ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Дослідження та реалізація web-додатку прокладання оптимальних маршрутів для подорожей»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Селезень Артем
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти Артем СЕЛЕЗЕНЬ
група ІСДМ-62

Керівник: Оксана ТКАЛЕНКО.
науковий ступінь, вчене звання д.т.н., доцент

Рецензент:
науковий ступінь, вчене звання _____
Ім'я, ПРІЗВИЩЕ

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС

_____ Каміла СТОРЧАК
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Селезня Артема Васильовича

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Дослідження та реалізація web-додатку для прокладання оптимальних маршрутів для подорожей
керівник кваліфікаційної роботи Ткаленко О.М. д.т.н., доцент,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023р. №145
2. Строк подання кваліфікаційної роботи «29» грудня 2023р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, дані з сайтів компаній-виробників.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження ринку

5. Перелік графічного матеріалу: *презентація*
    1. Актуальність теми подорожей
    2. Порівняльний аналіз існуючих додатків
    3. Алгоритми маршрутизації
    4. Вибір методологій для створення застосунку
    5. Вибір технологій для створення застосунку
    6. Приклад використання застосунку
    7. Висновки

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Ознайомлення з наявною науково-технічною літературою	19.10-05.11.23	
2	Аналіз актуальності теми подорожей	05.11-12.11.23	
3	Перегляд сайтів та застосунків для подорожей	13.11-19.11.23	
4	Огляд застосунків керування освітленням	20.11-25.11.23	
5	Вибір технологій для розробки веб застосунку	27.11-03.12.23	
6	Розробка веб застосунку	04.12-10.12.23	
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	
8	Розробка демонстраційних матеріалів	21.12-29.12.23	

Здобувач вищої освіти _____ Артем СЕЛЕЗЕНЬ

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник

кваліфікаційної роботи _____ Оксана ТКАЛЕНКО

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 84 стор., 2 табл., 7 рис., 13 джерел.

Мета роботи – концептуалізація, розробка та впровадження функціонального веб-додатка, спеціально створеного для визначення оптимальних маршрутів подорожей.

Об'єкт дослідження – веб-додаток, спеціально розроблений для допомоги людям у ретельному плануванні оптимальних маршрутів подорожей.

Предмет дослідження – концептуалізація, дизайн, розробка та оцінка веб-додатку, призначеного для допомоги користувачам у ретельному плануванні оптимальних маршрутів подорожей.

Короткий зміст роботи: Цей комплексний дослідницький проект зосереджується на концептуалізації, дизайні, розробці та оцінці веб-додатку, призначеного для допомоги користувачам у ретельному плануванні оптимальних маршрутів подорожей. Основна мета цієї ініціативи — розробити інтуїтивно зрозумілий, високоефективний інструмент, який дозволить мандрівникам планувати свої подорожі з точністю та зручністю.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, ОПТИМІЗАЦІЯ, АЛГОРИТМ, МЕТОЛОГОГІЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

ABSTRACT

Text part of the master's qualification work: 91 pages, 19 pictures, 1 table, 37 sources.

Purpose of the work: conceptualization, development, and implementation of a functional web application specifically designed to determine optimal travel routes.

Object of research: a web application specifically designed to help people carefully plan optimal travel routes Subject of study Conceptualization, design, development, and evaluation of a web application designed to assist users in carefully planning optimal travel routes.

Short content of the work: This comprehensive research project focuses on conceptualizing, designing, developing, and evaluating a web application for determining the optimal travel routes. The primary objective of this initiative is to develop an intuitive, high-performance tool that enables travelers to plan their journeys with accuracy and convenience. the work: conceptualization, development and implementation of a functional web application specifically designed to determine optimal travel routes. The main goal is to create an intuitive and effective platform that uses advanced algorithms, real-time data analysis and user-oriented functionality to recommend the most optimal routes for different types of trips, taking into account the diverse preferences and limitations of users.

KEYWORDS: WEB APPLICATION, OPTIMZATION, ALGORYTHM, METHOLOGY, SOFTWARE.

ЗМІСТ

ВСТУП…….….. 8
РОЗДІЛ 1. ОГЛЯД ЛІТЕРАТУРИ……..…… 10
1.1 Еволюція технологій оптимізації маршрутів подорожей ……..…….. 11
1.1.1 Ранні початки та ручні методи……. 12
1.1.2 Розвиток технології GPS…… 15
1.1.3 Підйом цифрового картографування …….. 16
1.2 Веб-додатки для планування маршрутів: Огляд та аналіз ……. 18
1.2.1 Функціональність веб-додатків для планування маршрутів……. 19
1.2.2 Популярні веб-додатки для планування маршрутів…….. 21
1.3 Алгоритми та підходи у плануванні оптимальних маршрутів……. 26
1.3.1 Алгоритм Дейкстри…… 27
1.3.2 Алгоритм A*…… 28
1.3.3 Алгоритм Флойда–Воршалла……. 31
1.3.4 Генетичні алгоритми……... 32
1.3.5 Алгоритми імітації відпалу…… 34
1.3.6 Алгоритм ATSP…… 35
1.3.7 Алгоритм ACO…… 38
1.4 Дизайн з орієнтацією на користувача та інтерфейс у подорожніх додатках…… 41
РОЗДІЛ 2. ОГЛЯД МЕТОДОЛОГІЙ……..…… 43
2.1 Методологія збору даних та розробки програмного забезпечення ….. 44
2.1.1 Методи збору даних ……..……..... 44
2.1.2 Методології розробки програмного забезпечення ……. 46
2.1.3 Waterfall ……...… 47
2.1.4 Agile …….…… 49
2.1.5 V-модель……... 51
2.1.6 DevOps……. 54
2.1.7 Методологія швидкої розробки (RAD)…… 56
2.2 Вибір правильної методології…….. 58

