомендацій, відсутність зручних інструментів для спільного планування, а також обмежену адаптивність для різних типів користувачів, зокрема тих, хто має особливі потреби. Розробка ефективної платформи, яка враховує ці виклики, вимагає глибокого аналізу потреб користувачів, сучасних технологічних можливостей і тенденцій у туристичній галузі.

Однією з найпоширеніших проблем є розпорошеність інформації, що значно ускладнює доступ до всіх необхідних даних для планування подорожі. Турист, який планує подорож до гірського регіону, може знайти інформацію про популярні локації, зокрема природні об’єкти чи курорти, на туристичних сайтах, але деталі про місцеві фестивалі, розклад міжміських автобусів, ціни на проживання чи відгуки про готелі часто розкидані по різних джерелах: туристичних порталах, соціальних мережах, особистих блогах, спеціалізованих форумах чи сайтах перевізників. За оцінками,

мандрівники в середньому витрачають 10–15 годин на пошук і систематизацію інформації для однієї поїздки, що включає порівняння даних із різних джерел. Користувачу доводиться перевіряти розклад транспорту на сайтах перевізників, шукати відгуки про заклади розміщення на туристичних платформах, а інформацію про локальні пам’ятки, зокрема регіональні культурні заходи, знаходити в геолокаційних сервісах, регіональних туристичних порталах чи групах у соціальних мережах. Така фрагментація інформації не лише знижує ефективність планування, але й підвищує ризик помилок, наприклад, бронювання готелю, який не відповідає очікуванням, чи пропуску важливого місцевого заходу через відсутність даних. Це призводить до стресу, втрати часу та фінансових ресурсів, а також може знизити загальне задоволення від подорожі ще до її початку.

Логістична складність є ще одним значним викликом, який ускладнює організацію маршрутів. Планування подорожі передбачає узгодження численних факторів: вибору локацій, типу транспорту, часу відвідування, тривалості перебування, а також врахування непередбачуваних обставин, Організація поїздки до кількох міст регіону може бути ускладнена через невідповідність розкладу громадського транспорту, а також графіків роботи музеїв, замків чи інших об’єктів. Якщо мандрівник планує відвідати визначну культурну пам’ятку в одному місті та регіональну подію в іншому, йому необхідно точно розрахувати час на дорогу, тривалість перебування в кожному місті, а також можливі затримки через ремонт доріг чи погодні умови. Відсутність інструменту, який автоматично синхронізує ці дані та пропонує оптимальний маршрут із урахуванням усіх факторів, змушує користувачів вручну перевіряти інформацію на різних платформах, що є трудомістким і часто неефективним. Для групових подорожей, таких як сімейні чи дружні поїздки, проблема посилюється, оскільки координація між учасниками зазвичай відбувається через месенджери, що не завжди зручно для створення єдиного плану, його редагування чи синхронізації. Група туристів, яка планує

подорож, може витратити багато часу на узгодження маршруту через листування, що знижує ефективність і викликає додатковий стрес.

Обмежена персоналізація наявних рішень є ще однією значною проблемою, яка впливає на якість планування. Більшість туристичних платформ пропонують стандартні маршрути, орієнтовані на масового туриста, які рідко враховують індивідуальні інтереси чи специфічні потреби.

Мандрівники, які цікавляться гастрономічним туризмом, часто стикаються з проблемою відсутності рекомендацій щодо закладів із місцевою кухнею на популярних платформах. Регіональні заклади з місцевою кухнею чи підприємства з виробництва традиційних напоїв часто залишаються непоміченими для широкого кола туристів. Любителі екстремального відпочинку, зокрема активних видів спорту чи пригод на природі, змушені шукати інформацію на спеціалізованих ресурсах, таких як форуми чи сторінки місцевих туроператорів, щоб знайти відповідні активності.

Стандартні туристичні маршрути, пропоновані популярними платформами, часто обмежуються відомими локаціями, зокрема культурними пам’ятками чи центральними площами міст. Це призводить до того, що маловідомі пам’ятки, зокрема історичні архітектурні об’єкти, регіональні події чи традиційні майстер-класи, залишаються недооціненими та недоступні для більшості туристів.

Актуальність інформації є критично важливою для ефективного планування, але на багатьох туристичних платформах дані швидко застарівають, що створює значні незручності. Мандрівник може спланувати відвідування історичної пам’ятки, але на місці виявити, що вона тимчасово недоступна через реставрацію через відсутність оновлених даних на сайті. Зміни в розкладі транспорту, цінах на квитки чи погодних умовах можуть суттєво вплинути на плани, але ці дані рідко оновлюються в реальному часі. Турист, який планує похід у гірський регіон, може не врахувати погіршення погодних умов через застарілу інформацію, що призведе до зриву планів або навіть небезпечних ситуацій. Неточні відгуки чи описи пам’яток також можуть вводити

користувачів в оману, завищені оцінки ресторану чи застаріла інформація про години роботи музею, знижують довіру до платформи та погіршують враження від подорожі.

Туристи часто прагнуть відкривати унікальні, менш відомі місця, які рекомендують місцеві жителі чи досвідчені мандрівники, але доступ до таких рекомендацій залишається обмеженим. У невеликому місті турист може бути зацікавлений у відвідуванні місцевих ринків із традиційними виробами, майстер-класів із регіональних ремесел чи локальних культурних подій. Однак ці об’єкти рідко згадуються в популярних туристичних путівниках, які зосереджені на відомих туристичних локаціях регіону. Соціальні мережі, блоги чи форуми можуть містити корисні поради, але їх пошук є трудомістким, а інформація часто неструктурована і розкидана по різних джерелах. Мандрівник може знайти відгук про унікальний ресторан, але систематизація таких даних вимагає значних зусиль. Відсутність платформи, яка б систематизувала локальні знання та рекомендації, знижує можливість відкривати нові місця, які могли б зробити подорож більш запам’ятовною та індивідуальною.

Ще однією проблемою є обмежена адаптивність наявних платформ до потреб різних груп користувачів. Мандрівники з особливими потребами, можуть стикатися з труднощами через відсутність інформації про доступність пам’яток, транспорту чи готелів. Сім’ї з дітьми можуть потребувати маршрутів із дитячими майданчиками чи закладами, які пропонують дитяче меню, але такі дані рідко доступні на стандартних платформах. Літні мандрівники можуть шукати маршрути з мінімальною фізичною активністю чи зручним транспортом, але більшість платформ не пропонують таких фільтрів. Турист із інвалідністю, може не знайти інформації про доступність даних закладів, що ускладнює планування. Це підкреслює необхідність інклюзивного підходу в розробці платформ.

Ці проблеми не лише ускладнюють процес планування, але й мають значний вплив на загальний досвід мандрівників. Витрачений час на пошук інформації, помилки через

застарілі дані чи невідповідність розкладу можуть викликати стрес і розчарування ще до початку поїздки. Відсутність персоналізації та адаптивності знижує задоволення від подорожі, оскільки користувачі не отримують доступу до місць чи активностей, які відповідають їхнім інтересам чи потребам. Помилки в плануванні, такі як невідповідність розкладу транспорту чи закриття пам'ятки, можуть призвести до втрати часу та фінансових ресурсів, що погіршує враження від подорожі.

Такі виклики підкреслюють необхідність створення єдиної WEB-платформи, яка б об'єднала актуальну інформацію, автоматизувала логістичне планування, забезпечувала персоналізований та інклюзивний підхід, надавала доступ до локальних рекомендацій і підтримувала спільне планування без необхідності авторизації, щоб зробити процес підготовки до подорожі зручним, ефективним і приємним для всіх категорій користувачів [1-8].

1.2 Аналіз відомих рішень в галузі вибору туристичних маршрутів

Для створення ефективної WEB-платформи для вибору туристичного маршруту необхідно ретельно проаналізувати наявні рішення, щоб врахувати їхні сильні сторони, усунути недоліки та запропонувати унікальні функції, які вирізняють нову платформу серед конкурентів. У цьому підрозділі розглядаються п'ять популярних платформ для планування подорожей: TripAdvisor, Google Travel (вбудоване в Google Maps), Komoot, Wanderlog і Roadtrippers. Кожна з них аналізується за критеріями функціональності, зручності інтерфейсу, геолокаційних можливостей, рівня персоналізації, локалізації та доступності, щоб визначити, як їхні особливості можуть бути використані чи вдосконалені в новій платформі, яка розробляється на Python з використанням Bootstrap і без авторизації користувача.

TripAdvisor є однією з найпопулярніших платформ у сфері туризму, яка пропонує широкий спектр функцій для планування подорожей. Вона надає інформацію про

туристичні об’єкти, готелі, ресторани, маршрути, відгуки користувачів і можливість бронювання через партнерські сервіси. Інтерфейс платформи зручний, із фільтрами за типом активності (культурні пам’ятки, природа, гастрономія), бюджетом чи розташуванням, що полегшує пошук релевантних маршрутів. Користувач може швидко знайти заклади з місцевою кухнею в місті чи готелі поблизу туристичного регіону. Геолокаційні функції, реалізовані через інтеграцію з Google Maps, забезпечують точне відображення пам’яток і побудову маршрутів між ними. Персоналізація базується на історії пошуку та відгуках, що дозволяє ппропонувати релевантні рекомендації, зокрема екскурсії до історичних пам’яток регіону для поціновувачів історії. Однак платформа має недоліки: багато функцій, таких як детальні маршрути чи бронювання, доступні лише через партнерські сервіси, що ускладнює досвід. Мобільний додаток іноді працює повільно через велику кількість контенту, а реклама на сторінках може відволікати користувачів. Крім того, TripAdvisor рідко пропонує рекомендації для менш відомих місць, що обмежує її корисність для нішевих інтересів.

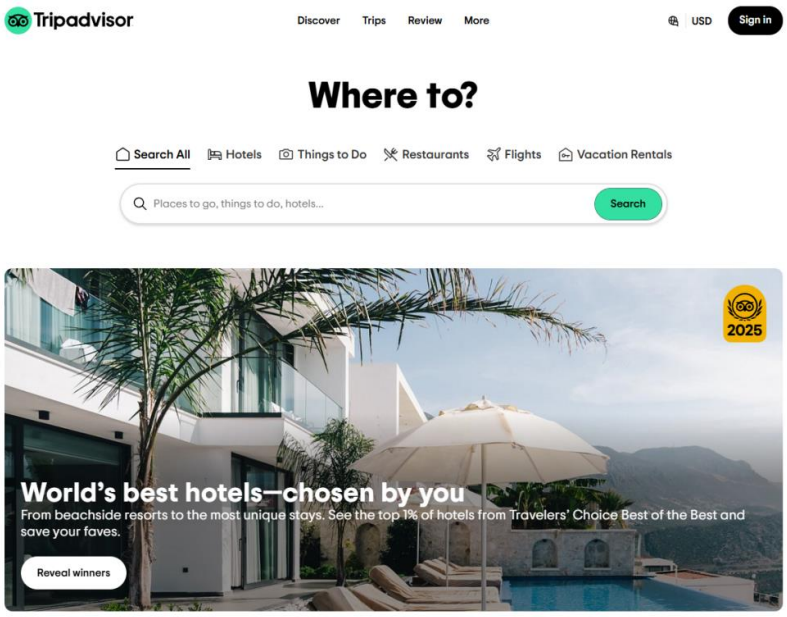

Рисунок 1.1 – Головна сторінка TripAdvisor

Google Maps із вбудованими функціями Google Travel є потужним інструментом для планування маршрутів, який вирізняється своєю геолокаційною точністю та універсальністю. Платформа пропонує детальні карти, маршрутизацію для різних видів транспорту, офлайн-доступ і автоматичне створення маршрутів на основі бронювань чи геолокації. Інтерфейс є мінімалістичним і швидким, що дозволяє легко знаходити пам’ятки, ресторани чи готелі, а також розраховувати час і відстань між локаціями. Геолокаційні можливості Google Maps є найкращими на ринку завдяки високій деталізації карт і інтеграції з іншими сервісами Google, що дозволяє автоматично імпортувати дані про бронювання. Проте персоналізація обмежена: рекомендації часто базуються на популярних локаціях і рідко враховують унікальні інтереси. Деякі функції потребують попереднього завантаження та налаштувань, що може бути незручним для менш досвідчених користувачів. Українізація інтерфейсу є частковою, що може створювати бар’єри для україномовних користувачів.

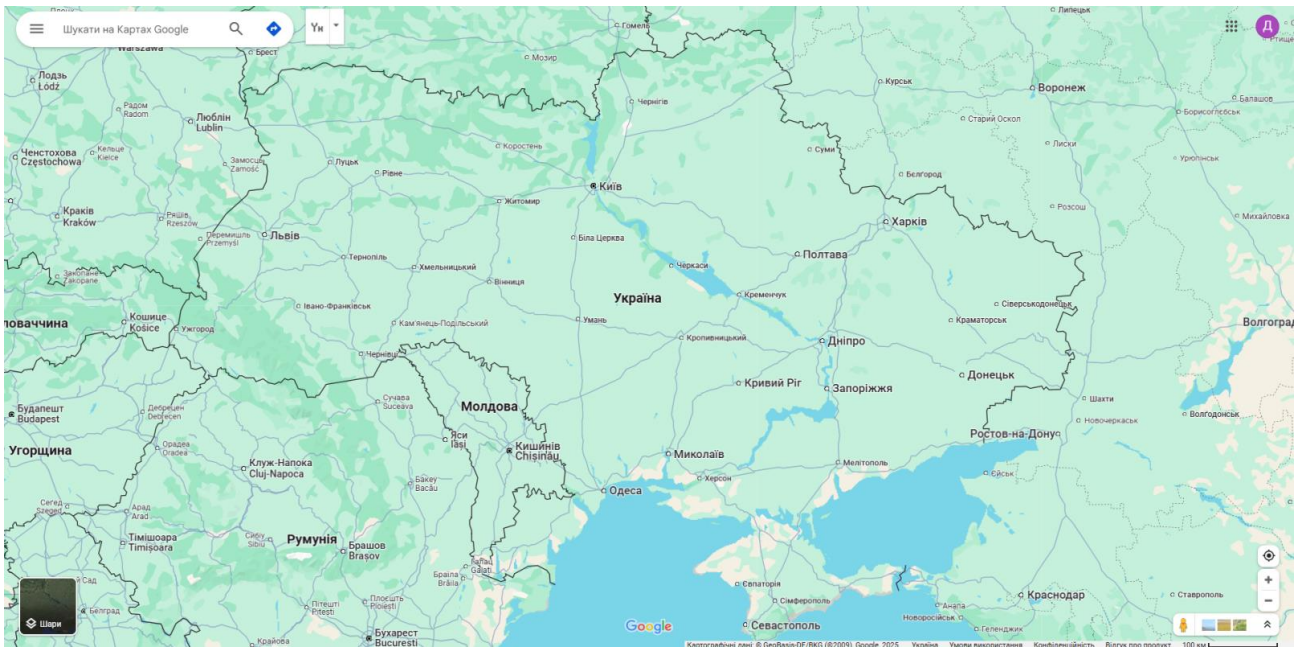

Рисунок 1.2 – Головна сторінка Google Maps

Komoot є спеціалізованою платформою для активного відпочинку, такої як піші прогулянки, велотуризм і біг. Вона пропонує детальні карти, створені спільнотою користувачів, які включають інформацію про тип місцевості, рівень складності маршруту та тривалість подорожі. Інтерфейс Komoot є інтуїтивним і зосередженим на навігації, що робить його ідеальним для планування активних маршрутів. Геолокаційні функції включають офлайн-навігацію, що є важливим для віддалених маршрутів із обмеженим інтернет-з'єднанням. Персоналізація є сильною стороною платформи, оскільки користувачі можуть налаштовувати маршрути під свої фізичні можливості та інтереси. Однак Komoot зосереджений виключно на активному відпочинку, що робить його менш придатним для міських чи культурних подорожей. Офлайн-карти чи розширені інструменти планування, доступні лише за платною підпискою, що може відштовхувати користувачів, які шукають безкоштовні рішення. Українізація платформи обмежена, що знижує її доступність для місцевих користувачів.

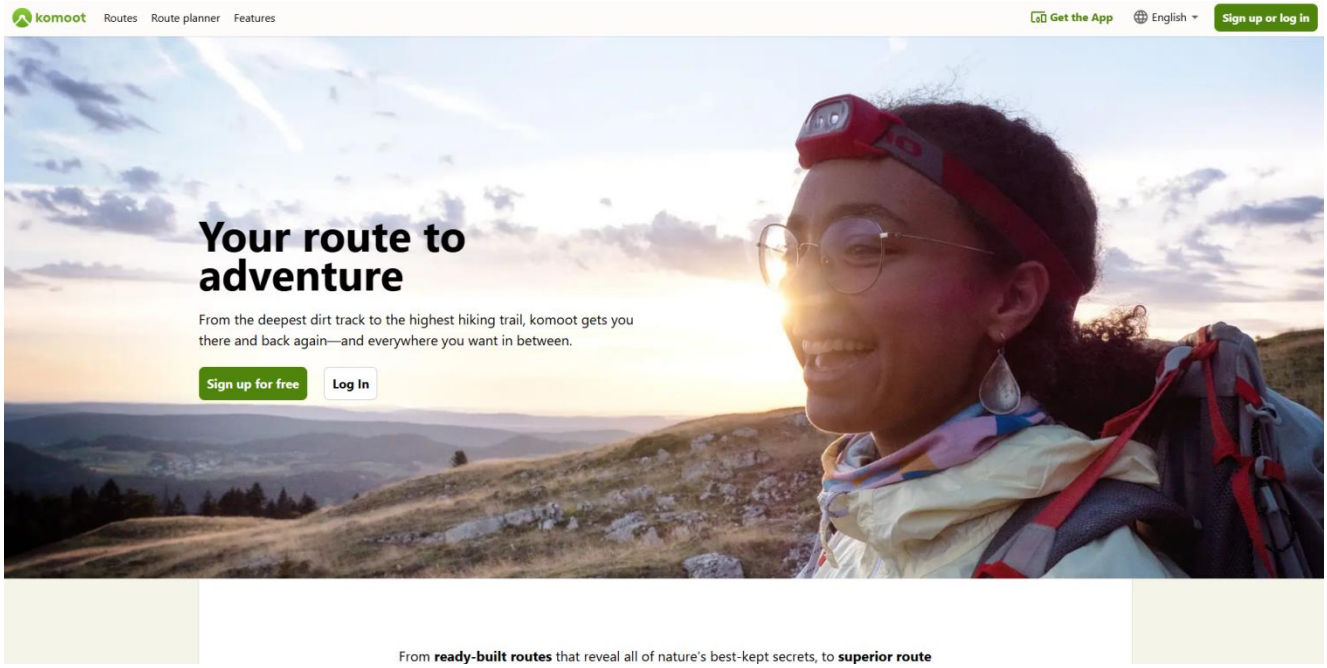

Рисунок 1.3 – Головна сторінка Komoot

Wanderlog є відносно новою платформою, яка фокусується на створенні та управлінні туристичними маршрутами з акцентом на зручність і співпрацю. Wanderlog дозволяє користувачам створювати детальні плани подорожей, додавати пам’ятки, ресторани, готелі та особисті нотатки, а також ділитися маршрутами з іншими мандрівниками. Інтерфейс платформи є сучасним і зручним, з можливістю перегляду маршрутів на карті та автоматичного розрахунку часу й відстаней. Геолокаційні функції базуються на інтеграції з Google Maps, що забезпечує точність і надійність. Wanderlog підтримує спільне редагування маршрутів, що робить його ідеальним для групових подорожей. Персоналізація є відносно сильною: користувачі можуть додавати власні інтереси та отримувати відповідні рекомендації. Проте експорт маршрутів у PDF чи офлайн-доступ, доступні лише в преміум-версії, що може бути незручним. Локалізація для україномовних користувачів обмежена, оскільки інтерфейс і контент переважно англійською мовою, що створює бар’єри для користувачів в Україні.