2.3 Вибір алгоритмів та стратегії впровадження……. 59
2.4 Тестування на користування та оцінка…… 62
РОЗДІЛ 3. АНАЛІЗ ВИМОГ ТА РОЗРОБКА…….. 64
3.1 Специфікація функціональних вимог…….. 64
3.2 Специфікація нефункціональних вимог……. 66
3.3 Аналіз вимог для профіля користувача та використання …… 68
3.4 Архітектура системи та огляд компонентів ……... 70
3.4.1 Компоненти …… 72
3.5 Дизайн інтерфейсу користувача та створення прототипів…… 73
3.6 Розробка та інтеграція алгоритмів ……. 74
3.6.1 Функція astar…….…. 76
3.6.2 Евристична функція…….. 78
3.6.3 Клас графа……. 79
3.7 Структура бази даних та обробка даних……. 82
3.7.1 Структура бази даних…… 82
3.7.2 Обробка даних…… 83
3.8 Оцінка та результати…… 84
ВИСНОВКИ……..……. 86
ПЕРЕЛІК ПОСИЛАНЬ……..……..…….. 88
ДОДАТОК А: Макети інтерфейсу користувача…….. 90
ДОДАТОК Б: Витрішки коду або псевдокод алгоритмів…… 94
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) ……...……..... 98

ВСТУП

Актуальність теми: швидка урбанізація призводить до зростання транспортних заторів у містах, тому ефективні маршрути важливі для зменшення часу подорожі, покращення мобільності та зниження забруднення. Неефективні маршрути також призводять до зайвих витрат часу та грошей для осіб, бізнесу та громадських служб. Водночас, зростаюча залежність від смартфонів підвищує попит на інтуїтивні, користувацькі веб-додатки для планування маршрутів, які відповідають на потреби сучасних мандрівників у персоналізованих, ефективних рішеннях. Недавні технологічні досягнення в області картографування та аналізу даних надають можливість розробки складних веб-додатків для динамічного планування маршрутів, що відповідають на змінювані очікування та уподобання користувачів.

Об'єкт дослідження: функціональність та ефективність веб-додатка, створеного спеціально для оптимізації маршрутів подорожей.

Предмет дослідження: оцінка методик, алгоритмів та технологічних рамок, використаних при впровадженні веб-додатка, різноманітність технік, використаних для оптимізації маршрутів з урахуванням факторів, таких як інтеграція даних у реальному часі, введення користувача, ефективність алгоритмів та можливість адаптації додатка до різних сценаріїв.

Завдання дослідження: комплексний аналіз існуючих технологій, алгоритмів та методологій у веб-додатках для оптимізації маршрутів подорожей. Це включає огляд поточних технічних рішень та їхні переваги та недоліки. Також передбачається збір вимог до веб-додатка від зацікавлених сторін та користувачів, визначення функціональних можливостей і нефункціональних аспектів, таких як масштабованість та безпека. Дослідження також охоплює розробку інтуїтивно зрозумілих користувацьких інтерфейсів та проектування алгоритмів для обробки даних подорожей і генерації оптимальних маршрутів, з можливістю динамічного коригування маршрутів на основі актуальних даних.

Наукова новизна: розробка та впровадження інноваційних алгоритмів, спеціально створених для оптимізації маршрутів подорожей.

Практична значущість результатів: зменшення часу подорожей та оперативних витрат, а також сприяючи екологічно стійким рішенням. Воно включає розробку інтуїтивно зрозумілого веб-додатку, що використовує передові алгоритми та обробку даних, забезпечуючи користувачам зручність та ефективність у плануванні маршрутів.

РОЗДІЛ 1. ОГЛЯД ЛІТЕРАТУРИ

У першому розділі даної дипломної роботи виконується важливий крок у сприйнятті об'єкта та предмета дослідження - огляд літератури. Цей розділ відіграє ключову роль у розкритті контексту та стану сучасних досліджень та розробок, що стосуються тематики роботи.

Маючи на меті розглянути аспекти оптимізації маршрутів подорожей та веб-додатків для їх планування, цей розділ включає в себе аналіз різноманітних джерел, від наукових статей та книг до практичних додатків та веб-сервісів. Він також простежує історичний шлях розвитку технологій у цій галузі та стежить за перетвореннями вимог користувачів і відповідною реакцією розробників на ці зміни.

Спеціальна увага приділяється аналізу еволюції технологій оптимізації маршрутів подорожей, що включає в себе розгляд динаміки змін у цій сфері, від початкових варіантів до сучасних рішень, що використовують передові технології і методи оптимізації.

Додатково, в розділі проводиться огляд веб-додатків, які призначені для планування маршрутів, та їхній аналіз. Відомі додатки, які забезпечують користувачам можливість вибору оптимального маршруту для подорожей, розглядаються з точки зору їхніх функціональних можливостей, інтерфейсу та задоволення потреб користувачів.

Далі у розділі досліджуються різні алгоритми та підходи, які використовуються в плануванні оптимальних маршрутів, із наголосом на їхню ефективність та застосування в реальних ситуаціях. Також розглядаються аспекти дизайну з орієнтацією на користувача та інтерфейсу в подорожніх додатках, оскільки зручність взаємодії та сприйняття інформації мають велике значення для успішного використання таких додатків.

1.1 Еволюція технологій оптимізації маршрутів подорожей

Приход Інтернету позначив революційний стрибок у галузі картографії та оптимізації маршрутів, відкривши еру, де цифрові карти стали більш доступними, інтерактивними та необхідними в повсякденному житті. Цей розділ досліджує трансформаційний вплив онлайн-служб картографування, які переосмислили те, як люди та підприємства орієнтуються в світі.