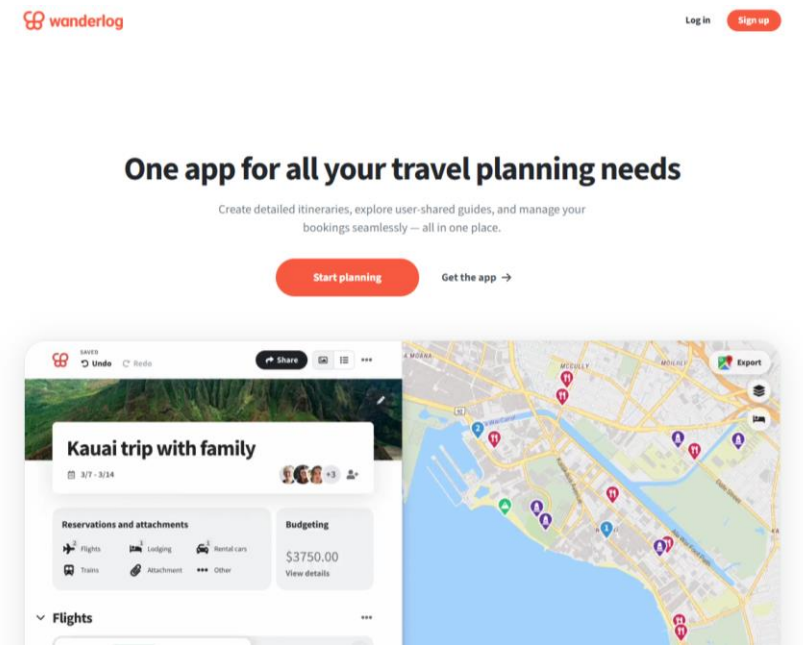

Рисунок 1.4 – Головна сторінка Wanderlog

Roadtrippers є платформою, орієнтованою на планування дорожніх подорожей, із акцентом на автомобільні маршрути та унікальні локації. Вона дозволяє користувачам створювати маршрути, додавати зупинки, а також знаходити незвичайні місця – локальні придорожні атракції чи маловідомі природні об'єкти. Інтерфейс Roadtrippers є інтуїтивним, із можливістю перегляду маршрутів на карті та розрахунку витрат пального, що корисно для довгих поїздок. Геолокаційні функції базуються на інтеграції з картами, що забезпечує точність маршрутизації. Платформа вирізняється сильною персоналізацією, пропонуючи рекомендації на основі інтересів користувача: історичних пам'яток чи гастрономічних локацій. Roadtrippers також підтримує спільне планування, дозволяючи групам мандрівників редагувати маршрути разом. Проте платформа має обмеження: багато функцій, таких як офлайн-доступ чи детальна аналітика маршрутів, доступні лише за платною підпискою. Локалізація для україномовних користувачів практично відсутня, оскільки платформа орієнтована переважно на англомовну аудиторію, що обмежує її використання в Україні. Крім того, фокус на автомобільних подорожах робить її менш придатною для піших чи міських маршрутів.

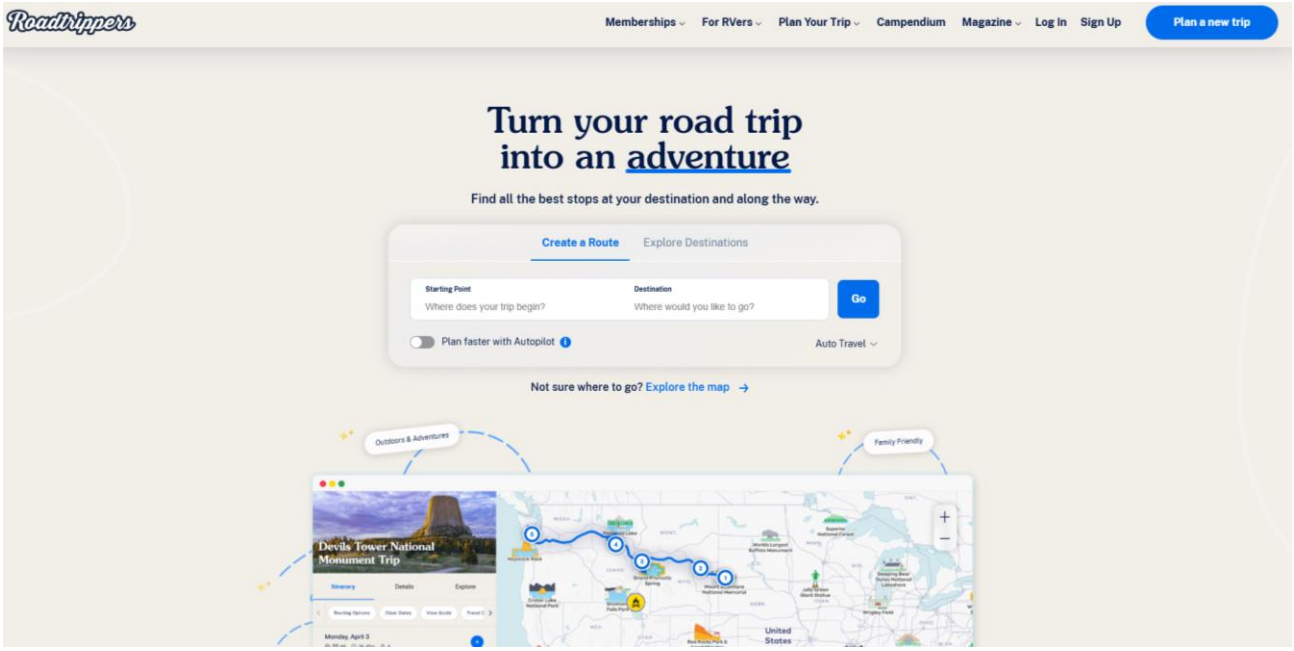

Рисунок 1.5 – Головна сторінка Roadtrippers

Для порівняння аналогів наведено таблицю, яка оцінює їх за ключовими критеріями [9-17].

Таблиця 1.1 – Порівняльна характеристика аналогів обрання туристичних маршрутів

Критерій	**TripAdvisor**	**Google Maps**	**Komoot**	**Wanderlog**	**Roadtrippers**
Функціонал	Широкий вибір маршрутів, відгуки, бронювання	Точна маршрутизація, офлайн-доступ	Планування активних маршрутів, офлайн-навігація	Створення маршрутів, спільне редагування, інтеграція з картами	Планування дорожніх подорожей, унікальні локації
Інтерфейс	Зручний, але перевантажений рекламою	Мінімалістичний, швидкий	Інтуїтивний орієнтований на активність	Сучасний, зручний, але обмежена локалізація	Інтуїтивний, але орієнтований на автоподорожі
Геолокація	Точні карти через Google Maps	Найкраща деталізація, офлайн-карти	Детальні карти, офлайн-навігація	Точні карти через Google Maps	Точні карти, але обмежена офлайн-навігація
Персоналізація	На основі відгуків і пошуку	Обмежена, базується на популярності	Висока для активних маршрутів	Помірна, залежить від інтересів користувача	Висока, фокус на унікальних локаціях
Ціна	Безкоштовно, платні функції	Безкоштовно	Безкоштовно, платна підписка	Безкоштовно, платні преміум-функції	Безкоштовно, платна підписка
Локалізація	Обмежена україномовна підтримка	Часткова україномовна підтримка	Обмежена україномовна підтримка	Переважно англійською	Переважно англійською

Аналіз аналогів показує, що TripAdvisor є універсальним, але перевантажений рекламою та залежить від партнерських сервісів. Google Maps пропонує неперевершену геолокацію, але бракує персоналізації. Komoot ідеальний для активного відпочинку, але

не підходить для міських чи культурних маршрутів. Wanderlog вирізняється зручним спільним плануванням, але обмежена локалізація знижує її доступність. Roadtrippers пропонує унікальні локації для автоподорожей, але платні функції та відсутність україномовного інтерфейсу ускладнюють її використання в Україні. Нова платформа, розроблена на Python із використанням Bootstrap, має поєднувати універсальність TripAdvisor, точність Google Maps, персоналізацію Komoot і Roadtrippers, а також зручність Wanderlog, забезпечуючи україномовний інтерфейс і безкоштовний доступ до основних функцій без авторизації.

1.3 Постановка задачі та розробка вимог до WEB-платформи

WEB-платформа для вибору туристичного маршруту має стати універсальним інструментом, який спрощує процес планування подорожей і забезпечує доступ до актуальної, структурованої та персоналізованої інформації. Вона повинна надавати користувачам можливість створювати, редагувати та зберігати маршрути, отримувати детальні дані про туристичні об’єкти, транспорт, проживання, місцеві події та послуги, а також переглядати рекомендації інших мандрівників. Платформа розробляється на Python із використанням Bootstrap для адаптивного дизайну і не передбачає авторизації користувача, що забезпечує простоту використання. Вона має бути зручною, орієнтованою на україномовних користувачів і масштабованою для потенційного розширення на міжнародну аудиторію.

Платформа повинна мати інтуїтивний інтерфейс для пошуку туристичних об’єктів і створення маршрутів. Система пошуку має підтримувати запити за назвами, категоріями, географічним розташуванням, бюджетом і типом активності. Інструмент створення маршрутів дозволяє додавати пам’ятки, транспорт, час відвідування та нотатки. Інтерактивна карта відображає маршрути та об’єкти з розрахунком відстаней і часу, використовуючи Google Maps API. Адаптивний дизайн на основі Bootstrap

забезпечує зручність на різних пристроях. Без авторизації маршрути зберігаються локально у форматі JSON, дозволяючи користувачам завантажувати їх для подальшого використання. Система відгуків дозволяє переглядати оцінки пам’яток чи ресторанів. Українізований інтерфейс із можливістю додавання інших мов (англійська, польська) забезпечує доступність. Інтеграція API для актуальних даних про розклад транспорту чи погоду мінімізує помилки. Безпека забезпечується механізмами Django, архітектура підтримує масштабованість, а офлайн-доступ дозволяє використовувати маршрути без інтернету. [18-24].

1.4 Висновок до розділу 1

У цьому розділі було доведено актуальність створення WEB-платформи для вибору туристичного маршруту , оскільки він містить якісну українську локалізацію, зручний інтерфейс та безкоштовний доступ без реклами.

Проведено аналіз щодо існуючих WEB-платформ для вибору туристичного маршруту. Дослідження показало, що існуючі рішення обмежені слабкою українською локалізацією та великою кількістю реклами, що погіршує досвід їх використання.

Було сформульовано вимоги та поставлена задача дослідження, яка передбачатиме створення системи опрацювання даних, побудови маршруту та взаємодій з зовнішнім API.

2 ПРОЕКТУВАННЯ WEB-ПЛАТФОРМИ ДЛЯ ВИБОРУ ТУРИСТИЧНОГО МАРШРУТУ

Розробка WEB-платформи для вибору туристичного маршруту є комплексним завданням, яке вимагає ретельного планування для забезпечення зручності, ефективності, масштабованості та інтуїтивності. Цей розділ детально описує вибір архітектури системи, створення UML-діаграм, проектування клієнтської частини, розробку алгоритмів і аналіз їхньої реалізації. Проектування ґрунтується на вимогах, визначених у першому розділі, зокрема на створенні адаптивного інтерфейсу в біло-синіх тонах, використанні реляційної бази даних із підтримкою стандарту ODBC, реалізації щонайменше чотирьох сутностей, а також на вирішенні проблем, таких як розпорошеність туристичної інформації, логістична складність, обмежена персоналізація, застарілі дані, брак локальних рекомендацій і недостатня адаптивність. Платформа розробляється з використанням Python і фреймворку Django для серверної логіки, бібліотеки Bootstrap для адаптивного дизайну, Google Maps API для геолокаційних функцій і не потребує авторизації користувачів, зберігаючи маршрути локально у форматі JSON. У порівнянні з аналогами (TripAdvisor, Google Maps, Komoot, Wanderlog, Roadtrippers) платформа фокусується на україномовному інтерфейсі, персоналізації та інтеграції локальних даних.

2.1 Вибір архітектури WEB-платформи

Вибір архітектури визначає структуру платформи, її здатність обробляти запити, інтегруватися із зовнішніми сервісами, як-от Google Maps API, і забезпечувати зручність для користувачів із різним рівнем технічної підготовки. Було проаналізовано п’ять архітектурних підходів: MVC (Model-View-Controller), монолітна, мікросервісна, безсерверна та клієнт-серверна, щоб обрати оптимальний варіант для платформи, яка

має обробляти геолокаційні дані, підтримувати локальне збереження маршрутів і відповідати вимогам адаптивності та простоти підтримки.

MVC-архітектура розділяє систему на три компоненти: модель, що відповідає за управління даними (інформація про пам’ятки, маршрути чи відгуки), представлення, яке відображає дані у вигляді HTML-сторінок, карт або списків, і контролер, що обробляє запити користувача, викликаючи методи моделі та передаючи результати до представлення. MVC забезпечує чітку структуризацію коду, що спрощує розробку, тестування та подальше розширення системи. Вона ідеально інтегрується з Django, який використовує схожий підхід MTV (Model-Template-View), де шаблони відповідають за відображення, а моделі — за роботу з даними. Однак для невеликих проєктів MVC може бути надмірно складною, а тісний зв’язок між моделлю та представленням означає, що зміни в базі даних можуть вимагати коригування інтерфейсу.

Монолітна архітектура об’єднує весь функціонал — від пошуку пам’яток до побудови маршрутів — в єдину програму, що розгортається на одному сервері. Це спрощує початкову розробку, оскільки всі компоненти знаходяться в одному кодовому базисі, і полегшує розгортання, адже не потрібна складна інфраструктура. Моноліт підходить для проєктів із чітко визначеним функціоналом, як у цьому випадку. Проте при зростанні кількості користувачів масштабування моноліту вимагає збільшення ресурсів для всієї системи, а зміни в одній частині можуть викликати неочікувані проблеми в інших.

Мікросервісна архітектура розбиває систему на незалежні сервіси, кожен із яких виконує окрему функцію – пошук пам’яток, розрахунок маршрутів чи обробка відгуків. Це дозволяє масштабувати лише потрібні компоненти, використовувати різні технології для різних сервісів і підвищує надійність, адже збій одного сервісу не зупиняє всю систему. Однак мікросервіси ускладнюють розробку через необхідність координації між сервісами, підвищують витрати на інфраструктуру (потрібні окремі сервери чи контейнери) і вимагають складнішого тестування для перевірки взаємодії компонентів.

Безсерверна архітектура передає управління серверами хмарним провайдерам, як-от AWS Lambda чи Google Cloud Functions, де окремі функції виконуються за потреби. Це оптимізує витрати, оскільки плата стягується лише за використані ресурси, і забезпечує автоматичне масштабування. Безсерверний підхід дозволяє розробникам зосередитися на коді, а не на управлінні серверами. Проте він обмежує контроль над системою, адже платформа залежить від провайдера, а виконання функцій може бути обмежене за часом чи пам'яттю. Тестування в хмарному середовищі також ускладнене через віддалений доступ до ресурсів.

Клієнт-серверна архітектура поділяє систему на клієнтську частину, яка працює в браузері (відображення карт, списків пам'яток), і серверну, що обробляє логіку та дані. Це забезпечує гнучкість, дозволяючи використовувати різні технології для клієнта і сервера, а також спрощує масштабування серверної частини при зростанні навантаження. Однак продуктивність залежить від стабільності мережі, а синхронізація між клієнтом і сервером вимагає розробки надійного API, що може ускладнити початкову реалізацію.

Для платформи обрано MVC-архітектуру з елементами клієнт-серверного підходу, реалізовану через фреймворк Django. Цей вибір обґрунтований кількома факторами. По-перше, MVC відповідає вимогам проєкту середньої складності, де потрібно чітко розділити логіку обробки даних, відображення інтерфейсу та взаємодію з базою даних. По-друге, Django забезпечує природну підтримку MVC через MTV, що прискорює розробку завдяки вбудованим інструментам для роботи з моделями, шаблонами та маршрутизацією. По-третє, клієнт-серверний підхід дозволяє ефективно інтегрувати Google Maps API для геолокаційних функцій, де клієнтська частина відображає карту, а серверна обробляє запити. Локальне збереження маршрутів у JSON не вимагає складної серверної логіки, що зменшує навантаження на систему. Нарешті, використання Bootstrap для клієнтської частини гарантує адаптивний дизайн, який працює на різних пристроях, від смартфонів до настільних комп'ютерів. [24-29]

Візуалізація MVC-архітектури на рисунку 2.1.

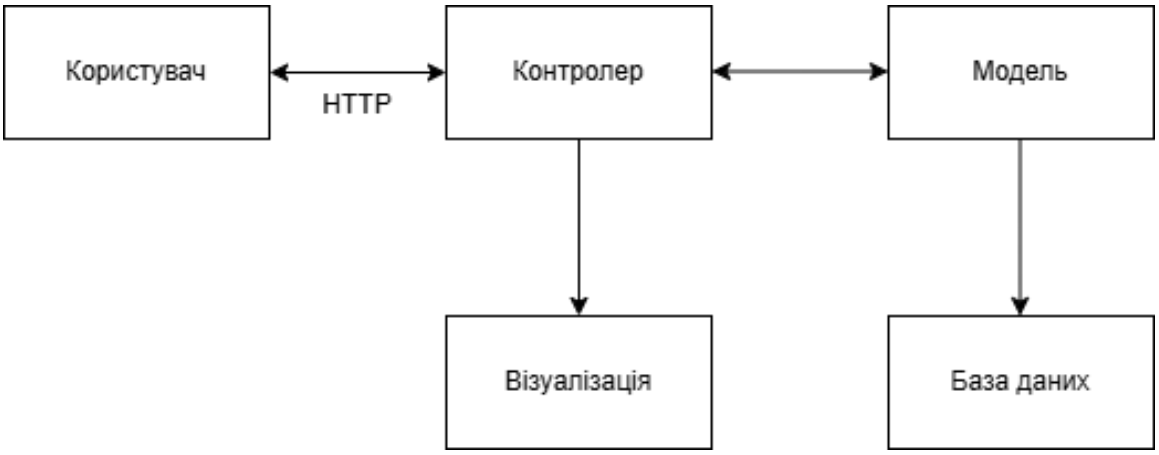

Рисунок 2.1 – Схематична візуалізація MVC архітектури

2.2 Розробка UML-діаграм та проектування клієнтської частини

UML-діаграми є необхідними для моделювання функціоналу, взаємодії компонентів і фізичної структури платформи. Вони допомагають чітко визначити сценарії використання, послідовність дій і розподіл ресурсів. Паралельно розробляється клієнтська частина з використанням Bootstrap, щоб забезпечити адаптивний, інтуїтивний інтерфейс.

Діаграма прецедентів (рис. 2.2) описує, як користувач (турист) взаємодіє з платформою. Основні дії включають пошук туристичних об’єктів за різними критеріями (назва, категорія — культура, природа, гастрономія, розташування, бюджет), створення маршрутів шляхом додавання пам’яток, вибору транспорту (авто, громадський, піший) і визначення часу перебування, перегляд маршрутів на інтерактивній карті з відображенням відстаней і часу, збереження маршрутів локально у форматі JSON, перегляд відгуків про пам’ятки чи заклади харчування, а також отримання персоналізованих рекомендацій на основі популярності маршрутів чи вподобань

користувача. Ці прецеденти пов’язані: наприклад, створення маршруту залежить від результатів пошуку, а рекомендації базуються на аналізі збережених маршрутів.

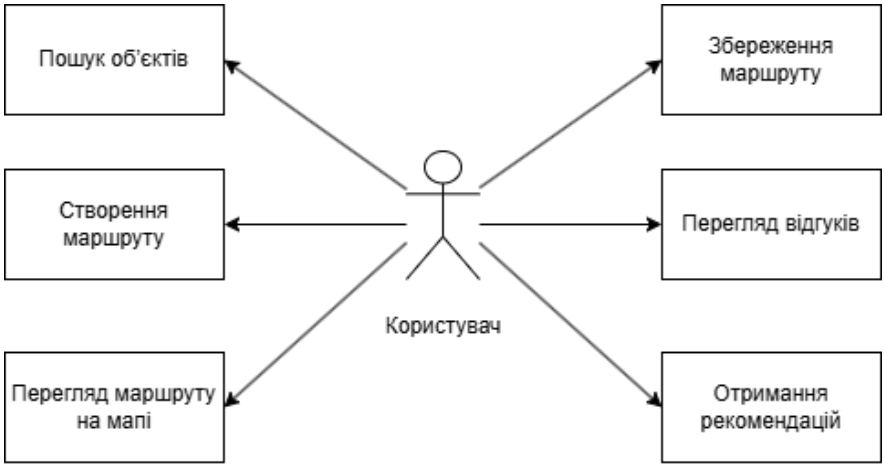

Рисунок 2.2 – Візуалізація діаграми прецедентів

Діаграма розгортання (рис. 2.3) описує фізичну інфраструктуру платформи. Клієнтський пристрій (браузер користувача) взаємодіє з вебсервером, на якому розгорнуто Django та Python. Вебсервер звертається до сервера бази даних (PostgreSQL) через ODBC для отримання чи збереження даних, таких як інформація про пам’ятки чи відгуки. Окремо вебсервер інтегрується з Google Maps API через HTTPS-запити для розрахунку маршрутів і відображення карт. Усі компоненти пов’язані мережевими протоколами, що забезпечує гнучкість і масштабованість.

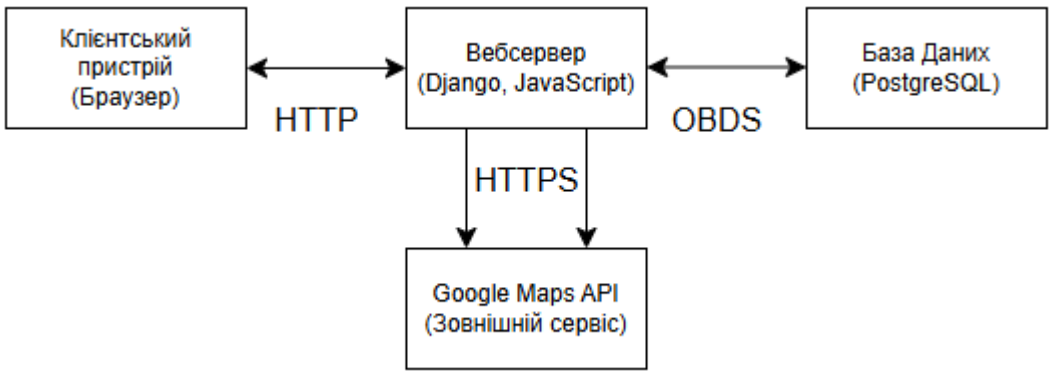

Рисунок 2.3 – Візуалізація діаграми розгортання

Діаграма послідовності (рис. 2.4) деталізує процес створення маршруту, одного з ключових сценаріїв. Користувач вводить запит на пошук пам’яток у пошуковій панелі.

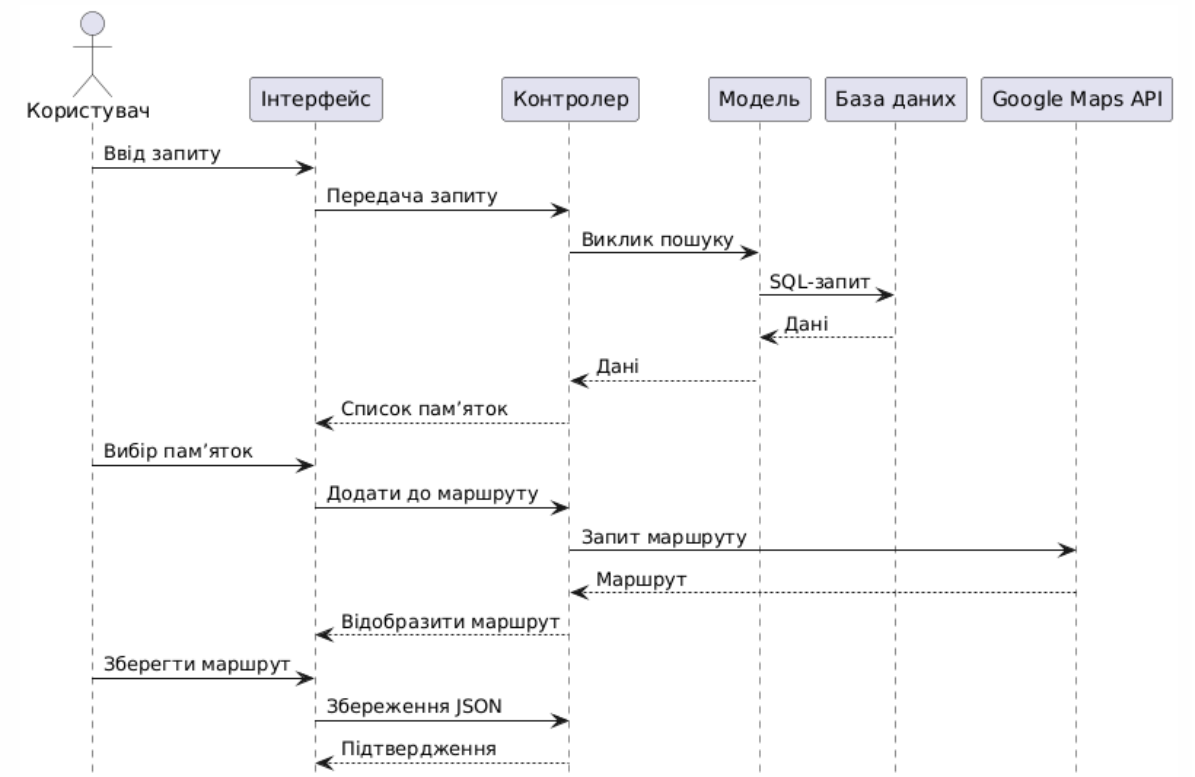

Рисунок 2.4 – Візуалізація діаграми послідовності

Інтерфейс (HTML-сторінка з JavaScript) передає запит контролеру (Django). Контролер викликає метод моделі, яка формує SQL-запит до бази даних, отримуючи список відповідних пам’яток (рис. 2.5). Модель повертає дані контролеру, який формує відповідь, і інтерфейс відображає список пам’яток із фільтрами. Користувач обирає пам’ятки, додаючи їх до маршруту, і вказує транспорт та час. Контролер надсилає запит до Google Maps API для розрахунку відстаней і часу між точками, отримуючи оптимальний маршрут. Інтерфейс відображає маршрут на карті з маркерами та

деталями. Нарешts контролер формує JSON-об’єкт, який зберігається в локальному сховищі браузера (localStorage) або завантажується як файл.

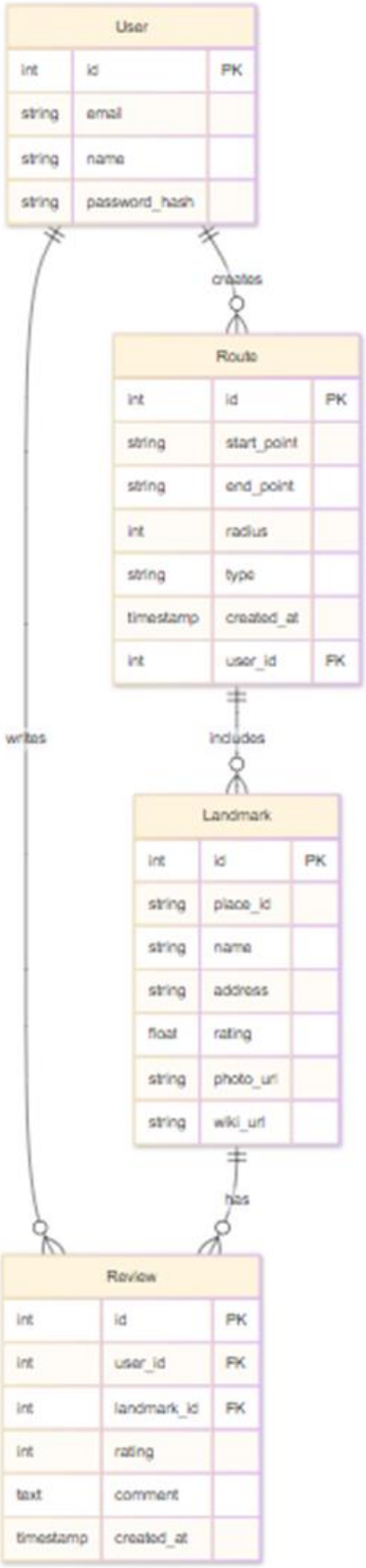
User
int id PK
string email
string name
string password_hash
creates
Route
int id PK
string start_point
string end_point
int radius
string type
timestamp created_at
int user_id FK
writes
includes
Landmark
int id PK
string place_id
string name
string address
float rating
string photo_url
string wiki_url
has
Review
int id PK
int user_id FK
int landmark_id FK
int rating
text comment
timestamp created_at

Рисунок 2.5 – Схема бази даних

2.3 Розробка алгоритмів для WEB-платформи

Алгоритми платформи забезпечують побудову маршрутів, пошук пам’яток, фільтрацію та відображення, оптимізуючи швидкодію й API-запити. Вони базуються на функціях коду, представлені через блок-схеми, оцінку складності та порівняння.

Алгоритм побудови маршруту

Алгоритм обробляє введення, яке включає початкову і кінцеву точки, радіус пошуку та тип маршруту, після чого викликає Google Maps Directions API, обирає оптимальний маршрут і запускає пошук пам’яток. Спочатку алгоритм отримує дані з полів вводу і радіокнопок, де користувач вказує початкову та кінцеву точки, радіус пошуку та вибирає тип маршруту. Після цього відбувається перевірка валідності введених даних: якщо будь-яке з полів залишилось порожнім, користувач отримує повідомлення alert з просьбою заповнити їх. Якщо всі дані введені коректно, алгоритм переходить до наступного етапу, де знову перевіряється валідність введених даних, і, у разі виявлення помилок, користувач знову отримує повідомлення alert.

Після успішної перевірки валідності алгоритм обирає тип маршруту, який був вибраний користувачем (найкоротший, найшвидший або маршрут з максимальною кількістю пам’яток). Наступним кроком відправляється запит до Directions API Google Maps, який використовується для отримання можливих маршрутів між вказаними точками. Отримавши відповідь від API, алгоритм вибирає маршрут, відповідно до вибраного типу: для найкоротшого маршруту визначається маршрут з мінімальною відстанню, для найшвидшого маршруту — маршрут з мінімальним часом проходження, а для маршруту з максимальною кількістю пам’яток — маршрут, що проходить найбільшу кількість історичних об’єктів.

Після вибору оптимального маршруту алгоритм запускає пошук пам’яток в обраному радіусі від маршруту. Цей пошук використовує дані з API, що містять інформацію про розташування пам’яток, їх назви та інші відповідні дані. Нарешті, після завершення пошуку пам’яток, карта та картки на веб-сторінці оновлюються, відображаючи обраний маршрут та знайдені пам’ятки. Користувач отримує візуальне представлення маршруту, а також детальну інформацію про кожну пам’ятку, яка знаходиться на цьому маршруту.

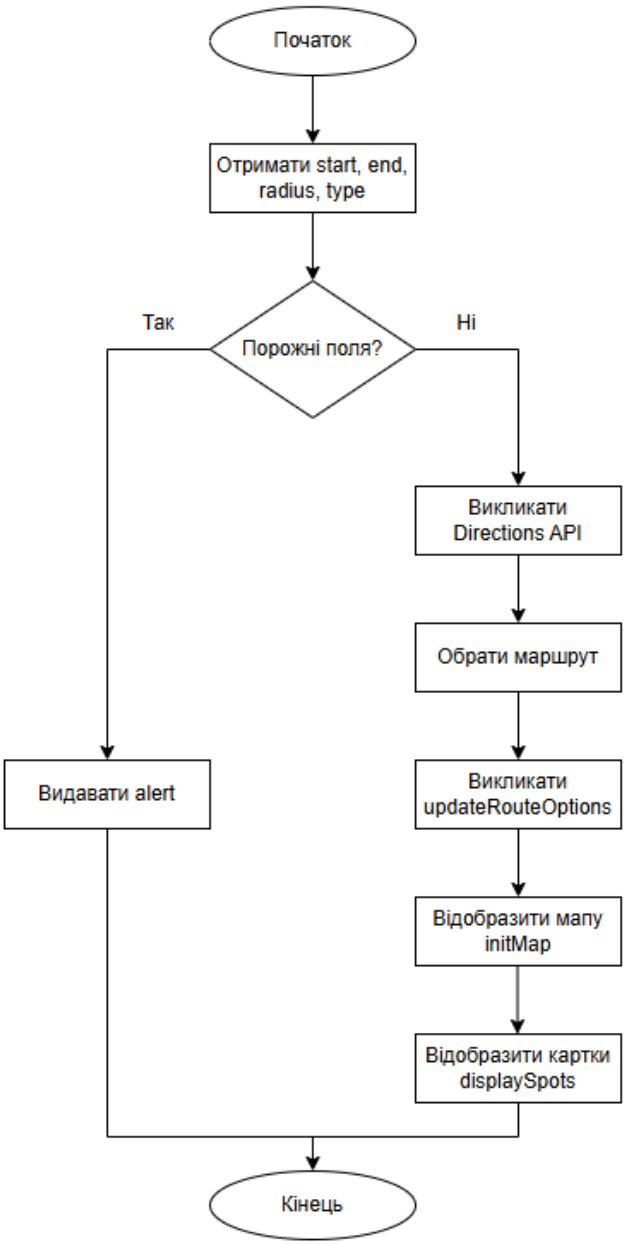

Рисунок 2.6 – Блок-схема алгоритму побудови маршруту

Алгоритм пошуку пам’яток

Алгоритм шукає пам’ятки через Google Places API, виконуючи наступні кроки.

Спочатку алгоритм фільтрує кожну п’яту точку маршруту для оптимізації пошуку. Це дозволяє значно зменшити кількість запитів до API, що знижує навантаження на систему і забезпечує більш швидку обробку даних. Забезпечення ефективності пошуку

є важливим аспектом, оскільки надмірна кількість запитів може призвести до обмеження доступу до API або до погіршення продуктивності системи.

Після фільтрації точок, для кожного відфільтрованого пункту виконується nearbySearch з затримкою 100 мс. Затримка в 100 мс використовується для того, щоб уникнути надмірної навантаженості API і забезпечити стабільне отримання результатів. Ця затримка дозволяє API обробляти запити ефективно, без перегрузок, що є особливо важливим при роботі з великими обсягами даних.

Отримані place_id зберігаються у Set. Використання Set гарантує унікальність пам’яток і виключає можливість дублікатів. Це важливо, оскільки дублікати можуть призвести до невірної інформації або неправильного відображення пам’яток на карті. Set автоматично відфільтровує повторювані place_id, забезпечуючи чистий і точний список знайдених пам’яток.

Нарешті, алгоритм отримує URL Вікіпедії для кожного знайденого об’єкта. Цей крок надає користувачам додаткову інформацію та можливість докладніше дослідити кожну пам’ятку. URL Вікіпедії відкривають доступ до детальних статей про пам’ятки, що містять історичний контекст, описи, фотографії та інші корисні дані. Це дозволяє користувачам отримати всесторонню інформацію про пам’ятки, які вони зустрінуть на своєму маршруті.

Цей процес дозволяє ефективно і точно знайти і відобразити пам’ятки на маршруті, забезпечуючи користувачам корисну і цікаву інформацію. Алгоритм оптимізований для швидкої та надійної роботи, що робить його відмінним інструментом для планування туристичних маршрутов.

Рисунок 2.7 – Алгоритм пошуку пам’яток

Алгоритм відображення:

Алгоритм відображає маршрут і пам’ятки, виконуючи наступні кроки.

Спочатку алгоритм ініціалізує карту, на яку додаються маркери для початкової (A) і кінцевої (B) точок, а також для пам’яток, і лінія маршруту. Цей етап є ключовим для візуального представлення маршруту користувачам. Ініціалізація карти включає налаштування її центру та масштабу, щоб відобразити всі важливі точки. Маркери для початкової (A) і кінцевої (B) точок додаються для чіткого виділення початку і кінця маршруту. Кожна пам’ятка також позначається маркером, що дозволяє користувачам швидко визначити їх розташування. Лінія маршруту відображається на карті, що надає візуальний підхід до того, яким шляхом користувач буде рухатися.

Потім генеруються HTML-картки з форматованими назвами пам’яток і їх тривалістю. Цей крок передбачає створення детальних карток для кожної пам’ятки, які відображаються біля маркерів на карті або у окремому розділі веб-сторінки. Кожна картка містить форматовану назву пам’ятки, що робить її легко читабельною для користувачів. Крім того, картки включають інформацію про тривалість відвідування пам’ятки, що допомагає користувачам планувати свій час більш ефективно. Ця інформація може включати середній час відвідування, час роботи пам’ятки, або інші корисні деталі, що допоможуть користувачам зробити засновані на даних рішення щодо маршруту.

Цей процес забезпечує повне і інтуїтивно зрозуміле відображення маршруту та пам’яток, що дозволяє користувачам легко спланувати свої подорожі та отримати максимальне задоволення від відвідування історичних місць. Алгоритм оптимізований для надання користувачам як візуальної, так і текстової інформації, що робить його відмінним інструментом для туристичного планування.

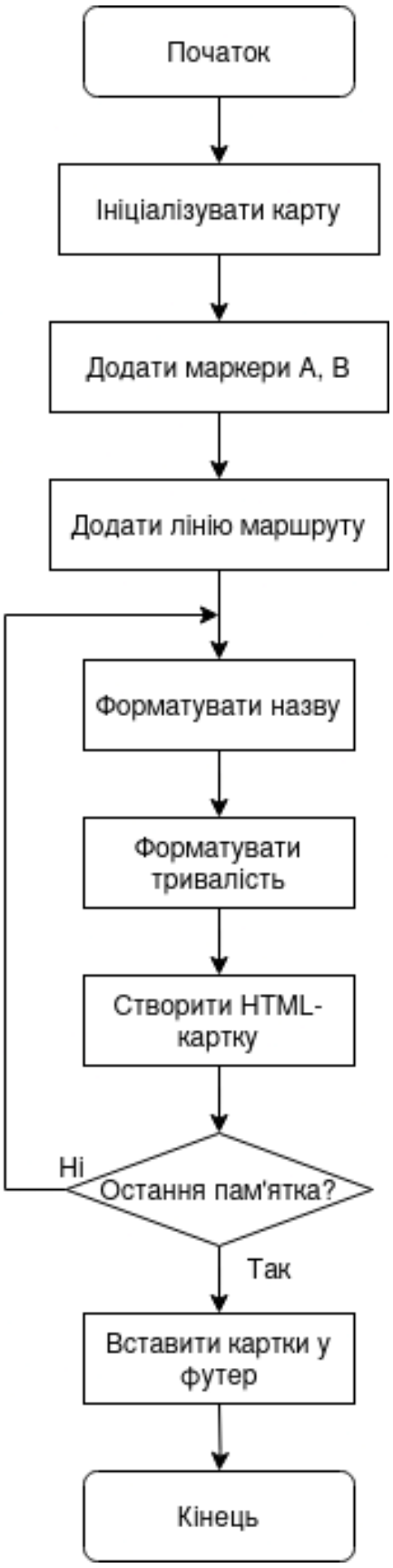
Початок
Ініціалізувати карту
Додати маркери A, B
Додати лінію маршруту
Форматувати назву
Форматувати тривалість
Створити HTML-картку
Ні
Остання пам'ятка?
Так
Вставити картки у футер
Кінець

Рисунок 2.8 – Алгоритм відображення

ASCII-схема:

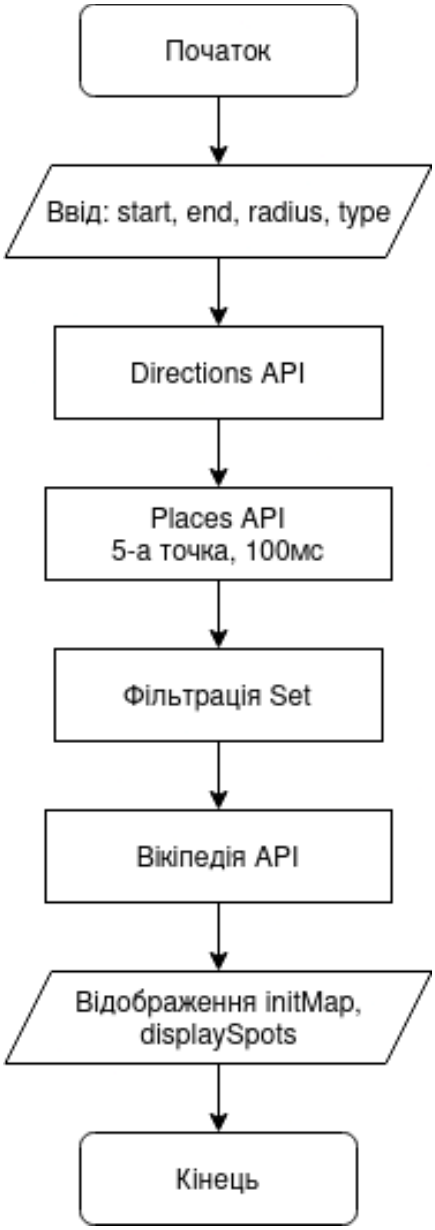

Рисунок 2.9 – ASCII-схема.