В центрі цієї трансформації було перехід від статичних фізичних карт до динамічних цифрових карт, доступних в мережі Інтернет. Ця зміна була не лише зміною в середовищі карт, але й повністю переробкою того, як люди взаємодіють із географічною інформацією. Це призвело до розвитку складних онлайн-платформ, які надавали не лише вказівки, але й оновлення в реальному часі про трафік, планування маршрутів на основі різних критеріїв та навіть зображення на рівні вулиць.

Буде поглиблено в ранні дні онлайн-картографування, виділяючи піонерські служби, які поклали основу для сучасних високотехнологічних систем. Акцент буде зроблено на тому, як ці платформи еволюціонували, інтегруючи все більш вдосконалену технологію та інтерфейси користувача. Ця еволюція була прискорена кількома ключовими чинниками: досягненнями в області ГІС (географічних інформаційних систем), широкою доступністю Інтернету та, пізніше, всезагальністю смартфонів.

Від ранніх примітивних онлайн-карт до сучасних платформ з багатим функціоналом, цей розділ подасть шлях швидкого зростання цифрового картографування та його глибокий вплив на особисту навігацію, міське планування та загальну галузь науки про географічну інформацію.

Під час вивчення цього шляху ми також розглянемо виклики та можливості, які виникли з цими технологіями, встановлюючи основу для розуміння їх поточного стану та потенційних майбутніх розвитків у постійно змінному пейзажі цифрового картографування та оптимізації маршрутів.

1.1.1 Ранні початки та ручні методи

Винахід друкарської машини Йоганном Гутенбергом у середині XV століття став однією з найважливіших подій у історії людства, що мала глибокий вплив на поширення знань та інформації, включаючи картографічні матеріали. До цього моменту карти створювалися вручну, що було дуже трудомістким та дорогим процесом, обмежуючи їхню доступність.

Революція у Виготовленні та Поширенні Карт:

-Стандартизація та Масове Виробництво. Друкарська машина дозволила стандартизувати виробництво карт, роблячи їх масово доступними. Вперше картографічні матеріали могли бути відтворені у великій кількості зі збереженням високої точності та якості.

-Зниження Вартості та Поширення Знан. Завдяки друкарській машині вартість виробництва карт значно знижувалася, що робило їх доступними ширшому колу людей. Це сприяло поширенню географічних знань серед громадськості та професійних мореплавців.

Вплив на Мореплавство та Географічні Відкриття:

-Підтримка Морських Подорожей: З доступністю точних карт мореплавці отримали кращі інструменти для навігації, що сприяло великим географічним відкриттям того періоду. Карти стали незамінним інструментом у плануванні маршрутів і дослідженні нових територій.

-Розвиток Картографії та Науки: Поширення карт також сприяло розвитку самої картографії як науки. Вчені та дослідники могли легше обмінюватися географічною інформацією, що призвело до більш точного зображення світу на картах.

Соціальний та Культурний Вплив

-Освітній Аспект: Доступність карт сприяла освіті та зростанню географічної обізнаності серед населення. Люди мали змогу вивчати світ за межами своєї місцевості, розширюючи свої знання та розуміння географії.

-Зміцнення Торгівлі та Економічний Розвиток: Точні карти сприяли розвитку міжнародної торгівлі, оскільки торговці та підприємці могли планувати ефективніші торговельні маршрути.

Винахід друкарської машини і подальше поширення карт значно вплинуло на хід історії, відкриваючи нові горизонти в мореплавстві, науці, освіті та культурі. Це було справжнє вікно у світ для багатьох людей того часу, розширюючи границі відомого та сприяючи загальному розвитку людства.

Вплив друкованих карт у епоху великих географічних відкриттів був визначальним фактором, що сприяв дослідженням та відкриттям нових територій мореплавцями, такими як Христофор Колумб, Васко да Гама та Фернан Магеллан. Поширення друкованих карт після винаходу друкарської машини Йоганном Гутенбергом у XV столітті мало значний вплив на морські дослідження. До винаходу друкарської машини карти розмножувалися вручну, що робило їх дорогими та недоступними для широкого кола людей. Друковані карти були значно доступнішими, що дозволило більшій кількості мореплавців та дослідників використовувати їх для планування подорожей. Друковані карти були точнішими та стандартизованими, надаючи мореплавцям надійніші та докладніші зображення земель та морських шляхів.

Вплив на Великих Дослідників:

-Христофор Колумб: Колумб використовував карти та морські хартії для планування своїх експедицій до Нового Світу. Доступ до друкованих карт допоміг йому визначити потенційні маршрути та спрогнозувати можливі перешкоди.

-Васко да Гама: Для Васко да Гами, який відкрив морський шлях до Індії, друковані карти були важливим джерелом інформації, що допомогло йому навігувати невідомими водами.

-Фернан Магеллан: Магеллан використовував друковані карти для планування своєї подорожі навколо світу, що дозволило йому зробити точніші припущення про маршрути та відстані.

Друковані карти в епоху великих відкриттів відіграли вирішальну роль у розвитку морських досліджень та географічних відкриттів. Вони не тільки надали

мореплавцям необхідні інструменти для навігації та дослідження, але й сприяли поширенню знань про світ, підсилюючи культурний та економічний зв'язок між різними частинами світу.

Впровадження магнітного компаса в навігації також мало фундаментальний вплив на морську навігацію та мореплавство. Його походження та вплив можна розглядати у кількох ключових аспектах. Магнітний компас має своє коріння у древньому Китаї, де він був спочатку використаний для фен-шуй та астрології. Згодом, його значення було виявлено в морській навігації. Існують свідчення того, що компас був використаний у Китаї для морської навігації вже у XI столітті, а пізніше він поширився в Арабському світі та Європі.

Вплив на Морську Навігацію:

-Орієнтація на Морі: Перед впровадженням компаса мореплавці орієнтувалися на зірках та інших природних орієнтирах, що було особливо важко під час поганої видимості або погоди. Компас забезпечив засіб постійної та точної орієнтації.