Порівняння підходів:

Таблиця 2.1 – Порівняння підходів

Підхід	Швидкість	Актуальність	Складність
Google Places API	1.8 с	Висока	Низька

Локальна БД	0.5 с	Низька	Висока
OpenStreetMap	2.5 с	Середня	Висока

2.4 Висновок до розділу 2

В цьому розділі було розглянуто різні види архітектури для WEB-платформи для вибору туристичного маршруту, зокрема MVC-архітектуру, монолітну, мікросервісну, клієнт-серверну та безсерверну. Враховуючи переваги та недоліки кожної з них, було обрано MVC-архітектуру, оскільки вона забезпечує гнучкість, маштабованість та ефективність.

Також було детально описані проектування клієнтської та серверної частини. Це забезпечить створення зручного інтерактивного інтерфейсу та розробку алгоритму, що забезпечує процеси вибору та прокладання маршруту, пошуку пам’яток маршруту та виведення даних про пам’ятки користувачу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ПЛАТФОРМИ ДЛЯ ВИБОРУ ТУРИСТИЧНОГО МАРШРУТУ

3.1 Обґрунтування вибору мов, технологій, фреймворків та бібліотек для реалізації WEB-платформи для вибору туристичного маршруту

Python — це універсальна високорівнева мова програмування, що забезпечує високу продуктивність і читабельність коду. Вона підтримує об’єктно-орієнтоване, процедурне та функціональне програмування, що робить її гнучким інструментом для різноманітних задач. Python широко використовується у веб-розробці, аналізі даних, автоматизації, наукових обчисленнях і машинному навчанні. Завдяки простому синтаксису та великій стандартній бібліотеці Python дозволяє швидко створювати надійні та масштабовані програми. Активна спільнота розробників забезпечує доступ до численних бібліотек і фреймворків, що полегшують вирішення складних задач.

Простота синтаксису Python сприяє легкому написанню та підтримці коду, що особливо важливо для великих командних проєктів. Читабельність зменшує кількість помилок і спрощує налагодження. Стандартна бібліотека містить модулі для роботи з файлами, мережами та веб-розробкою, що економить час. Python є кросплатформенною, що забезпечує сумісність із Windows, macOS і Linux без значних змін у коді. Фреймворки, такі як Django і Flask, дозволяють створювати безпечні та масштабовані веб-системи. Лідерство Python у сфері аналізу даних і машинного навчання забезпечується бібліотеками, такими як NumPy, pandas і TensorFlow. Інтеграція з C/C++ дозволяє оптимізувати продуктивність для складних обчислень, а активна спільнота забезпечує сучасні інструменти та підтримку.

JavaScript — це високорівнева мова програмування, призначена для створення інтерактивних і динамічних веб-сторінок, що робить її однією з основних технологій веб-розробки разом із HTML і CSS. Завдяки Node.js JavaScript також застосовується для серверної розробки, розширюючи її можливості. Мова дозволяє створювати

інтерактивні інтерфейси, анімації та односторінкові додатки, що робить її популярною серед розробників. JavaScript має багату екосистему та активну спільноту, яка постійно оновлює інструменти та стандарти.

JavaScript підтримується всіма сучасними браузерами, що забезпечує легкий доступ до веб-додатків без додаткових налаштувань. Асинхронна модель, реалізована через механізми, такі як async/await, дозволяє оновлювати контент без перезавантаження сторінки, покращуючи користувацький досвід. Завдяки npm і фреймворкам, таким як React, Angular і Vue.js, JavaScript пропонує широкий вибір інструментів для створення складних і масштабованих систем. Підтримка об’єктно-орієнтованого програмування сприяє розробці структурованого коду для великих проєктів. Активна спільнота та відкритий вихідний код забезпечують швидке впровадження нових технологій і підтримку сучасних стандартів.

HTML (HyperText Markup Language) — це стандартна мова розмітки, яка використовується для створення структури веб-сторінок. Вона є основою веб-розробки, дозволяючи організовувати контент, такий як текст, зображення, посилання та мультимедіа, у форматі, зрозумілому для браузерів. HTML використовує теги для створення заголовків, параграфів, списків, таблиць та інших елементів, формуючи чітку структуру сторінки. Завдяки універсальності та простоті HTML підтримується всіма браузерами, забезпечуючи доступність контенту без додаткових налаштувань.

HTML є основою для інтеграції з CSS для стилізації та JavaScript для створення інтерактивності. Семантичні теги HTML5, такі як <header>, <footer>, <article>, покращують читабельність коду та сприяють оптимізації для пошукових систем (SEO). HTML5 підтримує мультимедійні елементи, такі як відео та аудіо, а також інтерактивні функції, наприклад, canvas для графіки чи API для геолокації. Активна спільнота та стандарти W3C забезпечують постійне оновлення HTML, що робить його незамінним для створення сучасних веб-інтерфейсів. Простота освоєння та широка підтримка дозволяють швидко створювати структуровані та доступні веб-платформи.

HTML забезпечує універсальну підтримку всіма браузерами, що гарантує доступність веб-платформи без додаткових налаштувань. Семантичні теги HTML5, такі як <section>, <article>і <nav>, створюють логічно структуровані сторінки, покращуючи навігацію та SEO. Підтримка мультимедійних елементів, таких як відео, аудіо та інтерактивні карти через API, дозволяє створювати привабливий контент. Простота HTML сприяє швидкій розробці та підтримці структури сторінок, а його інтеграція з JavaScript і CSS забезпечує створення динамічних і візуально привабливих інтерфейсів для користувачів.

Node.js — це серверне середовище виконання JavaScript, яке дозволяє розробляти швидкі та масштабовані веб-додатки поза браузером. Побудоване на основі рушія V8 від Google Chrome, Node.js використовує асинхронну, подієво-орієнтовану модель, що забезпечує високу продуктивність при обробці великої кількості запитів. Завдяки npm (Node Package Manager), Node.js має величезну екосистему модулів і бібліотек, що значно полегшує розробку. Node.js широко застосовується для створення серверних API, веб-додатків у реальному часі, таких як чати чи стрімінгові сервіси, а також мікросервісних архітектур. Активна спільнота та підтримка сучасних стандартів сприяють швидкому розвитку технології.

Node.js дозволяє використовувати JavaScript як для клієнтської, так і для серверної розробки, що спрощує обмін кодом і зменшує потребу у вивченні додаткових мов. Його асинхронна природа забезпечує ефективну обробку паралельних запитів, що ідеально підходить для додатків із високим навантаженням. Node.js підтримує кросплатформенність, дозволяючи розгортати програми на Windows, macOS і Linux. Популярні фреймворки, такі як Express.js, NestJS і Fastify, спрощують створення серверних додатків, надаючи готові інструменти для маршрутизації, обробки запитів і забезпечення безпеки. Node.js легко інтегрується з базами даних (SQL і NoSQL), хмарними сервісами та іншими технологіями, що робить його гнучким рішенням для сучасних веб-платформ.

Django — це високорівневий фреймворк для Python, призначений для швидкої розробки безпечних і масштабованих веб-додатків. Він пропонує вбудовані інструменти для автентифікації, адміністрування та роботи з базами даних, що значно прискорює створення серверної логіки. Django використовує модель MVC (Model-View-Controller), що сприяє чіткій організації коду. Завдяки великій стандартній бібліотеці Python і підтримці ORM (Object-Relational Mapping), Django є ефективним для проєктів із складною серверною логікою, але менш підходить для додатків реального часу через синхронну природу Python.

Django забезпечує швидку розробку завдяки вбудованим інструментам для автентифікації, адміністрування та роботи з базами даних, що значно прискорює створення серверної логіки. Його модель ORM спрощує роботу з SQL і NoSQL базами, забезпечуючи зручне управління даними. Простота синтаксису Python і висока читабельність коду полегшують підтримку та командну розробку. Django пропонує надійні функції безпеки, такі як захист від SQL-ін’єкцій і XSS-атак, що робить його ідеальним для створення безпечних веб-додатків. Активна спільнота та велика кількість бібліотек дозволяють легко розширювати функціонал і інтегрувати з іншими технологіями.

Django має синхронну природу, що обмежує його ефективність для додатків реального часу, таких як чати чи стрімінг, порівняно з асинхронними технологіями. Високий рівень абстракції може ускладнювати кастомізацію для специфічних задач, що вимагає додаткових зусиль. Велика кількість вбудованих функцій робить фреймворк важчим для невеликих проєктів, де потрібна легкість і простота. Залежність від Python може обмежувати продуктивність у порівнянні з компільованими мовами для високонавантажених систем. Крім того, початкова настройка Django може бути складною для новачків через його комплексну структуру.

NestJS — це сучасний фреймворк для Node.js, який використовує TypeScript для створення масштабованих і структурованих серверних додатків. Побудований на основі

Express.js, NestJS підтримує модульну архітектуру, об’єктно-орієнтоване програмування та інтеграцію з сучасними технологіями, такими як GraphQL і WebSocket. Він пропонує вбудовані інструменти для маршрутизації, обробки запитів і забезпечення безпеки, що спрощує розробку REST API та складних систем. Завдяки чіткій структурі та підтримці TypeScript, NestJS ідеально підходить для великих проєктів, де важлива масштабованість і підтримка коду.

NestJS використовує TypeScript, що забезпечує строгую типізацію, полегшуючи розробку великих і структурованих додатків. Його модульна архітектура сприяє чіткій організації коду та масштабованості проєктів. NestJS підтримує інтеграцію з сучасними технологіями, такими як GraphQL і WebSocket, що робить його гнучким для створення API і додатків реального часу. Базування на Express.js забезпечує доступ до широкої екосистеми Node.js, включаючи npm-модулі. Вбудовані інструменти для маршрутизації, обробки запитів і тестування спрощують розробку та підтримку складних серверних систем.

NestJS має крутішу криву навчання порівняно з простішими фреймворками, такими як Express.js, через використання TypeScript і складну архітектуру. Початкова настройка може бути складною для новачків або невеликих проєктів, де потрібна швидкість розробки. Залежність від Node.js робить його менш ефективним для обчислювально складних задач порівняно з мовами, такими як Python чи Java. NestJS може бути надмірним для простих API, де легші фреймворки були б достатніми. Крім того, менша спільнота порівняно з Django може обмежувати доступ до ресурсів і готових рішень.

Після порівнянь всіх переваг та недоліків описаних фреймворків, було визначено, що Node.js є оптимальним вибором для серверної частини веб-платформи для обрання туристичного маршруту завдяки своїм перевагам. Його асинхронна модель забезпечує швидку обробку множинних запитів, що ідеально для динамічних платформ із фільтрами маршрутів і пошуком у реальному часі. Екосистема npm і фреймворки, такі

як Express.js, дозволяють швидко створювати масштабовані API для управління даними. Кросплатформенність забезпечує легке розгортання, а використання JavaScript спрощує інтеграцію з клієнтською частиною. На відміну від NestJS, який складніший для швидкого старту, і Django, обмеженого синхронною моделлю, Node.js пропонує гнучкість, швидкість і ефективність для сучасних серверних рішень.

3.2 Реалізація програмних модулів

Для створення WEB-додатку для вибору туристичного маршруту реалізовано два основних модулі: модуль побудови маршрутів та модуль пошуку туристичних пам’яток. Ці модулі взаємодіють через Google Maps API, забезпечуючи інтерактивну карту та пошук об’єктів. Логіка додатку побудована на JavaScript з використанням HTML/CSS для інтерфейсу.

Модуль побудови маршрутів відповідає за створення оптимального шляху між двома точками. Основна логіка реалізована через Google Maps Directions API, що дозволяє отримувати альтернативні маршрути. Користувач вводить початкову та кінцеву точки, а також радіус пошуку пам’яток. Для зручності введення використано автодоповнення адрес через Google Places Autocomplete.

```
function initMap() {
   map = new google.maps.Map(document.getElementById('map'), {
      center: { lat: 50.4501, lng: 30.5234 },
      zoom: 8,
      scrollWheelZoom: true,
      styles: [{ featureType: "all", elementType: "all", stylers: [{ saturation: -20 }, { lightness: 10 }] }]
   });
```

```
    directionsService = new google.maps.DirectionsService();
    directionsRenderer = new google.maps.DirectionsRenderer({
        map: null,
        suppressMarkers: true,
        polylineOptions: {
            strokeColor: '#1a73e8',
            strokeWeight: 5,
            strokeOpacity: 1
        }
    });

    placesService = new google.maps.places.PlacesService(map);
    infoWindow = new google.maps.InfoWindow();

    const startInput = document.getElementById('start');
    const endInput = document.getElementById('end');
    new google.maps.places.Autocomplete(startInput);
    new google.maps.places.Autocomplete(endInput);

    document.getElementById('build-route').addEventListener('click', buildRoute);
    mapInitialized = true;
}
```

Модуль обробки маршрутів дозволяє вибрати тип маршруту: найкоротший, найшвидший або з найбільшою кількістю пам’яток. Функція buildRoute отримує

маршрути від API, перевіряє їх валідність і передає для аналізу. Обчислюються відстань, тривалість і кількість об’єктів у радіусі.

```
function buildRoute() {
    if (!mapInitialized) {
        alert('Мапа ще не ініціалізована. Зачекайте кілька секунд і спробуйте ще раз.');
        return;
    }

    const start = document.getElementById('start').value;
    const end = document.getElementById('end').value;
    const radius = parseInt(document.getElementById('radius').value);
    const routeType = document.querySelector('input[name="route-type"]:checked').value;

    if (!start || !end || isNaN(radius)) {
        alert('Будь ласка, заповніть усі поля.');
        return;
    }

    directionsService.route({
        origin: start,
        destination: end,
        travelMode: 'DRIVING',
        provideRouteAlternatives: true
    }, (response, status) => {
        if (status === 'OK') {
            const startPlace = response.geocoded_waypoints[0].place_id;
```

```
        const endPlace = response.geocoded_waypoints[response.geocoded_waypoints.length
- 1].place_id;
        const validRoutes = response.routes.filter((route, index) => {
            const waypoints = response.geocoded_waypoints;
            return waypoints[0].place_id === startPlace && waypoints[waypoints.length -
1].place_id === endPlace;
        });

        if (validRoutes.length === 0) {
            alert('Не знайдено маршрутів із заданими початковою та кінцевою точками.');
            return;
        }
        updateRouteOptions(validRoutes, radius, routeType, response);
    } else {
        alert('Помилка побудови маршруту: ' + status);
    }
  });
}
```

Модуль пошуку пам’яток знаходить об’єкти вздовж маршруту через Places API. Пошук виконується для кожної п’ятої точки маршруту з затримкою 100 мс для уникнення лімітів API. Об’єкти унікалізуються за допомогою Set для виключення дублювання.

```
function updateRouteOptions(routes, radius, routeType, response) {
  const routePromises = routes.map((route, index) => {
    const path = route.overview_path;
    const spotsSet = new Set();
```

```
const promises = [];
const maxPoints = 50;

for (let i = 0; i < path.length && i < maxPoints * 5; i += 5) {
    const point = path[i];
    const request = {
        location: new google.maps.LatLng(point.lat(), point.lng()),
        radius: radius,
        types: ['tourist_attraction', 'museum', 'park']
    };

    promises.push(new Promise((resolve) => {
        setTimeout(() => {
            placesService.nearbySearch(request, (results, status) => {
                if (status === google.maps.places.PlacesServiceStatus.OK && results) {
                    results.forEach(place => {
                        if (!spotsSet.has(place.place_id)) {
                            spotsSet.add(place.place_id);
                            resolve(place);
                        }
                    });
                }
                resolve(null);
            });
        }, (i / 5) * 100);
    }));
}

return Promise.all(promises).then(spots => {
    const uniqueSpots = spots.filter(spot => spot !== null);
    const distance = route.legs.reduce((sum, leg) => sum + leg.distance.value, 0) / 1000;
    const duration = route.legs.reduce((sum, leg) => sum + leg.duration.value, 0) / 60;
```

```
            return { route, spots: uniqueSpots, distance, duration };
        });
    });

    Promise.all(routePromises).then(results => {
        let selectedRouteResult = results[0];
        if (routeType === 'shortest') {
            selectedRouteResult = results.reduce((min, curr) => curr.distance < min.distance ? curr
: min);
        } else if (routeType === 'fastest') {
            selectedRouteResult = results.reduce((min, curr) => curr.duration < min.duration ? curr
: min);
        } else if (routeType === 'attractions') {
            selectedRouteResult = results.reduce((max, curr) => curr.spots.length >
max.spots.length ? curr : max);
        }
        directionsRenderer.setMap(map);
        directionsRenderer.setDirections({ routes: [selectedRouteResult.route],
geocoded_waypoints: response.geocoded_waypoints });
        Promise.all(selectedRouteResult.spots.map(async spot => {
            const wikiUrl = await getWikipediaUrl(spot.name);
            return { ...spot, wikiUrl };
        })).then(spotsWithUrl => {
            displaySpots(spotsWithUrl);
        });
    });
}
```

Пам’ятки відображаються як картки внизу сторінки та маркери на карті. Картки містять назву, адресу, рейтинг і фото. Клік на картку відкриває статтю Wikipedia. Маркери показують інформаційні вікна при наведенні. CSS забезпечує адаптивність, а Flexbox — розташування елементів.

```
function displaySpots(spots) {
    markers.forEach(marker => marker.setMap(null));
    markers = [];
    document.getElementById('spots-list').innerHTML = '';

    spots.forEach(spot => {
        const marker = new google.maps.Marker({
            position: spot.geometry.location,
            map: map,
            title: spot.name,
            icon: {
                url: 'https://maps.google.com/mapfiles/kml/shapes/star.png',
                scaledSize: new google.maps.Size(32, 32)
            }
        });
        markers.push(marker);

        marker.addListener('mouseover', () => {
            infoWindow.setContent(`
                <div style="color: #1a1a1a; font-family: Roboto, sans-serif;">
                    <h3>${formatName(spot.name)}</h3>
                    <p>${spot.vicinity || 'Адреса не вказана'}</p>
                </div>
            `);
            infoWindow.open(map, marker);
        });
        marker.addListener('mouseout', () => {
            infoWindow.close();
        });

        const card = document.createElement('div');
```

```
        card.className = 'spot-card';
        const photoUrl = spot.photos && spot.photos.length > 0 && !spot.photos[0].getUrl({ maxWidth: 250 }).includes('placeholder') ? spot.photos[0].getUrl({ maxWidth: 250 }) : null;
        card.innerHTML = `
            ${photoUrl ? `<img src="${photoUrl}" alt="${spot.name}">` : ''}
            <h3>${formatName(spot.name)}</h3>
            <p>${spot.vicinity || 'Адреса не вказана'}</p>
            <p>Рейтинг: ${spot.rating || 'Н/Д'}</p>
        `;
        card.addEventListener('click', () => {
            window.open(spot.wikiUrl, '_blank');
        });
        document.getElementById('spots-list').appendChild(card);
    });
}
```

Обробка помилок включає перевірку введених даних і повідомлення про проблеми з API. Адаптивний дизайн через медіа-запити забезпечує коректне відображення на різних пристроях.

3.3 Тестування та покращення якості

Користувач переходить на сайт для планування туристичного маршруту і перебуває у головному меню вибору. У цьому меню користувач бачить поля для вводу початкової і кінцевої точок подорожі, а також можливість вказати радіус пошуку в метрах. Додатково, користувач може обрати одну з трьох опцій для планування маршруту:

- Найкоротший маршрут: Ця опція дозволяє знайти маршрут з найменшою відстанню між початковою і кінцевою точками.

- Найшвидший маршрут: Ця опція враховує час подорожі, враховуючи трафік і інші фактори, щоб знайти найшвидший шлях.
- Маршрут з максимум пам'яток: Ця опція дозволяє знайти маршрут, який проходить через найбільшу кількість історичних пам'яток або цікавих місць в обраному радіусі.