-Розвиток Торговельних Маршрутів: Використання компаса дозволило мореплавцям здійснювати довші та складніші плавання, відкриваючи нові торговельні шляхи та стимулюючи морську торгівлю.

-Безпека та Надійність: Завдяки компасу мореплавці могли впевнено навігувати в відкритому морі, зменшуючи ризик заблукати або зазнати корабельної аварії.

1.1.2 Розвиток технології GPS

Історія розвитку Глобальної Системи Позиціонування (GPS) є вражаючим прикладом переходу технології від військового до цивільного використання. Спочатку розроблений та впроваджений військовими США для забезпечення точного позиціонування, навігації та часового синхронізування, GPS став незамінним інструментом у сучасному світі, що використовується у різноманітних цивільних сферах.

Первісна концепція GPS була розроблена у 1970-х рыоках військовими США як відповідь на потребу в глобальній, цілодобовій навігаційній системі. Перші супутники були запущені у 1978 році, і система поступово розвивалася, доки не стала повністю функціональною. Вона дозволяла з високою точністю визначати географічне положення об'єктів у будь-якій точці земної кулі, що було критично важливим для військових потреб.

Вплив цієї технології на цивільні сфери почав відчуватися у 1980-х роках, коли уряд США прийняв рішення надати доступ до GPS для цивільного використання. Це рішення було частково спонукане трагедією збитого південнокорейського пасажирського літака у 1983 році, який відхилився від курсу через помилки в навігації.

Відкриття GPS для цивільного використання сприяло стрімкому розвитку та інтеграції технології у повсякденне життя. Від автомобільної навігації до смартфонів, від геодезичних досліджень до відстеження маршрутів доставки - GPS став невід'ємною частиною сучасного суспільства. Він дозволив значно підвищити точність та ефективність багатьох процесів, сприяючи розвитку комерції, транспорту, науки та безпеки.

Таким чином, історія GPS є прикладом успішного переходу від військової технології до широкого цивільного використання, що відкрило нові можливості та сприяло глобальному технологічному прогресу.

Покращення точності та надійності: З часу свого створення технологія GPS пройшла значний шлях розвитку, значно покращивши точність та надійність, що

зробило її незамінною у сучасному плануванні маршрутів. Спочатку розроблена для військових цілей, GPS тепер є основою для широкого спектру цивільних застосувань, від навігації в автомобілях до синхронізації часу в мережах зв'язку.

Одним з ключових досягнень у вдосконаленні GPS було зняття обмежень точності, яке було введено урядом США для цивільних користувачів. Це рішення, прийняте на початку 2000-х років, значно підвищило точність цивільних GPS-приймачів, дозволяючи визначати місцезнаходження з точністю до кількох метрів.

Крім того, технологічні інновації, такі як вдосконалення супутникових технологій та алгоритмів обробки сигналів, значно покращили надійність та точність GPS. Розвиток альтернативних глобальних навігаційних супутникових систем, таких як Галілео (Європейський Союз) та BeiDou (Китай), забезпечив додаткову точність та надійність шляхом використання різних супутникових мереж.

Ці покращення мали великий вплив на планування маршрутів у сучасному світі. GPS тепер використовується для широкого спектру застосувань, від особистої навігації та відстеження місцезнаходження до розробки складних логістичних систем та управління транспортними потоками. Точність GPS сприяє ефективній організації транспорту, зменшує час у дорозі та підвищує безпеку переміщення.

У цілому, досягнення в технології GPS не тільки покращили точність та надійність, але й відкрили нові можливості для розвитку сучасного суспільства, роблячи GPS незамінним інструментом у плануванні маршрутів та багатьох інших аспектах повсякденного життя.

1.1.3 Підйом цифрового картографування

Перехід до цифрової картографії почався у 1950-60-х роках з розвитку комп'ютерних технологій і створення ГІС (геоінформаційних систем). Роджер Томлінсон, відомий як "батько ГІС", відіграв ключову роль у цьому процесі. Запуск першого метеорологічного супутника TIROS-1 у 1960 році та подальший розвиток технологій дистанційного зондування сприяли збору детальних геоданих. У 1980-

90-х роках ці технології стали більш доступними, що призвело до їх широкого використання та інтеграції у різні сфери.

Розвиток цифрових карт почався з появи комп'ютерних технологій у середині 20-го століття. Ранні цифрові карти були в основному простими векторними зображеннями, створеними за допомогою комп'ютерної графіки. Одним з перших прикладів використання цифрових технологій у картографії був проект, відомий як SYMAP (Synagraphic Mapping System), розроблений у Гарвардському університеті в 1960-х роках.

Важливим кроком у розвитку цифрової картографії було створення ГІС. Перші системи, такі як Канадська ГІС (CGIS), розроблені у 1960-х роках, дозволяли збирати, зберігати, аналізувати та візуалізувати просторові дані на небаченому раніше рівні. Ще одним важливим моментом став розвиток програмного забезпечення AutoCAD компанії Autodesk у 1982 році, яке значно спростило створення цифрових карт та планів.

У 1990-х роках з появою Інтернету стали доступними перші онлайн-карти. Одним з перших значних онлайн-сервісів картографування був MapQuest, запущений у 1996 році. Він дозволив користувачам переглядати карти та отримувати напрямки в мережі. А вже введення Google Maps у 2005 році стало революційним моментом у цифровому картографуванні. З його появою користувачі отримали доступ до детальних, інтерактивних карт з можливістю зуму, пошуку місць та отримання маршрутних напрямків.

Заснований у 2004 році, OpenStreetMap став прикладом краудсорсингового підходу до створення карт, де користувачі з усього світу додають та редагують дані.

З появою смартфонів та мобільних додатків сталася значна трансформація в підходах до оптимізації маршрутів подорожей. Смартфони, оснащені GPS та іншими технологіями геолокації, відіграли революційну роль, роблячи навігацію в реальному часі широко доступною та зручною. Мобільні додатки забезпечують не тільки базову навігацію, але й розширені функції, як-от планування маршрутів, інтеграція з соціальними та комунікаційними платформами, а також доступ до інформації про трафік, погодні умови, інтереси користувача та інші аспекти

подорожі. Це сприяло значному поліпшенню ефективності та персоналізації планування подорожей, зміцнивши взаємозв'язок між технологією та користувачем. Завдяки цим інноваціям, користувачі отримали можливість легко адаптувати свої маршрути під особисті потреби і переваги, враховуючи змінні умови в дорозі.

Збір даних та можливості спільноти в мобільних додатках, таких як Waze, радикально змінили підходи до оптимізації маршрутів. Ці додатки використовують дані, зібрані спільнотою користувачів, для надання інформації про рух, затори, ДТП, дорожні роботи та інші фактори, які впливають на рух. Користувачі активно діляться оновленнями в реальному часі, що дозволяє додаткам надавати більш точні та актуальні маршрути. Ця інтерактивність сприяє створенню динамічної карти дорожньої ситуації, що не тільки покращує планування маршрутів, але й сприяє більшій безпеці та комфорту в дорозі. Завдяки цьому підходу, додатки можуть швидко реагувати на зміни у дорожніх умовах, пропонуючи альтернативні маршрути та допомагаючи уникнути непередбачуваних затримок.

1.2 Веб-додатки для планування маршрутів: Огляд та аналіз

У сучасному світі ритм життя постійно набуває темпу, і швидкість переміщення стає важливою складовою щоденного існування. Незалежно від того, чи вирушає людина в подорож на вихідні, відправляється на роботу, чи здійснюються логістичні операції в бізнесі, ефективне планування маршруту стає ключовою задачею. І саме тут на допомогу приходять веб-додатки для планування маршрутів.

Веб-додатки для планування маршрутів представляють собою потужні інструменти, які поєднують в собі сучасні технології, інформаційні ресурси та алгоритми штучного інтелекту для створення оптимальних маршрутів для подорожей, доставки товарів та багатьох інших сфер життя. Вони змінюють бачення навігації, роблять її більш зручною, надійною та інтерактивною.

У цьому об'ємному огляді будуть ретельно розглянуті різні аспекти веб-додатків для планування маршрутів, починаючи з їхньої історії та розвитку, будуть

аналізовані їхня роль у сучасному світі та величезний вплив на повсякденне життя. Також будуть розглянуті ключові функції, переваги та недоліки цих додатків, а також їхня роль у таких галузях, як логістика, громадський транспорт, особиста навігація та інші.

Веб-додатки для планування маршрутів стали невід'ємною частиною нашого життя, і, розвиваючись разом із технологіями, вони продовжують спрощувати і полегшувати переміщення. Цей огляд допоможе зрозуміти, як вони працюють, які можливості вони надають та як вони впливають на різні сфери сучасного існування. Нехай цей огляд стане вашим керівником у світі веб-додатків для планування маршрутів і розкриє перед вами всю їхню розмаїтість і важливість.

Веб-додатки для планування маршрутів є невід'ємною частиною сучасного світу, який пропонує безліч можливостей для ефективної навігації та оптимізації переміщень. Ці додатки стали надійними помічниками як в повсякденному житті, так і у бізнесі, де швидкі та оптимальні маршрути мають вирішальне значення. У цьому огляді ми розглянемо ключові аспекти веб-додатків для планування маршрутів, їхню функціональність, переваги та недоліки.

1.2.1 Функціональність веб-додатків для планування маршрутів

Ось перелік та короткий опис функціоналу веб додатків для планування маршрутів:

1. Пошук оптимального маршруту: Веб-додатки дозволяють користувачам знайти найкоротший, найшвидший або найбільш оптимальний маршрут в залежності від їхніх потреб.

2. Інтерактивні карти: Користувачі можуть використовувати інтерактивні карти, на яких відображені заплановані маршрути, точки і об'єкти інтересу.

3. Навігація в реальному часі: Деякі веб-додатки надають можливість навігації в режимі реального часу, надаючи користувачам оновлення про трафік, перешкоди та інші фактори.

4. Мультимодальні маршрути: Вони дозволяють користувачам вибирати різні види транспорту, включаючи автомобілі, громадський транспорт, велосипеди та піші прогулянки.

5. Оцінка витрат: Деякі додатки розраховують витрати на пальне, платні дороги та інші витрати, пов'язані з маршрутом.

6. Оптимізація маршруту для доставки: У бізнесі веб-додатки допомагають оптимізувати маршрути доставки, що дозволяє зекономити час та ресурси.

7. Інтеграція з GPS: Деякі додатки можуть інтегруватися з GPS-пристроями для точної навігації.

Переваги веб-додатків для планування маршрутів:

1. Зручність: Веб-додатки доступні на різних пристроях і можна використовувати в будь-якому місці і часі.

2. Ефективність: Вони дозволяють знайти найкращий маршрут, що зменшує витрати часу і ресурсів.

3. Інтерактивність: Інтерактивні карти та навігація в реальному часі полегшують процес переміщення та навігації.

4. Оптимізація: У бізнес-сфері веб-додатки допомагають оптимізувати логістичні процеси та витрати.

5. Інтеграція з іншими сервісами: Деякі додатки інтегруються з іншими сервісами, такими як бронювання готелів або ресторанів.

Недоліки веб-додатків для планування маршрутів:

1. Залежність від мережі: Деякі додатки вимагають доступу до Інтернету, що може бути обмежене в окремих місцях.

2. Приватність і безпека: Використання веб-додатків може ставити питання щодо приватності та безпеки ваших даних та місцезнаходження.