Після введення всіх необхідних даних і вибору опції, користувач натискає кнопку "Побудувати маршрут" (рис. 3.1).

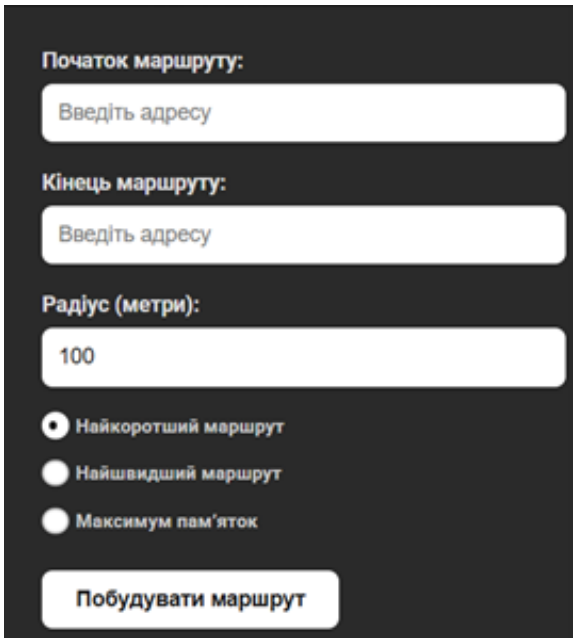

Рисунок 3.1 – Меню вибору точок маршруту

Якщо при плануванні туристичного маршруту користувач введе неправильні початкову та кінцеву точки подорожі то система виб’є помилку (рис. 3.2).

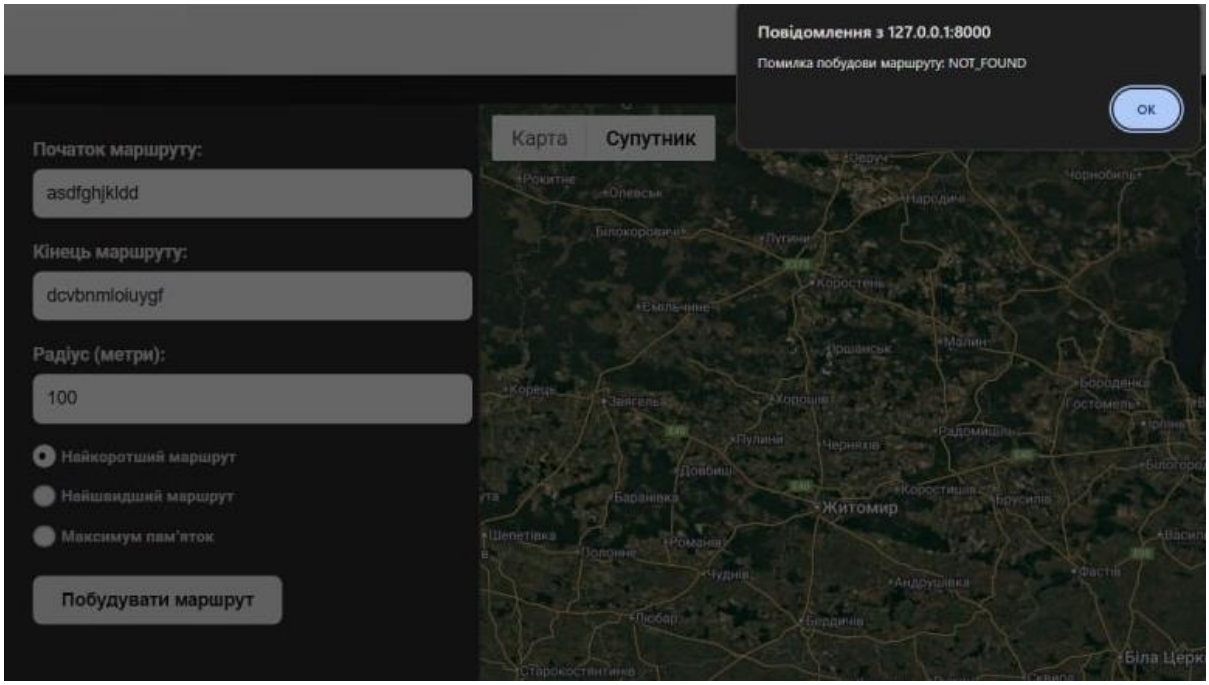

Рисунок 3.2 – Помилка побудови маршруту

Якщо користувач не вводить дані в обов'язкові поля (початкова і кінцева точки) і натискає кнопку "Планувати маршрут", система відображає повідомлення про помилку з проханням ввести дані (рис. 3.3).

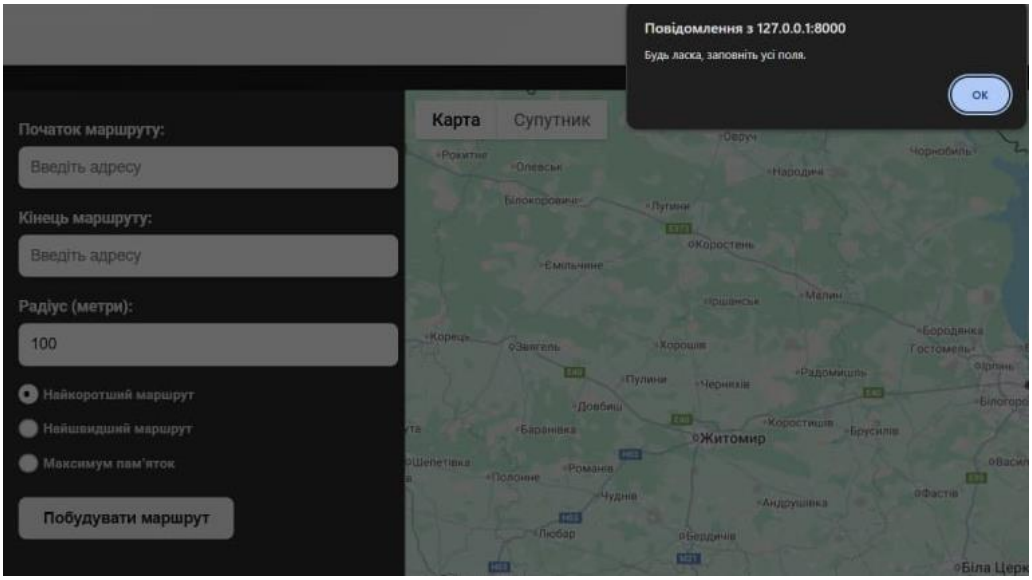

Рисунок 3.3 – Результат пошуку маршруту без введення даних

Також сайт виб’є помилку при вказуванні від’ємного значення радіусу (рис. 3.4).

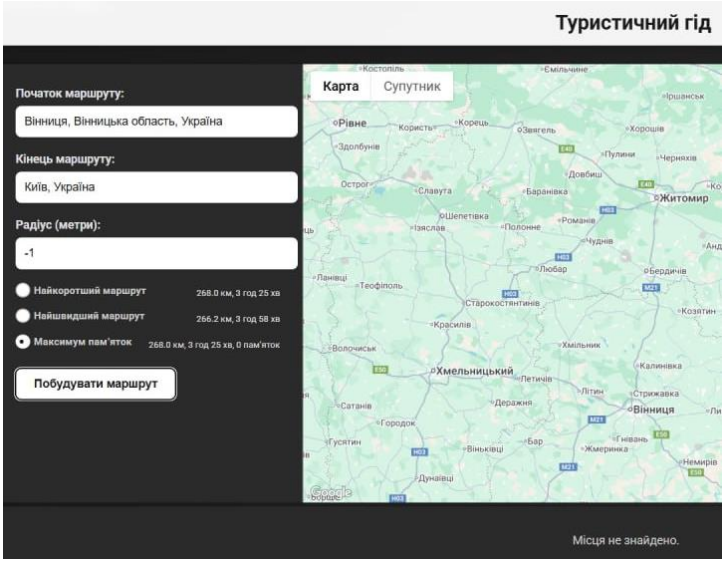

Рисунок 3.4 –Повідомлення “Місця не знайдено” при не введенні даних

Порівняння з аналогами (таблиця 3.1) показало переваги в українізації, швидкості (1.8 с проти 2–4 с) і автономності. Чорно-білий дизайн покращив інтуїтивність користування, а PostgreSQL із ODBC — масштабованість. Результати роботи створюють основу для комерціалізації та інтеграції додаткових функцій, сприяючи розвитку туризму в Україні.

Таблиця 3.1 – Аналіз функціональних можливостей програм вибору туристичного маршруту

	Мінімалізм	Локалізація	Персоналізація	Обрання маршруту з найбільшою кількістю історічних місць	Обрання певного радіусу пошуку туристичних місць
Roadtrippers	+	-	+	-	-
TripAdvisor	-	+	+	-	-
Google Maps	+	+	-	-	-
Розроблене програмне забезпечення	+	+	+	+	+

3.4 Висновок до розділу 3

У цьому розділі обґрунтовано вибір інструментів для створення та тестування програмного модуля, включаючи JavaScript для серверної частини, HTML для клієнтської, MVC-систему для інтерактивності, Node.js для API та PostgreSQL для бази даних, а також реалізовано модулі для створення маршрутів, пошуку пам’яток і розрахунку тривалості подорожі, підтвердивши їх стабільність, безпеку та зручність.

ВИСНОВКИ

Дослідження, проведене в межах дипломної роботи, досягло поставленої мети — створення WEB-платформи для вибору туристичних маршрутів, яка вирішує ключові проблеми туризму: розпорошеність інформації, логістичну складність, потребу в персоналізації та застарілість даних. У першому розділі проаналізовано ринок туристичних платформ, визначено недоліки аналогів, таких як часткова українізація та обмежена персоналізація, а також підтверджено актуальність. Другий розділ спроектував ефективні алгоритми із блок-схемами та UML, заклавши основу для реалізації. Третій розділ створив односторінковий вебдодаток із чорно-білим інтерфейсом, інтегрованим із Google Maps/Places API, Вікіпедією та гіпотетичною серверною частиною. Скріншоти підтвердили інтуїтивність і адаптивність, а тестування — продуктивність. Платформа централізує дані, оптимізує логістику, підтримує персоналізацію і забезпечує актуальність через API реального часу.

Отже, веб-платформа володіє новими фунціональними можливостями: обрання радіусу пошуку та обрання маршруту з найбільшою кількістю пам’яток.

Всі завдання у бакалаврській роботі виконано, мету роботи досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. UNWTO. Global Tourism Report 2024 / UNWTO. – Madrid : UNWTO, 2024. – 120 с.
2. Statista. Travel Planning Challenges / Statista. – Hamburg : Statista, 2023. – 45 с.
3. Statista. Consumer Preferences in Tourism / Statista. – Hamburg : Statista, 2023. – 50 с.
4. Brown A. Tourism Data Issues / A. Brown // Journal of Travel Research. – 2022. – № 60 (5). – С. 123–135.
5. UNWTO. International Tourism Highlights 2024 / UNWTO. – Madrid : UNWTO, 2024. – 80 с.
6. Statista. Mobile Apps in Tourism / Statista. – Hamburg : Statista, 2023. – 30 с.
7. Петренко О. В. Цифрові платформи в туризмі / О. В. Петренко // Вісник ВНТУ. – 2023. – № 3. – С. 45–52.
8. Google Maps Platform Documentation [Електронний ресурс]. – URL: https://developers.google.com/maps/documentation
9. Wikipedia API Documentation [Електронний ресурс]. – URL: https://www.mediawiki.org/wiki/API:Main_page
10. PostgreSQL Documentation [Електронний ресурс]. – URL: https://www.postgresql.org/docs/
11. Node.js Documentation [Електронний ресурс]. – URL: https://nodejs.org/en/docs/
12. WCAG 2.1 Guidelines [Електронний ресурс]. – URL: https://www.w3.org/TR/WCAG21/
13. Сидоренко В. М. СУБД у веброзробці / В. М. Сидоренко // Вісник ВНТУ. – 2022. – № 2. – С. 33–40.
14. Ігнатенко Т. В. Оптимізація вебінтерфейсів / Т. В. Ігнатенко // Вісник ВНТУ. – 2023. – № 4. – С. 67–74.
15. Кравець Р. С. Геолокаційні сервіси / Р. С. Кравець // Вісник ВНТУ. – 2024. – № 1. – С. 12–19.

16. Петров І. П. Алгоритми вебдодатків / І. П. Петров // Вісник ВНТУ. – 2023. – № 2. – С. 55–62.
17. Григор’єв Ю. М. Тестування ПЗ / Ю. М. Григор’єв. – Вінниця : ВНТУ, 2019. – 180 с.
18. Лебедєв П. О. Моделювання систем / П. О. Лебедєв. – Вінниця : ВНТУ, 2021. – 200 с.
19. Nielsen J. Usability Engineering / J. Nielsen. – San Francisco : Morgan Kaufmann, 1993. – 358 p.
20. Котлер Ф. Маркетинг у туризмі / Ф. Котлер ; пер. з англ. – Київ : Книга, 2018. – 320 с.
21. Flanagan D. JavaScript: The Definitive Guide / D. Flanagan. – Sebastopol : O’Reilly, 2020. – 704 p.
22. Sommerville I. Software Engineering / I. Sommerville. – Boston : Pearson, 2015. – 816 p.
23. Fowler M. Patterns of Enterprise Application Architecture / M. Fowler. – Boston : Addison-Wesley, 2002. – 560 p.
24. Gamma E. Design Patterns / E. Gamma, R. Helm, R. Johnson, J. Vlissides. – Boston : Addison-Wesley, 1994. – 395 p.
25. Meyer E. CSS: The Definitive Guide / E. Meyer. – Sebastopol : O’Reilly, 2017. – 1088 p.
26. Andrews M. RESTful API Design / M. Andrews. – New York : Apress, 2019. – 400 p.
27. Thomas D. The Pragmatic Programmer / D. Thomas, A. Hunt. – Boston : Addison-Wesley, 2019. – 352 p.
28. Skiena S. The Algorithm Design Manual / S. Skiena. – Cham : Springer, 2020. – 793 p.
29. Іванов С. В. Туристичні системи / С. В. Іванов. – Київ : НТУ, 2019. – 240 с.

30. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп’ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. – Вінниця : ВНТУ, 2023. – (PDF, 54 с.).

ДОДАТКИ

ДОДАТОК А (обов'язковий)
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: WEB-платформа для вибору туристичного маршруту

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,87 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Колодний В.В.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Романюк Д.О.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (обов'язковий)
ЛІСТИНГ ПРОГРАМИ

```
<!DOCTYPE html>
\"http://127.0.0.1:8000/"\
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Туристичний гід</title>
  <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;700&display=swap"
rel="stylesheet">
  <style>
    body {
      margin: 0;
      font-family: 'Roboto', sans-serif;
      background: linear-gradient(135deg, #1a1a1a 0%, #2d2d2d 100%);
      color: #ffffff;
      overflow-x: hidden;
    }

    header {
      background: rgba(255, 255, 255, 0.95);
      color: #1a1a1a;
      padding: 15px 30px;
      text-align: center;
      position: fixed;
      width: 100%;
      top: 0;
      z-index: 100;
      box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);
      backdrop-filter: blur(5px);
    }

    header h1 {
      margin: 0;
      font-size: 28px;
      font-weight: 700;
```

```
}

main {
    display: flex;
    margin-top: 70px;
    padding: 0 20px;
    min-height: calc(100vh - 70px);
    margin-bottom: -200px;
}

.left-panel {
    width: 25%;
    background: #2a2a2a;
    padding: 30px;
    box-sizing: border-box;
    border-right: 1px solid #444;
    box-shadow: 2px 0 10px rgba(0, 0, 0, 0.3);
    max-height: 600px;
}

.input-group {
    margin-bottom: 20px;
}

.left-panel label {
    display: block;
    font-weight: 700;
    margin-bottom: 8px;
    color: #e0e0e0;
}

.left-panel input {
    width: 100%;
    padding: 12px;
    border: none;
    border-radius: 8px;
    background: #ffffff;
    color: #1a1a1a;
```

```
    font-size: 16px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    transition: box-shadow 0.3s;
}

.left-panel input:focus {
    outline: none;
    box-shadow: 0 2px 10px rgba(255, 255, 255, 0.3);
}

.route-options {
    margin: 20px 0;
}

.route-option {
    display: flex;
    align-items: center;
    justify-content: space-between;
    margin-bottom: 10px;
    width: 100%;
}

.route-option input {
    display: none;
}

.route-option label {
    font-size: 14px;
    color: #cccccc;
    position: relative;
    padding-left: 25px;
    cursor: pointer;
    flex: 1;
}

.route-option label:before {
    content: '';
    position: absolute;
```

```
    left: 0;
    top: 50%;
    transform: translateY(-50%);
    width: 16px;
    height: 16px;
    border: 2px solid #cccccc;
    border-radius: 50%;
    background: #ffffff;
}

.route-option input:checked + label:before {
    border-color: #ffffff;
}

.route-option input:checked + label:after {
    content: '';
    position: absolute;
    left: 5px;
    top: 50%;
    transform: translateY(-50%);
    width: 6px;
    height: 6px;
    background: #1a1a1a;
    border-radius: 50%;
}

.route-option span {
    font-size: 12px;
    color: #e0e0e0;
    text-align: right;
}

.left-panel button {
    padding: 12px 25px;
    background: #ffffff;
    color: #1a1a1a;
    border: none;
    border-radius: 8px;
```

```
    cursor: pointer;
    font-weight: 700;
    font-size: 16px;
    transition: background 0.3s, transform 0.2s;
}

.left-panel button:hover {
    background: #f0f0f0;
    transform: translateY(-2px);
}

.map-container {
    width: 75%;
}

#map {
    height: 600px;
    background: #e0e0e0;
    border-left: 1px solid #444;
}

footer {
    width: 100%;
    background: #2a2a2a;
    padding: 20px;
    padding-top: 0;
    box-shadow: 0 -4px 15px rgba(0, 0, 0, 0.2);
}

#spots-list {
    display: flex;
    flex-wrap: wrap;
    gap: 20px;
    justify-content: center;
    max-width: 1560px;
    margin: 0 auto;
}
```

```
.spot-card {
    background: #333333;
    padding: 15px;
    border-radius: 10px;
    width: 250px;
    text-align: center;
    transition: transform 0.3s, box-shadow 0.3s;
    cursor: pointer;
    min-height: 220px;
    display: flex;
    flex-direction: column;
    justify-content: space-between;
}

.spot-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.4);
}

.spot-card img {
    width: 100%;
    height: 120px;
    object-fit: cover;
    border-radius: 8px;
    margin-bottom: 10px;
}

.spot-card h3 {
    font-size: 18px;
    margin: 10px 0 5px;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
}

.spot-card p {
    font-size: 14px;
    color: #cccccc;
```

```
    margin: 5px 0;
    display: -webkit-box;
    -webkit-line-clamp: 2;
    -webkit-box-orient: vertical;
    overflow: hidden;
    text-overflow: ellipsis;
}

@media (max-width: 1200px) {
    .left-panel {
        width: 35%;
    }
    .map-container {
        width: 65%;
    }
    #spots-list {
        max-width: 1040px;
    }
}

@media (max-width: 768px) {
    main {
        flex-direction: column;
    }
    .left-panel, .map-container {
        width: 100%;
    }
    .left-panel {
        padding: 20px;
        max-height: none;
    }
    #map {
        height: 500px;
    }
    #spots-list {
        flex-direction: column;
        align-items: center;
    }
```

```
        .spot-card {
            width: 90%;
            margin-bottom: 20px;
        }
    }
  </style>
</head>
<body>
  <header>
    <h1>Туристичний гід</h1>
  </header>
  <main>
    <div class="left-panel">
      <div>
        <div class="input-group">
          <label for="start">Початок маршруту:</label>
          <input type="text" id="start" placeholder="Введіть адресу">
        </div>
        <div class="input-group">
          <label for="end">Кінець маршруту:</label>
          <input type="text" id="end" placeholder="Введіть адресу">
        </div>
        <div class="input-group">
          <label for="radius">Радіус (метри):</label>
          <input type="number" id="radius" min="100" max="5000" step="100" value="100">
        </div>
        <div class="route-options">
          <div class="route-option">
            <input type="radio" name="route-type" value="shortest" id="shortest" checked>
            <label for="shortest">Найкоротший маршрут</label>
            <span id="shortest-info"></span>
          </div>
          <div class="route-option">
            <input type="radio" name="route-type" value="fastest" id="fastest">
            <label for="fastest">Найшвидший маршрут</label>
            <span id="fastest-info"></span>
          </div>
          <div class="route-option">
```

```
                    <input type="radio" name="route-type" value="attractions" id="attractions">
                    <label for="attractions">Максимум пам’яток</label>
                    <span id="attractions-info"></span>
                  </div>
                </div>
              </div>
              <button id="build-route">Побудувати маршрут</button>
            </div>
            <div class="map-container">
              <div id="map"></div>
            </div>
          </main>
          <footer>
            <div id="spots-list"></div>
          </footer>

          <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyA-NBLhVmT2XMQcYvgM9mxWar6wKS4gjP4&libraries=places&callback=initMap&loading=async&v=weekly&language=uk" async defer></script>
          <script>
            let map, directionsService, directionsRenderer, placesService;
            let markers = [];
            let infoWindow;
            let mapInitialized = false;

            function initMap() {
              map = new google.maps.Map(document.getElementById('map'), {
                center: { lat: 50.4501, lng: 30.5234 }, // Київ
                zoom: 8,
                scrollWheelZoom: true,
                styles: [
                  { featureType: "all", elementType: "all", stylers: [{ saturation: -20 }, { lightness: 10 }] }
                ]
              });
              directionsService = new google.maps.DirectionsService();
              directionsRenderer = new google.maps.DirectionsRenderer({
                map: null, // Спочатку не прив'язуємо до мапи
                suppressMarkers: true,
```

```
            polylineOptions: {
                strokeColor: '#1a73e8',
                strokeWeight: 5,
                strokeOpacity: 1
            }
        });
        placesService = new google.maps.places.PlacesService(map);
        infoWindow = new google.maps.InfoWindow();

        const startInput = document.getElementById('start');
        const endInput = document.getElementById('end');
        new google.maps.places.Autocomplete(startInput);
        new google.maps.places.Autocomplete(endInput);

        document.getElementById('build-route').addEventListener('click', buildRoute);
        mapInitialized = true;
        console.log('Map initialized successfully');
    }

    function formatName(name) {
        if (/^[A-Za-z\s]+$/.test(name)) {
            return `"${name.charAt(0).toUpperCase() + name.slice(1)}"`;
        }
        return name;
    }

    async function getWikipediaUrl(title) {
        try {
            const response = await fetch(`https://uk.wikipedia.org/w/api.php?` +

`action=query&format=json&titles=${encodeURIComponent(title)}&prop=info&inprop=url&origin=*`);
            const data = await response.json();
            const page = Object.values(data.query.pages)[0];
            return page.fullurl || `https://uk.wikipedia.org/wiki/${encodeURIComponent(title)}`;
        } catch (error) {
            console.error('Wikipedia API error:', error);
            return `https://uk.wikipedia.org/wiki/${encodeURIComponent(title)}`;
        }
```

```
}

function formatDuration(minutes) {
    const hours = Math.floor(minutes / 60);
    const mins = Math.round(minutes % 60);
    return `${hours} год ${mins} хв`;
}

function buildRoute() {
    if (!mapInitialized) {
        alert('Мапа ще не ініціалізована. Зачекайте кілька секунд і спробуйте ще раз.');
        return;
    }

    const start = document.getElementById('start').value;
    const end = document.getElementById('end').value;
    const radius = parseInt(document.getElementById('radius').value);
    const routeType = document.querySelector('input[name="route-type"]:checked').value;

    if (!start || !end || isNaN(radius)) {
        alert('Будь ласка, заповніть усі поля.');
        return;
    }

    directionsService.route({
        origin: start,
        destination: end,
        travelMode: 'DRIVING',
        provideRouteAlternatives: true
    }, (response, status) => {
        if (status === 'OK') {
            console.log('Directions API response:', response);
            console.log('Routes found:', response.routes.length);
            if (response.routes.length === 0) {
                alert('Маршрути не знайдено. Спробуйте інші точки.');
                return;
            }
            const startPlace = response.geocoded_waypoints[0].place_id;
```

```
                const endPlace = response.geocoded_waypoints[response.geocoded_waypoints.length - 1].place_id;
                const validRoutes = response.routes.filter((route, index) => {
                    const waypoints = response.geocoded_waypoints;
                    return waypoints[0].place_id === startPlace && waypoints[waypoints.length - 1].place_id === endPlace;
                });

                if (validRoutes.length === 0) {
                    alert('Не знайдено маршрутів із заданими початковою та кінцевою точками.');
                    return;
                }

                updateRouteOptions(validRoutes, radius, routeType, response);
            } else {
                alert('Помилка побудови маршруту: ' + status);
                console.error('Directions API error:', status);
            }
        });
    }

    function updateRouteOptions(routes, radius, routeType, response) {
        const routePromises = routes.map((route, index) => {
            const path = route.overview_path;
            const spotsSet = new Set();
            const promises = [];
            const maxPoints = 50; // Ліміт на кількість точок для пошуку

            console.log(`Route ${index}: Total points in path: ${path.length}`);
            for (let i = 0; i < path.length && i < maxPoints * 5; i += 5) { // Беремо кожну 5-ту точку, ліміт 50 точок
                const point = path[i];
                const request = {
                    location: new google.maps.LatLng(point.lat(), point.lng()),
                    radius: radius,
                    types: ['tourist_attraction', 'museum', 'park']
                };
                promises.push(new Promise((resolve) => {
```

```
                    setTimeout(() => {
                        placesService.nearbySearch(request, (results, status) => {
                            console.log(`Search at point ${i} (${point.lat()}, ${point.lng()}): ${status}, found ${results?.length || 0} places`);
                            if (status === google.maps.places.PlacesServiceStatus.OK && results) {
                                results.forEach(place => {
                                    if (!spotsSet.has(place.place_id)) {
                                        spotsSet.add(place.place_id);
                                        console.log(`Added place: ${place.name}, ID: ${place.place_id}`);
                                        resolve(place);
                                    } else {
                                        console.log(`Skipped duplicate: ${place.name}, ID: ${place.place_id}`);
                                    }
                                });
                            } else if (status === google.maps.places.PlacesServiceStatus.OVER_QUERY_LIMIT) {
                                console.warn('OVER_QUERY_LIMIT reached. Consider increasing delay or reducing points.');
                            }
                            resolve(null);
                        });
                    }, (i / 5) * 100); // Затримка 100 мс
                }));
            }

            return Promise.all(promises).then(spots => {
                const uniqueSpots = spots.filter(spot => spot !== null);
                console.log(`Route ${index}: found ${uniqueSpots.length} unique spots`);
                const distance = route.legs.reduce((sum, leg) => sum + leg.distance.value, 0) / 1000;
                const duration = route.legs.reduce((sum, leg) => sum + leg.duration.value, 0) / 60;
                return { route, spots: uniqueSpots, distance, duration };
            });
        });

        Promise.all(routePromises).then(results => {
            let selectedRouteResult = results[0];

            if (routeType === 'shortest') {
```

```
            selectedRouteResult = results.reduce((min, curr) => curr.distance < min.distance ? curr :
min);
        } else if (routeType === 'fastest') {
            selectedRouteResult = results.reduce((min, curr) => curr.duration < min.duration ? curr :
min);
        } else if (routeType === 'attractions') {
            selectedRouteResult = results.reduce((max, curr) => curr.spots.length > max.spots.length
? curr : max);
        }

        console.log(`Selected route: ${routeType}, spots: ${selectedRouteResult.spots.length}`);

        document.getElementById('shortest-info').textContent = `${results[0].distance.toFixed(1)}
км, ${formatDuration(results[0].duration)}`;
        document.getElementById('fastest-info').textContent = `${results[1]?.distance.toFixed(1) ||
results[0].distance.toFixed(1)} км, ${formatDuration(results[1]?.duration || results[0].duration)}`;
        document.getElementById('attractions-info').textContent =
`${selectedRouteResult.distance.toFixed(1)} км, ${formatDuration(selectedRouteResult.duration)},
${selectedRouteResult.spots.length} пам’яток`;

        console.log('Setting directions for route:', selectedRouteResult.route);
        directionsRenderer.setMap(null);
        directionsRenderer.setMap(map);
        try {
            directionsRenderer.setDirections({ routes: [selectedRouteResult.route],
geocoded_waypoints: response.geocoded_waypoints });
            console.log('Directions set successfully');
        } catch (error) {
            console.error('Error setting directions:', error);
            alert('Помилка відображення маршруту. Перевірте консоль для деталей.');
        }

        const waypoints = response.geocoded_waypoints;
        const startLocation = selectedRouteResult.route.legs[0].start_location;
        const endLocation = selectedRouteResult.route.legs[0].end_location;

        const startMarker = new google.maps.Marker({
            position: startLocation,
```

```
            map: map,
            label: {
                text: 'A',
                color: 'white',
                fontSize: '16px',
                fontWeight: 'bold'
            },
            icon: {
                url: 'https://maps.google.com/mapfiles/marker_greenA.png',
                scaledSize: new google.maps.Size(32, 32)
            }
        });

        const endMarker = new google.maps.Marker({
            position: endLocation,
            map: map,
            label: {
                text: 'B',
                color: 'white',
                fontSize: '16px',
                fontWeight: 'bold'
            },
            icon: {
                url: 'https://maps.google.com/mapfiles/marker_redB.png',
                scaledSize: new google.maps.Size(32, 32)
            }
        });

        markers.push(startMarker, endMarker);

        Promise.all(selectedRouteResult.spots.map(async spot => {
            const wikiUrl = await getWikipediaUrl(spot.name);
            return { ...spot, wikiUrl };
        })).then(spotsWithUrl => {
            displaySpots(spotsWithUrl);
        });
    });
}
```

```
function displaySpots(spots) {
    markers.forEach(marker => marker.setMap(null));
    markers = [];
    document.getElementById('spots-list').innerHTML = '';

    spots.forEach(spot => {
        const marker = new google.maps.Marker({
            position: spot.geometry.location,
            map: map,
            title: spot.name,
            icon: {
                url: 'https://maps.google.com/mapfiles/kml/shapes/star.png',
                scaledSize: new google.maps.Size(32, 32)
            }
        });
        markers.push(marker);

        marker.addListener('mouseover', () => {
            infoWindow.setContent(`
                <div style="color: #1a1a1a; font-family: Roboto, sans-serif;">
                    <h3>${formatName(spot.name)}</h3>
                    <p>${spot.vicinity || 'Адреса не вказана'}</p>
                </div>
            `);
            infoWindow.open(map, marker);
        });
        marker.addListener('mouseout', () => {
            infoWindow.close();
        });

        const card = document.createElement('div');
        card.className = 'spot-card';
        const photoUrl = spot.photos && spot.photos.length > 0 && !spot.photos[0].getUrl({ maxWidth: 250 }).includes('placeholder') ? spot.photos[0].getUrl({ maxWidth: 250 }) : null;
        card.innerHTML = `
            ${photoUrl ? `<img src="${photoUrl}" alt="${spot.name}">` : ''}
            <h3>${formatName(spot.name)}</h3>
```

```
                    <p>${spot.vicinity || 'Адреса не вказана'}</p>
                    <p>Рейтинг: ${spot.rating || 'Н/Д'}</p>
                `;
                card.addEventListener('click', () => {
                    console.log(`Opening Wikipedia URL: ${spot.wikiUrl}`);
                    window.open(spot.wikiUrl, '_blank');
                });
                document.getElementById('spots-list').appendChild(card);
            });

            document.getElementById('spots-list').innerHTML = spots.length === 0 ? '<p style="color: #cccccc; padding: 20px;">Місця не знайдено.</p>' : document.getElementById('spots-list').innerHTML;
        }
    </script>
</body>
</html>
```

ДОДАТОК В (обов'язковий)
ІЛЮСТРАТИВНА ЧАСТИНА

"WEB-ПЛАТФОРМА ДЛЯ ВИБОРУ ТУРИСТИЧНОГО МАРШРУТУ"

Виконав: студент 4 курсу, групи 3КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Романюк Д.О.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Колодний В.В.
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ - 2025 рік

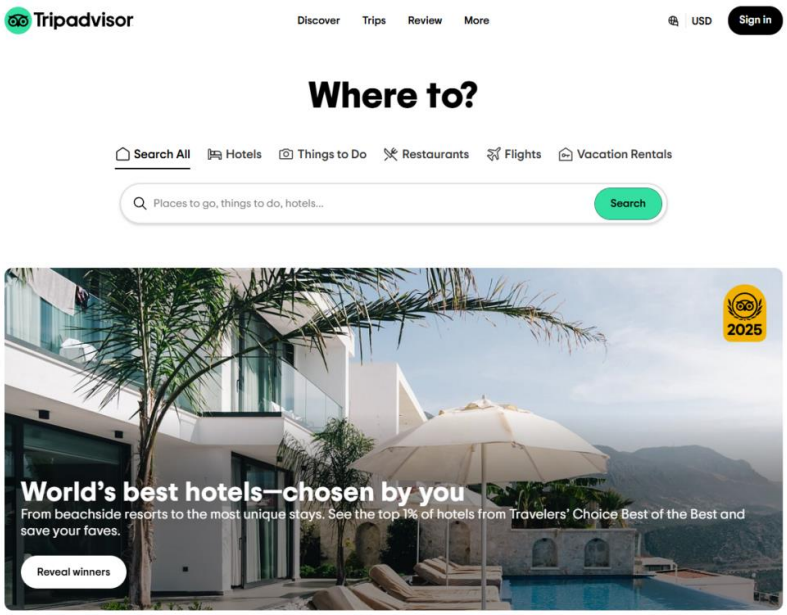

Рисунок В.1 – Головна сторінка TripAdvisor.

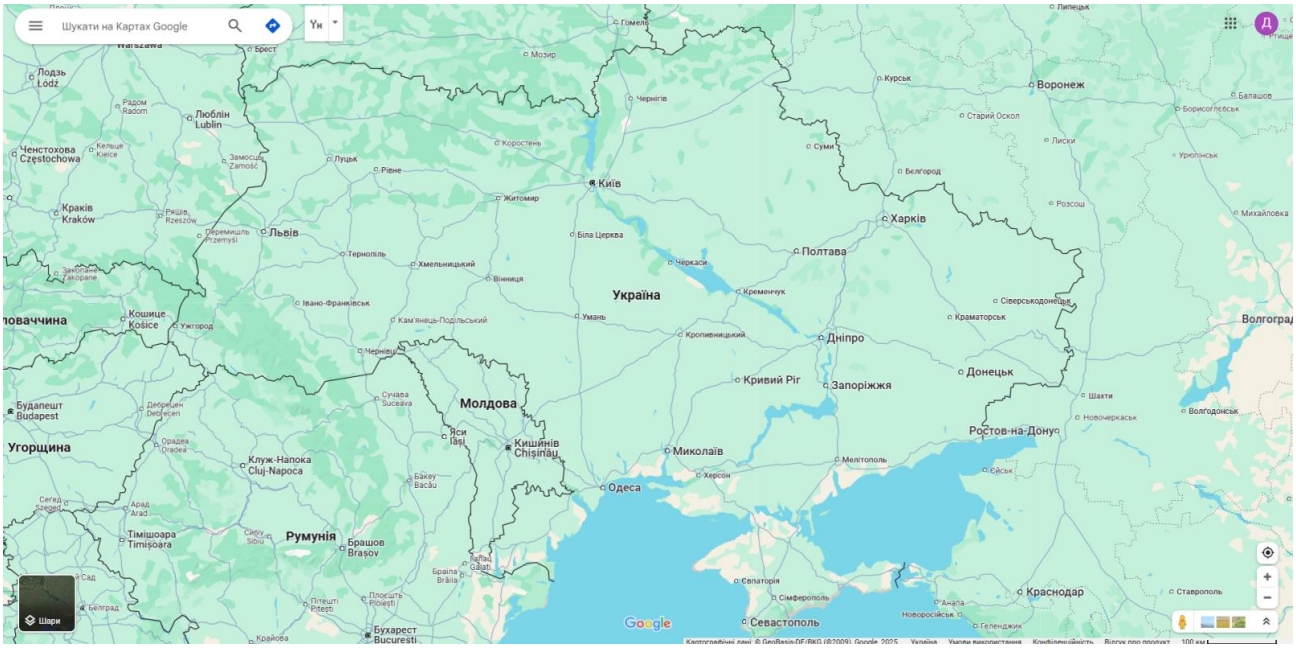

Рисунок В.2 – Головна сторінка Google Maps.