3. Оновлення та сумісність: Деякі додатки вимагають постійних оновлень та можуть бути несумісними з деякими пристроями чи операційними системами.

4. Обмеження функцій у безкоштовних версіях: Деякі додатки надають базовий функціонал у безкоштовних версіях, а розширений функціонал доступний за плату.

1.2.2 Популярні веб-додатки для планування маршрутів

З дослідження в інтернеті та з власного досвіду можна зробити список найпопулярніших веб та мобільних додатків для планування маршрутів:

1. Google Maps*: Один з найпопулярніших і широко використовуваних сервісів, який має великий функціонал та доступний безкоштовно.

2. Waze: Спеціалізується на навігації та наданні інформації про трафік в реальному часі.

3. MapQuest: Надає розширені можливості для планування маршрутів та індивідуалізації.

4. HERE WeGo: Дозволяє планувати маршрути для автомобілів, велосипедів, громадського транспорту та пішоходів, навігація доступна в режимі офлайн.

5. Sygic: Спеціалізується на автомобільній навігації та має додаткові функції, такі як оповіщення про камери швидкості.

6. HERE Routing: Використовується у бізнес-середовищі для оптимізації маршрутів доставки та логістики.

MapQuest — це добре зарекомендувала себе програма для карт і навігації, спрямована на надання точних і надійних маршрутів. Він пропонує повний набір функцій, включаючи покрокові вказівки, голосові підказки, зображення вулиць, супутникові зображення, а також місцеві підприємства та визначні місця. MapQuest також надає дані про дорожній рух у реальному часі та звіти про дорожні події. Однак MapQuest має менше функцій спільноти, ніж деякі інші програми, які ми порівнювали, і його загальна точність може бути не такою хорошою, як Google Maps або Apple Maps.

HEREWeGo — це ще одна популярна програма для карт і навігації, яка пропонує широкий спектр функцій, зокрема голосові підказки, 3D-види, покрокові вказівки, вказівки щодо смуг руху, звіти про дорожні події, дані про дорожній рух у реальному часі, супутникові зображення та інформацію про місцеві підприємства. і визначні місця. HEREWeGo також надає маршрути громадського транспорту, маршрути для велосипедистів і пішки. HEREWeGo є особливо сильним з точки зору

даних про дорожній рух у реальному часі, і він має велику спільноту користувачів, які роблять свій внесок у його карти та дані про дорожній рух.

Карти Google є найпопулярнішою програмою для карт і навігації у світі, і це не дарма. Він пропонує повний набір функцій, включаючи покрокові вказівки, голосові підказки, 3D-види, зображення рівня вулиць, супутникові зображення, місцеві підприємства та визначні місця, маршрути громадського транспорту, велосипедні та пішохідні маршрути. Карти Google також надають дані про дорожній рух у реальному часі, звіти про дорожні події та можливість ділитися своїм приблизним прибуттям з іншими. Карти Google постійно оновлюються новими функціями та вдосконаленнями, і це найточніша та найнадійніша програма в нашому порівнянні.

Apple Maps — це вбудована програма для карт і навігації для пристроїв Apple, і це надійний варіант для користувачів, які шукають просту та легку у використанні програму. Він пропонує покрокові вказівки, голосові підказки, 3D-види, зображення рівня вулиць, супутникові зображення, місцеві підприємства та визначні місця, а також маршрути громадського транспорту. Карти Apple також надають дані про дорожній рух у реальному часі, звіти про дорожні події та можливість ділитися вашим приблизним прибуттям з іншими. Однак Карти Apple можуть бути не такими точними, як Карти Google або HERE WeGo, і не мають стільки функцій спільноти.

Waze – це програма для карт і навігації, створена спільнотою, яка відома своїми даними про дорожній рух у реальному часі та звітами про події. Він використовує краудсорсингову інформацію від інших користувачів Waze, щоб надавати найновішу інформацію про затори, аварії та інші небезпеки. Waze також пропонує покрокові вказівки, голосові підказки та підказки щодо смуг руху. Однак Waze може бути не таким точним для далеких поїздок, як Google Maps або Apple Maps, і він не має стільки функцій.

У Табл. 1.1 наведені результату аналізу з характеристиками п’яти різних мобільних додатків для навігації.

Таблиця 1.1

Порівняння характеристик найпопулярніших додатків

Характеристика	**MapQuest**	**HERE WeGo**	**Google Maps**	**Apple Maps**	**Waze**
Голосове керівництво	Так	Так	Так	Так	Так
3D-види	Так	Так	Так	Так	Так
Зображення на рівні вулиць	Ні	Так	Так	Так	Ні
Поворот-за-поворотом напрямки	Так	Так	Так	Так	Так
Повідомлення про інциденти на дорозі	Так	Так	Так	Так	Так
Керівництво по смугах	Так	Так	Так	Так	Так
Повідомлення про інциденти на дорозі	Так	Так	Так	Так	Так
Дані про трафік в реальному часі	Так	Так	Так	Так	Так

Продовження таблиці 1.1

Характеристика	**MapQuest**	**HERE WeGo**	**Google Maps**	**Apple Maps**	**Waze**
Попередження про затори	Так	Так	Так	Так	Так
Швидкісний вимірювач	Ні	Ні	Так	Так	Так
Уникнення трафіку	Так	Так	Так	Так	Так
Краудсорсинг інцидентів на дорозі	Так	Так	Так	Так	Так
Зображення з супутника	Так	Так	Так	Так	Так
Фотографії на рівні вулиць	Ні	Так	Так	Так	Ні
Пошук за адресою, ім’ям або ключовим словом	Так	Так	Так	Так	Так
Напрямки громадського транспорту	Так	Так	Так	Так	Обмежено