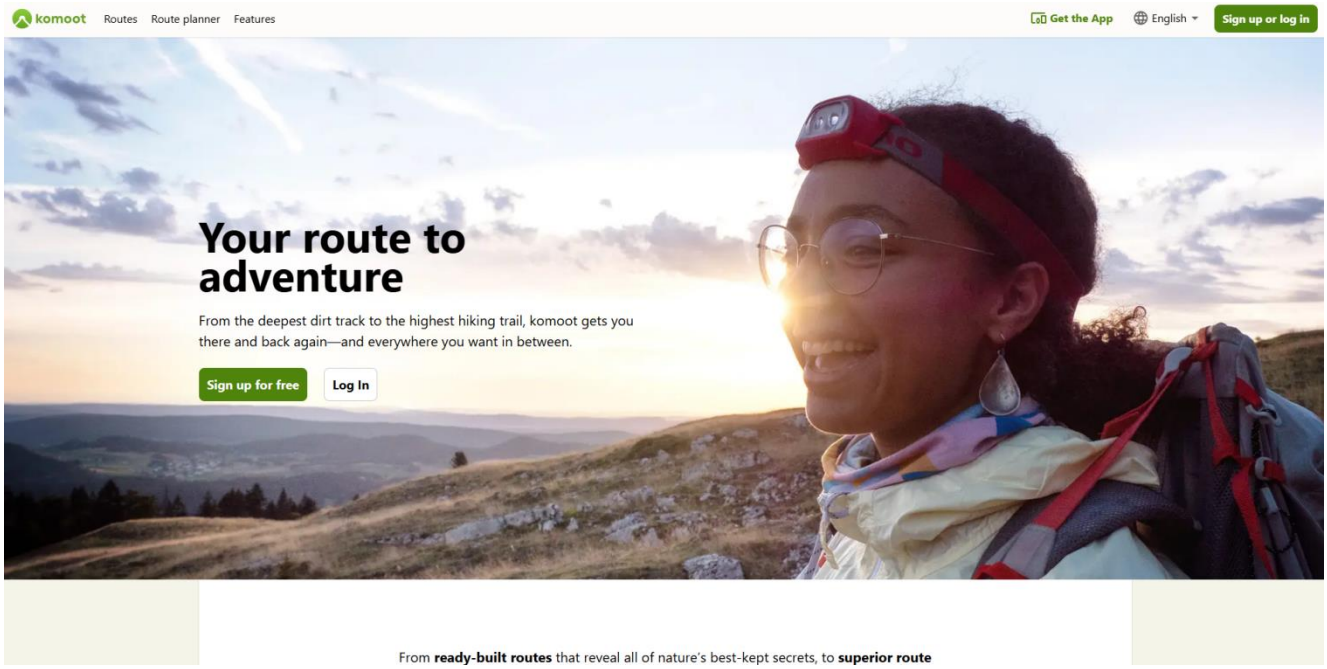

Рисунок В.3 – Головна сторінка Komoot

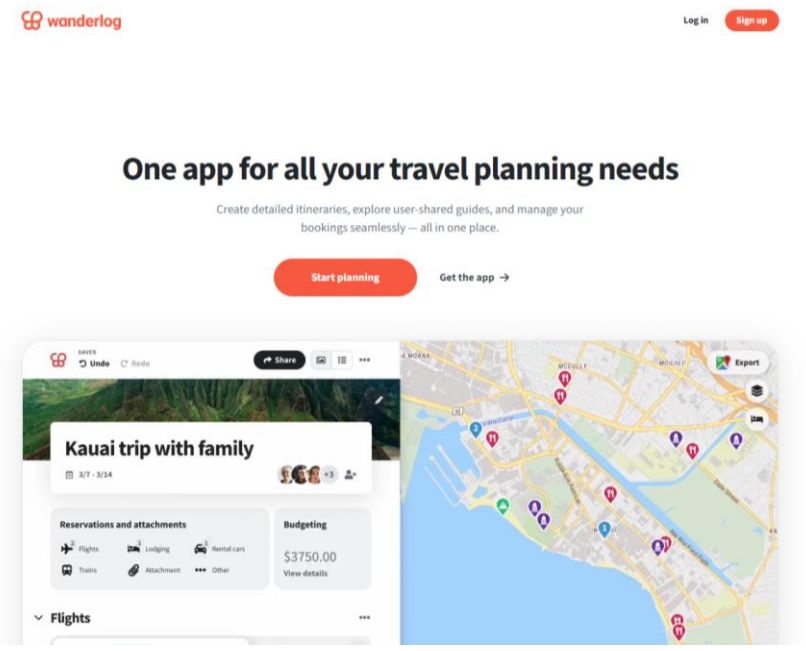

Рисунок В.4 – Головна сторінка Wanderlog

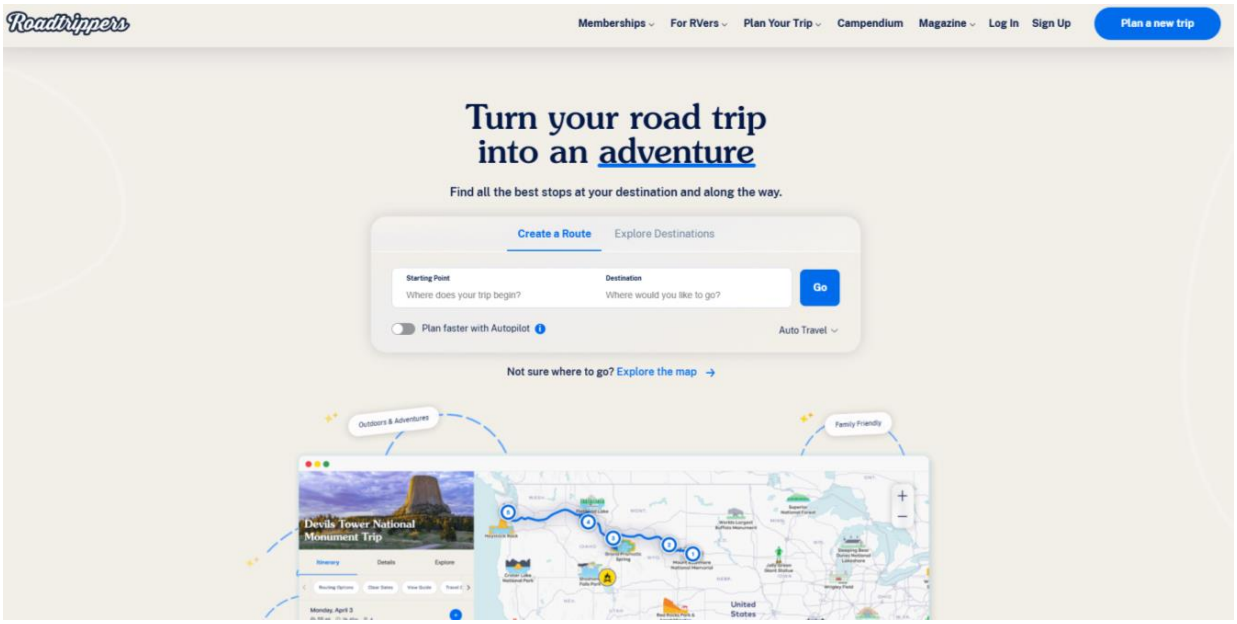

Рисунок В.5 – Головна сторінка Roadtrippers

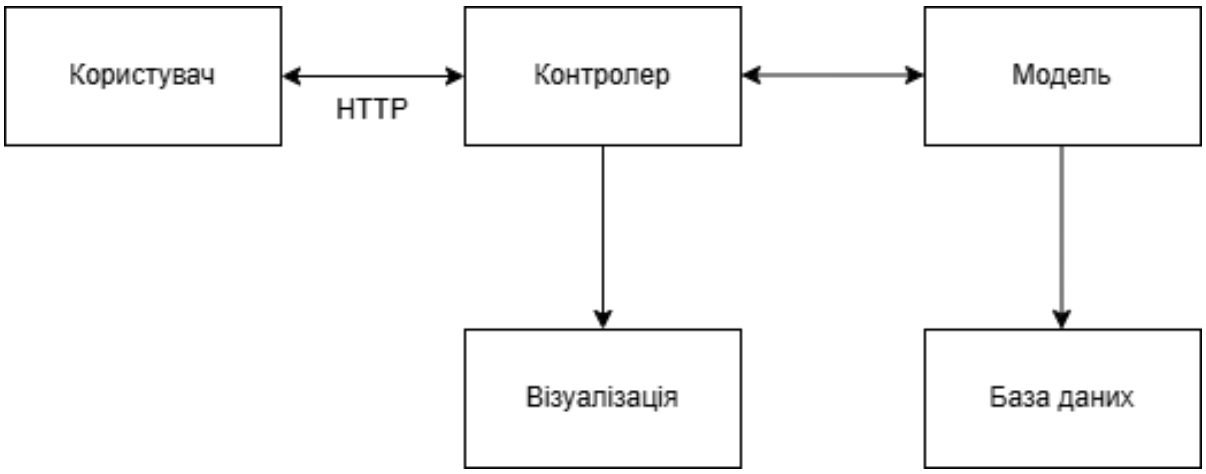

Рисунок В.6 – Схематична візуалізація MVC архітектури

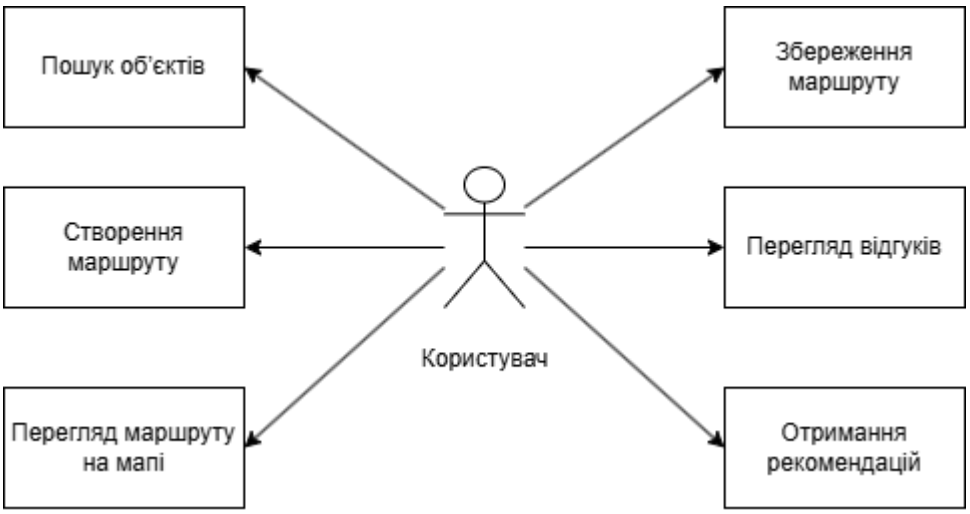

Рисунок В.7 – Візуалізація діаграми прецедентів

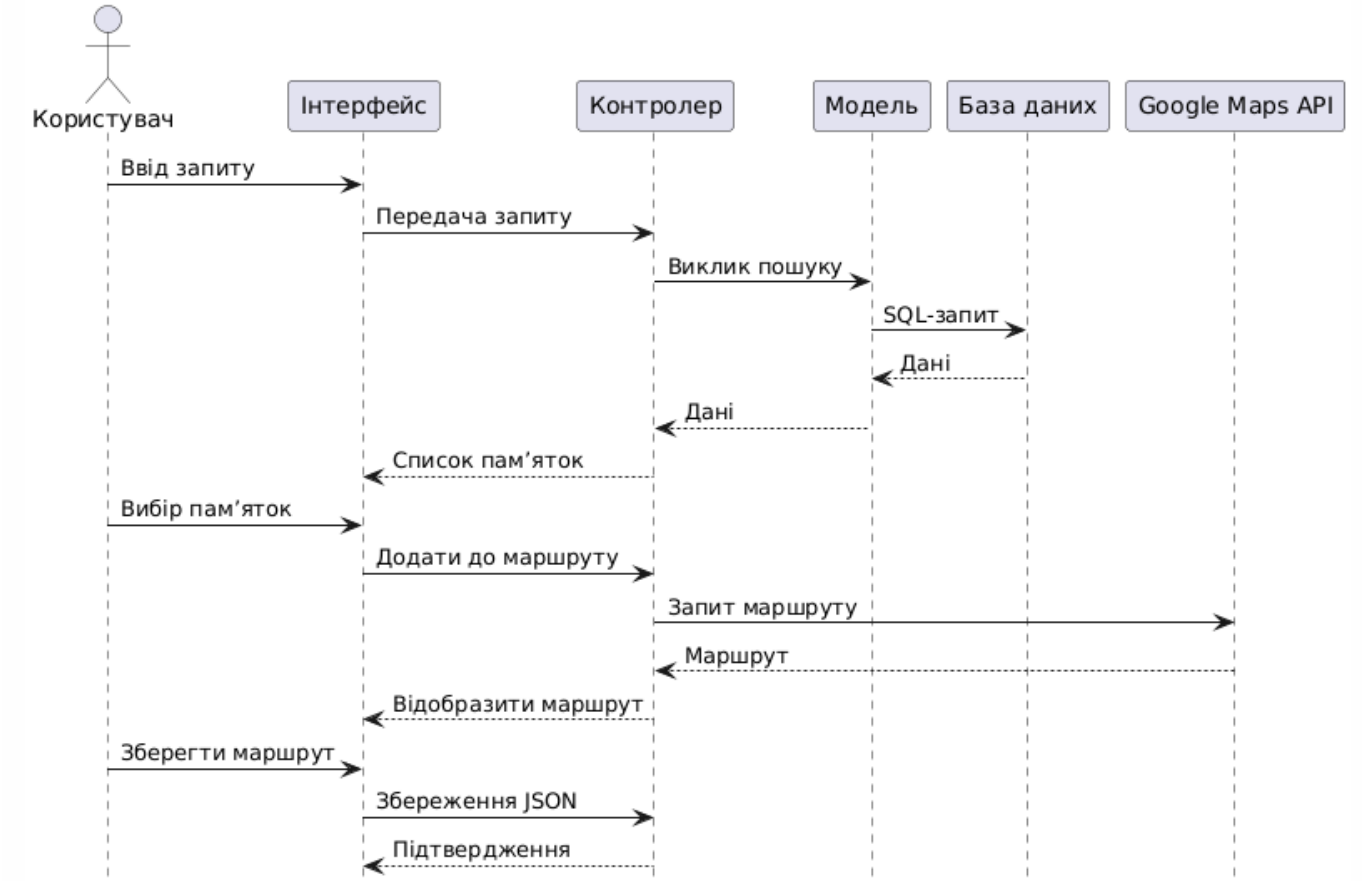

Рисунок В.8 – Візуалізація діаграми послідовності

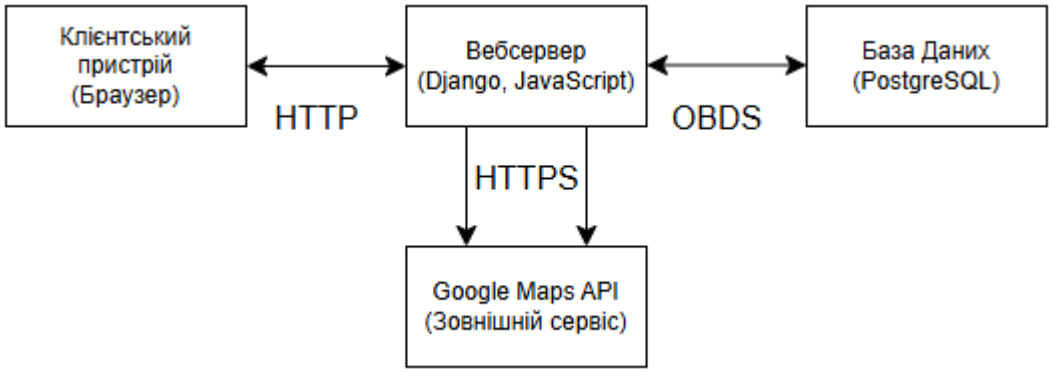

Рисунок В.9 – Візуалізація діаграми розгортання

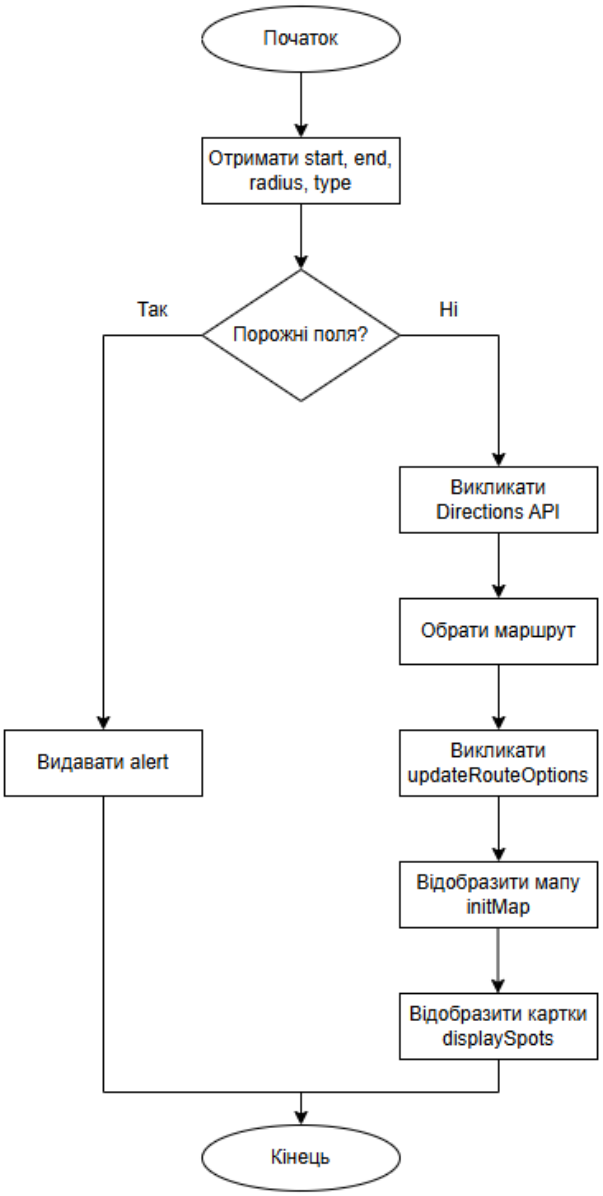

Рисунок В.10 – Блок-схема алгоритму побудови маршруту.

Рисунок В.11 – Блок-схема алгоритму пошуку пам’яток.

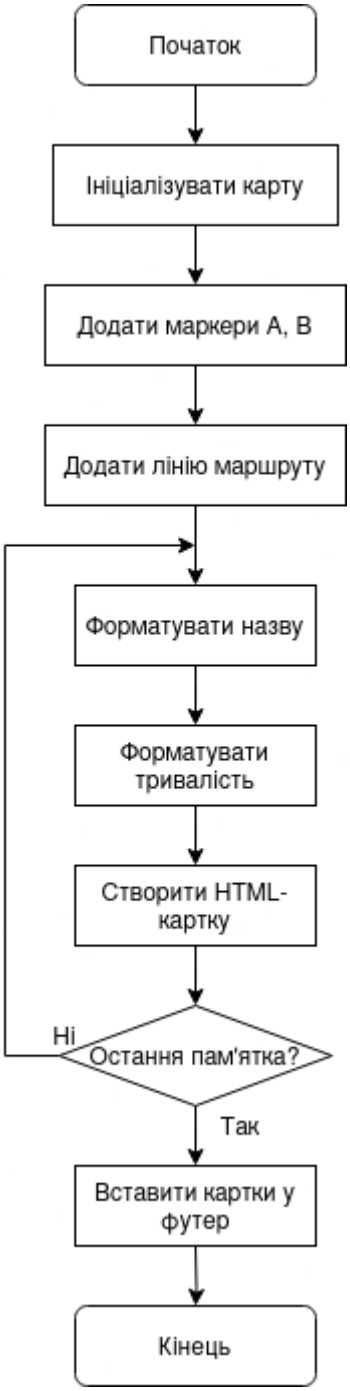

Рисунок В.12 – Блок-схема алгоритму відображення

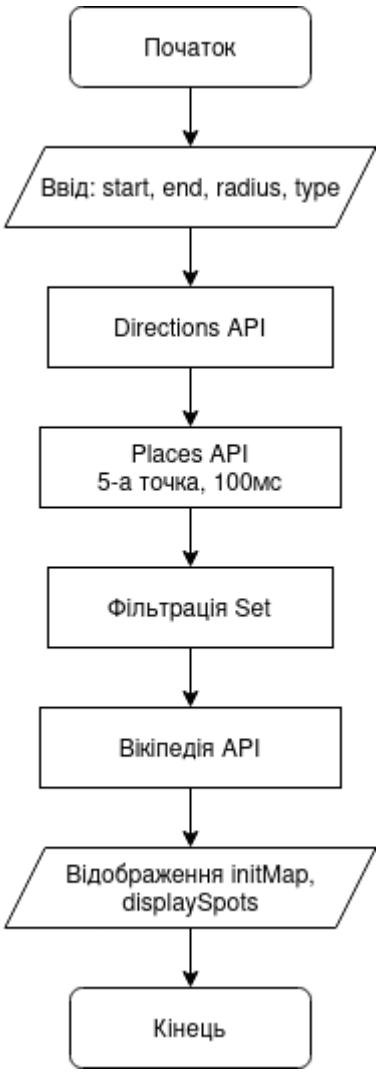

Рисунок В.13 – ASCII-схема.

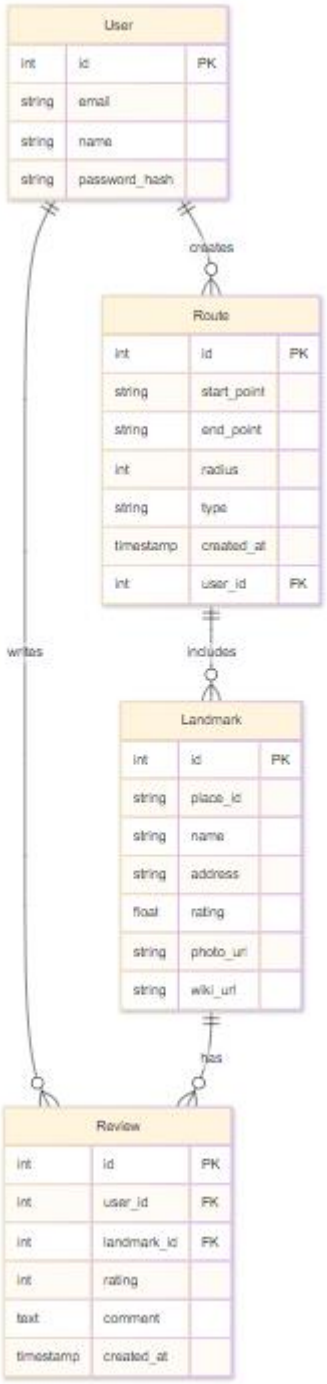

Рисунок В.14 – Схема бази даних

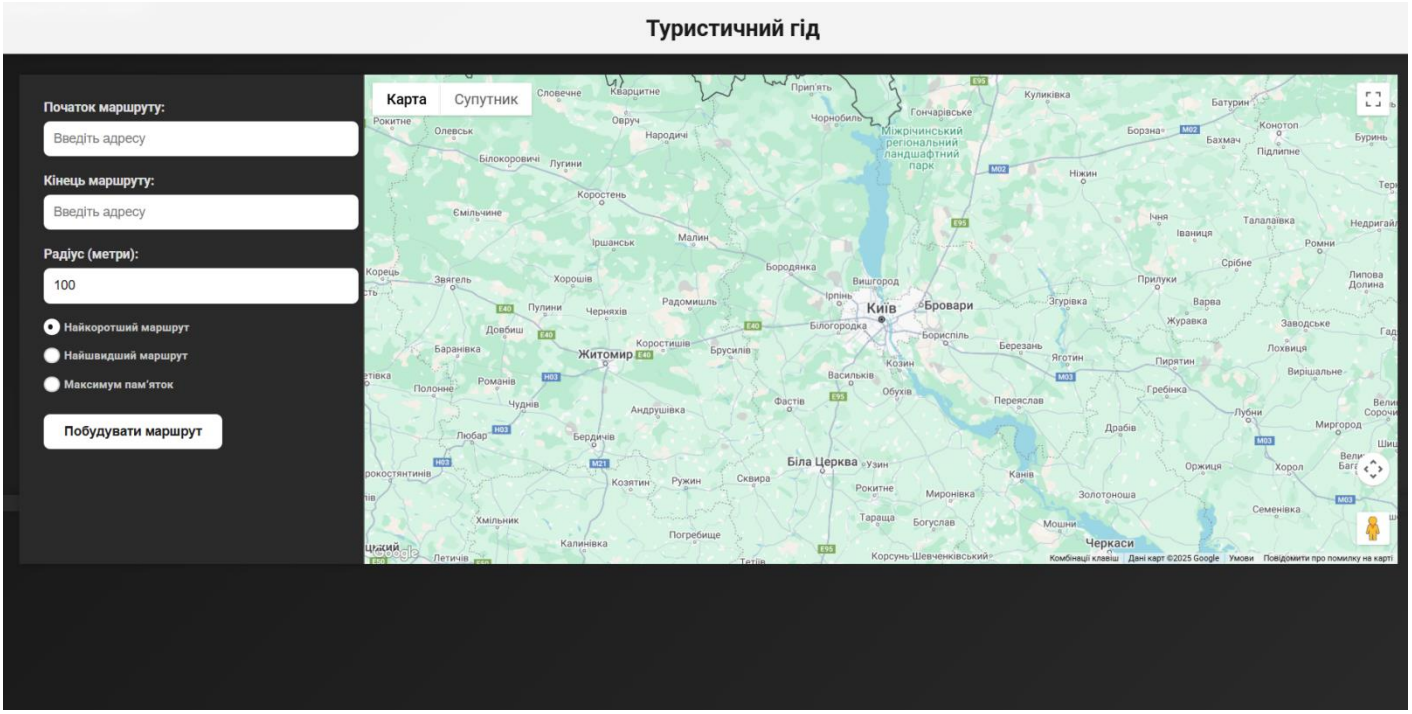

Рисунок В.15 – Інтерфейс головної сторінки

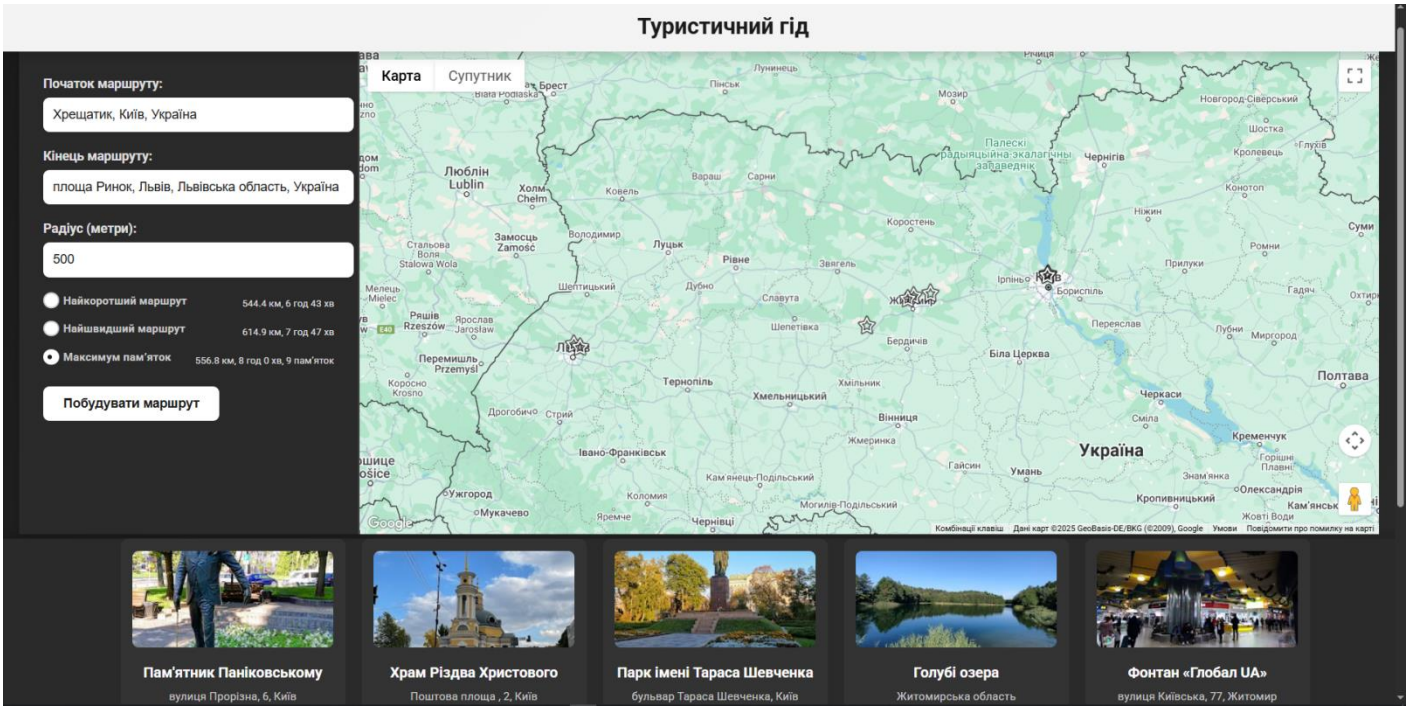

Рисунок В.16 – Сторінка з маршрутом і пам’ятками

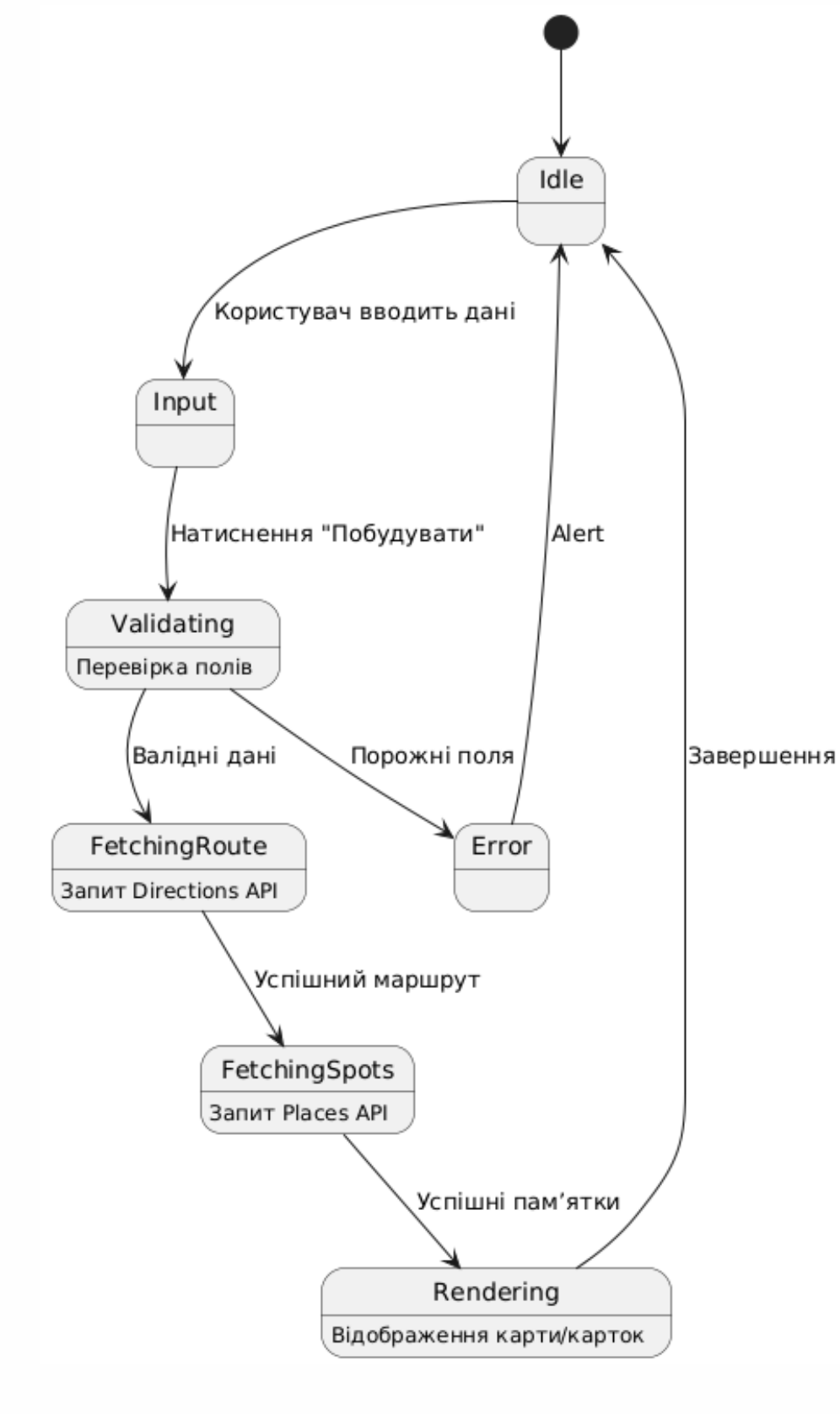

Рисунок В.17 – UML-діаграма станів

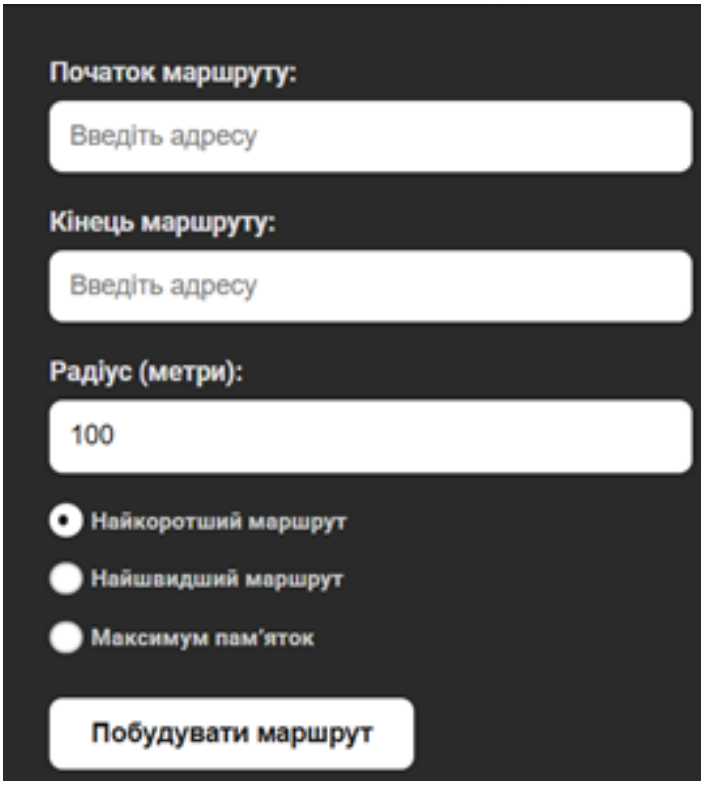

Рисунок В.18 – Меню вибору точок маршруту

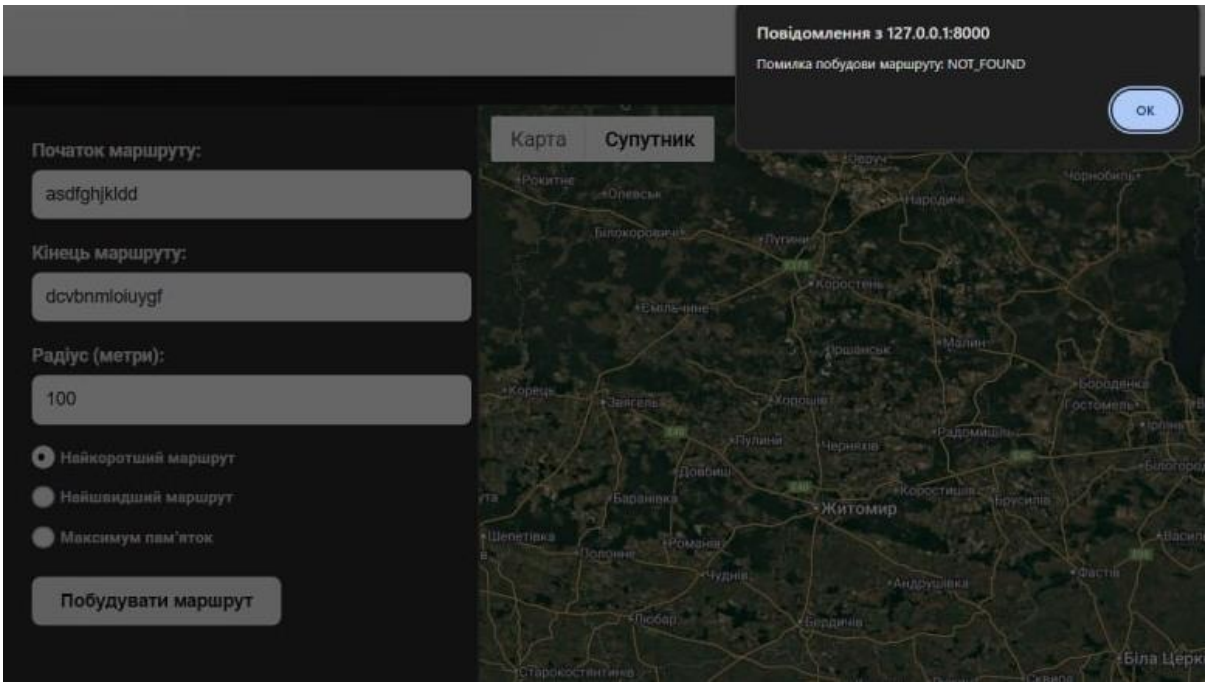

Рисунок В.19 – Помилка побудови маршруту

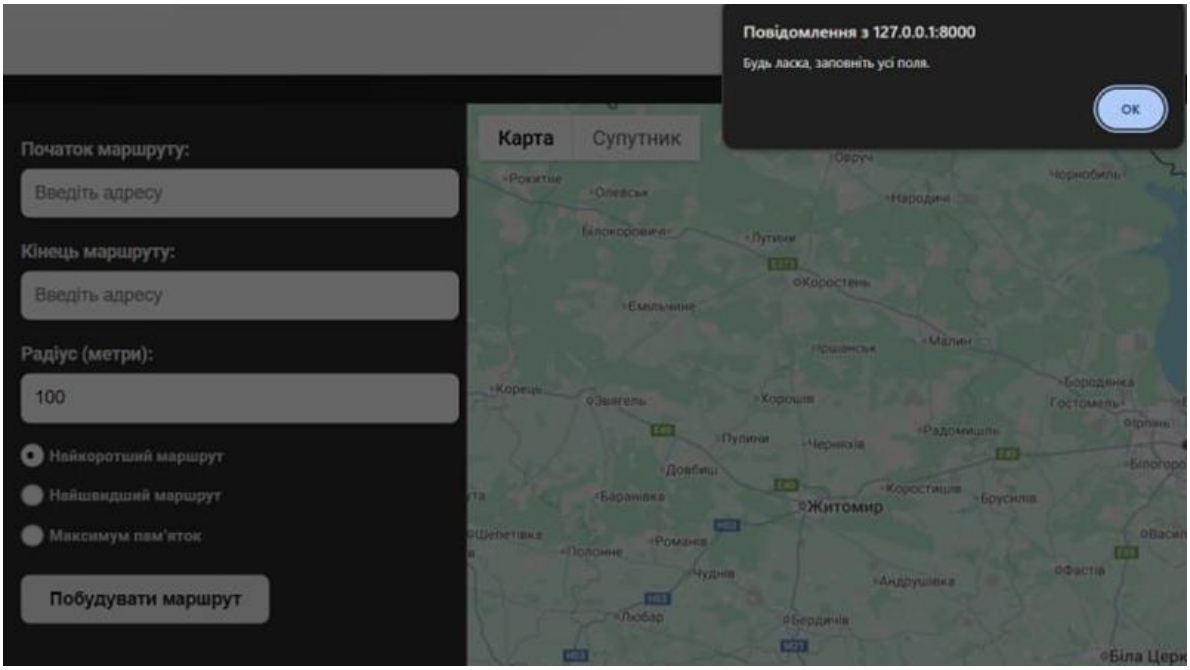

Рисунок В.20 – Результат пошуку маршруту без введення даних

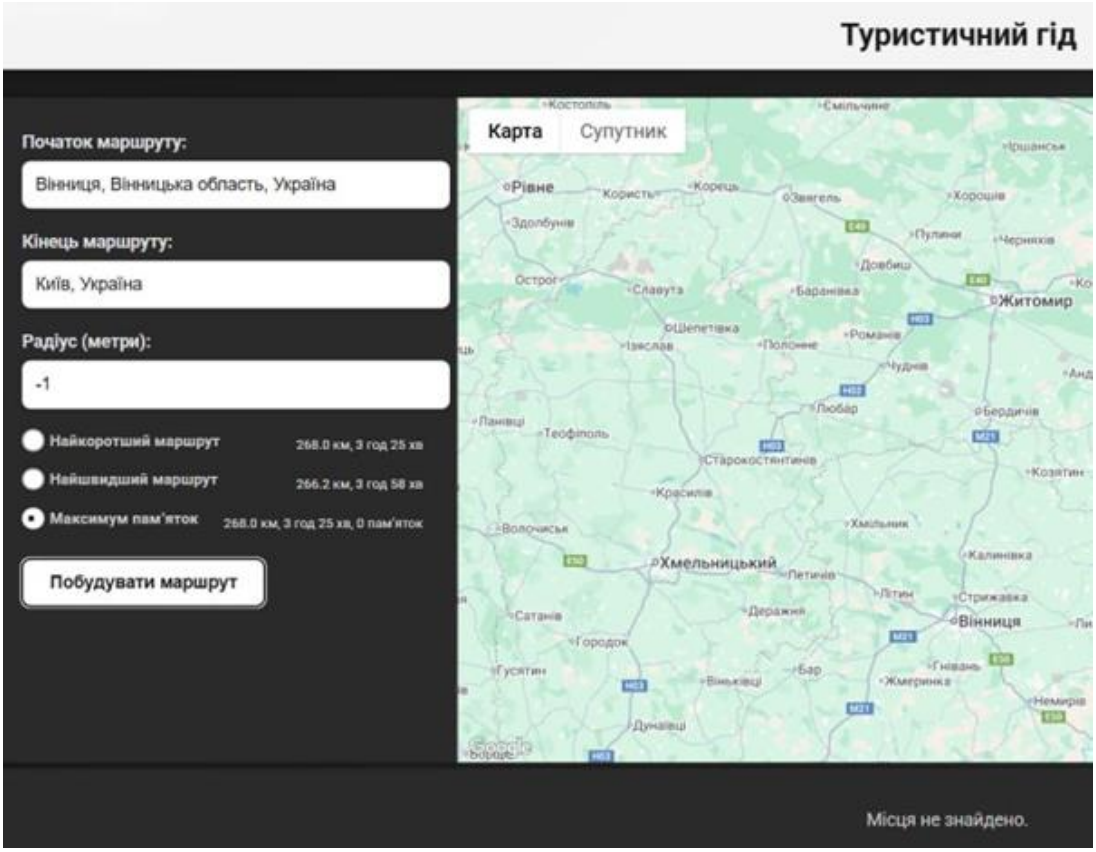

Рисунок В.21 – Повідомлення “Місця не знайдено” при не введенні даних

ДОДАТОК Г (довідниковий)

ДОВІДКА ПРО ВПРОВАДЖЕННЯ

МАЛЕ НАУКОВО-ВИРОБНИЧЕ ПІДПРИЄМСТВО "ТОВ "ІТІ"
Україна, 21021, м.Вінниця, вул. Келецька, 56

№ 60 від "6" 06 2025 р.

ДОВІДКА

Дана Романюк Дмитру Олександровичу в тому, що результати, одержані ним в процесі виконання бакалаврської кваліфікаційної роботи, а саме алгоритми та програмні засоби для вибору туристичних маршрутів, планується використати в розробках ТОВ «ІТІ».

Зам. директора ТОВ «ІТІ» Бодяк В.М.