Продовження таблиці1.1

Характеристика	**MapQuest**	**HERE WeGo**	**Google Maps**	**Apple Maps**	**Waze**
Напрямки для велосипедистів	Так	Так	Так	Так	Так
Напрямки для пішоходів	Так	Так	Так	Так	Так
Спільнотні внески	Обмежено	Так	Так	Обмежено	Так
Спільнотні оновлення про трафік	Так	Так	Так	Так	Так
Соціальні функції	Обмежено	Обмежено	Обмежено	Обмежено	Так

Всі переваги та недоліки були виведені в Табл. 1.2

Таблиця 1.2

Переваги та недоліки найпопулярніших додатків

Назва програми	**Переваги**	**Недоліки**
MapQuest	Комплексні можливості, точні напрямки	Обмежені можливості спільноти
HERE WeGo	Дані про трафік в реальному часі, велика спільнота	Не так точний, як Google Maps

Продовження таблиці 1.2

Назва програми	**Переваги**	**Недоліки**
Google Maps	Найточніший та надійний	Може бути складним у використанні
Apple Maps	Простий та легкий у використанні	Не так точний, як Google Maps або HERE WeGo
Waze	Дані про трафік в реальному часі, залежить від спільноти	Не такий точний для довгих поїздок

Загалом Google Maps є найкращим вибором для більшості користувачів завдяки своїй точності, повноті та популярності. Однак MapQuest і HERE WeGo також є хорошими варіантами для користувачів, які шукають надійні навігаційні програми з фокусом на даних про дорожній рух у реальному часі. Apple Maps — хороший вибір для користувачів, які шукають просту та легку у використанні програму, а Waze — хороший вибір для користувачів, які надають перевагу даним про дорожній рух у реальному часі та функціям спільноти.

1.3 Алгоритми та підходи у плануванні оптимальних маршрутів

Планування оптимальних маршрутів є важливою задачею в різних сферах, включаючи транспорт, логістику, географічну інформаційну систему та багато інших. Для досягнення цієї мети існують різні алгоритми та підходи, які допомагають знайти оптимальний маршрут в залежності від різних умов та обмежень. Нижче буде розглянуто деякі з найбільш поширених алгоритмів та підходів у плануванні оптимальних маршрутів.

1.3.1 Алгоритм Дейкстри

Алгоритм Дейкстри — це алгоритм пошуку графа в інформатиці для пошуку найкоротшого шляху між двома вузлами (вершинами) у графі з невід'ємними вагами ребер. Він був запропонований Едсгером В. Дейкстрою в 1959 році і є одним із найвідоміших і найефективніших алгоритмів пошуку графів.

Алгоритм працює, зберігаючи пріоритетну чергу вузлів, які зараз досліджуються. Пріоритетна черга впорядковується за відстанню до початкового вузла, причому найближчий вузол має найвищий пріоритет. Алгоритм ітеративно видаляє вузол із найвищим пріоритетом із черги, позначає його як досліджений, а потім досліджує його невідвіданих сусідів. Для кожного невідвіданого сусіда алгоритм обчислює найкоротший шлях від початкового вузла до сусіда та, якщо необхідно, оновлює чергу пріоритетів. Цей процес триває до тих пір, поки не будуть досліджені всі вузли, після чого буде відомий найкоротший шлях до вузла призначення.

Алгоритм Дейкстри є прикладом жадібного алгоритму, який означає, що він приймає локально оптимальні рішення в надії знайти глобально оптимальне рішення. У випадку алгоритму Дейкстри локально оптимальним рішенням є завжди досліджувати вузол, найближчий до початкового вузла. Ця стратегія добре працює, оскільки гарантує, що алгоритм завжди досліджує найкоротший можливий шлях до нового вузла.

Алгоритм Дейкстри ефективний, оскільки він досліджує лише ті вузли, які необхідні для пошуку найкоротшого шляху. Це пов'язано з тим, що алгоритм підтримує чергу пріоритетів, яка визначає пріоритети вузлів на основі їх відстані до початкового вузла. Як наслідок, алгоритм досліджує лише ті вузли, які знаходяться на найкоротшому шляху до вузла призначення.

Алгоритм Дейкстри використовується в широкому спектрі програм, включаючи планування маршруту, оптимізацію мережі та ліз соціальних мереж. Це потужний інструмент для розв'язання задач пошуку графів, і завдяки його ефективності він став одним із основних алгоритмів інформатики.

Приклад того, як за допомогою алгоритму Дейкстри можна знайти найкоротший шлях між двома містами на карті. Карту можна представити у вигляді графіка, де вузли представляють міста, а краї — дороги між містами. Вага кожного ребра представляє відстань між двома містами. Алгоритм Дейкстри можна використовувати для пошуку найкоротшого шляху між будь-якими двома вузлами на графі.

Покрокове пояснення того, як працює алгоритм Дейкстри:

1) Ініціалізується пріоритетна черга, щоб містити початковий вузол із відстанню 0.

2)Якщо в пріоритетній черзі є невідвідані вузли:

a. Видаляється вузол із найкоротшою відстанню з черги пріоритету.

b. Познається вузол як відвіданий.

в. Для кожного невідвіданого сусіда вузла:

i. Обчислюється відстань від початкового вузла до сусіднього.

ii. Якщо обчислена відстань менша за поточну відстань сусіда, оновлюється відстань сусіда до розрахованої відстані.

iii. Додається сусід до черги пріоритетів з його оновленою відстанню.

Найкоротший шлях від початкового вузла до вузла призначення можна знайти, простеживши шлях назад від вузла призначення до початкового вузла.

Алгоритм Дейкстри — це потужний і універсальний алгоритм пошуку графів, який можна використовувати для вирішення широкого спектру задач. Це важливий інструмент для будь-якого комп’ютерника, якому потрібно знайти найкоротший шлях між двома точками на графіку.

1.3.2 Алгоритм A*

A* (вимовляється як «А-зірка») — це інформований алгоритм пошуку, який означає, що він враховує вартість досягнення кожного вузла та приблизну вартість досягнення цільового вузла з цього вузла. Це дозволяє A* знаходити найкоротший

шлях між двома вузлами ефективніше, ніж інші алгоритми пошуку, такі як алгоритм Дейкстри, який не враховує розрахункову вартість досягнення цільового вузла.

A* базується на ідеї використання евристичної функції для оцінки вартості досягнення цільового вузла від заданого вузла. Евристична функція — це оцінка справжньої вартості, але вона не обов'язково має бути ідеальною. Фактично, евристична функція, яка не є надто точною, все ще може бути корисною для A*, оскільки вона може більш ефективно направляти пошук до цільового вузла.

Евристична функція використовується для обчислення f-значення вузла, яке є мірою загальної вартості досягнення цільового вузла з цього вузла. Значення f обчислюється як сума значень g і h. G-value — це фактична вартість досягнення вузла від початкового вузла, а h-value — це оціночна вартість досягнення цільового вузла від вузла.

A* потім сортує вузли в черзі пріоритетів на основі їхніх f-значень, при цьому вузли з найнижчими f-значеннями обробляються першими. Це гарантує, що A* спочатку досліджує вузли, які, найімовірніше, ведуть до цільового вузла.

Ось покрокове пояснення того, як працює алгоритм A*:

1. Ініціалізується відкриті та закриті списки. Відкритий список — це пріоритетна черга, яка містить усі вузли, які ще не досліджено. Закритий список - це набір, який містить усі вузли, які вже були досліджені.

2. Вставляється початковий вузол у відкритий список. F-значення початкового вузла дорівнює його g-значенню, яке є відстанню від початкового вузла до самого себе (що дорівнює 0). Значення h — це розрахункова відстань від початкового вузла до цільового вузла.

3. Поки відкритий список не порожній:

   a. Видаляється вузол із найнижчим значенням f із відкритого списку.Це вузол, який є найбільш перспективним для наступного дослідження.

   b. Позначається вузол як досліджений. Цей вузол більше не розглядатиметься.

   в. Для кожного невідвіданого сусіда вузла:

i. Обчислюється g-значення сусіда. Це відстань від початкового вузла до сусіда плюс відстань від вузла до сусіда.

ii. Обчислюється h-значення сусіда. Це розрахункова відстань від сусіда до цільового вузла.

iii. Обчисліюється f-значення сусіда. Це сума g-значення та h-значення.

iv. **Якщо f-значення сусіда менше, ніж його поточне f-значення, оновлюється його f-значення та помістіть у відкритий список. Це гарантує, що вузол буде досліджено, якщо це кращий шлях до мети вузол.

4. Коли цільовий вузол знайдено:

a. Зворотній шлях від цільового вузла до початкового вузла. Це найкоротший шлях від початкового вузла до цільового вузла.

Алгоритм A* — це ефективний і дієвий алгоритм для пошуку найкоротшого шляху між двома вузлами в графі. Він особливо добре підходить для задач, де евристична функція точна, наприклад, планування маршруту та пошуково-рятувальні роботи.

A* — це універсальний алгоритм, який можна використовувати для вирішення широкого спектру завдань, зокрема:

1) Планування маршруту: A* можна використовувати, щоб знайти найкоротший шлях між двома точками на карті.

2) Пошук і порятунок: A* можна використовувати для пошуку загублених людей або предметів.

3) Робототехніка: A* можна використовувати для керування роботами, які об’їжджають перешкоди та досягають місця призначення.

A* — це потужний алгоритм, який можна використовувати для пошуку найкоротшого шляху між двома точками на графіку ефективніше, ніж інші алгоритми пошуку.

1.3.3. Алгоритм Флойда–Воршалла

Алгоритм Флойда–Воршалла — це алгоритм пошуку найкоротших шляхів між усіма парами вузлів у орієнтованому графі. Це вдосконалення алгоритму Дейкстри, який знаходить лише найкоротші шляхи від одного вихідного вузла до всіх інших вузлів у графі.

Алгоритм Флойда–Воршалла працює шляхом ітеративного оновлення найкоротших шляхів між усіма парами вузлів. У кожній ітерації алгоритм розглядає всі можливі шляхи між парою вузлів і оновлює відстань найкоротшого шляху, якщо знайдено коротший шлях. Алгоритм продовжує повторюватися, доки не буде визначено найкоротший шлях між усіма парами вузлів.

Ось покрокове пояснення того, як працює алгоритм Флойда–Воршалла:

1. Ініціалізація матриці відстані: Створюється матриця, де кожен елемент представляє відстань між двома вузлами. Ініціалізується всі елементи з відповідними вагами ребер, якщо ребро існує, і нескінченністю, якщо ребро не існує.

2. Розслабляються краї: Для кожної пари вузлів, якщо прямий край між ними не існує, але існує шлях через інший вузол, який скорочує відстань, відповідно оновіть матрицю відстані.

3. Повторити: Повторюється крок 2, доки матриця відстані не стабілізується, що означає, що більше не буде можливим оновлення.

4. Знаходяться найкоротші шляхи: Після стабілізації матриці відстаней можна знайти найкоротші відстані між усіма парами вузлів шляхом відстеження шляхів назад від вузла призначення до вузла джерела.

Алгоритм Флойда–Воршалла — це ефективний алгоритм для пошуку найкоротших шляхів між усіма парами вузлів у графі. Він особливо добре підходить для графів з негативними вагами ребер, оскільки алгоритм Дейкстри не може працювати з негативними вагами ребер.

Окрім пошуку найкоротших шляхів, алгоритм Флойда–Воршалла також можна використовувати для пошуку:

- Довжина найкоротшого циклу на графіку, якщо такий існує.

- Кількість різних найкоротших шляхів між двома вузлами.

- Найкоротші шляхи всіх пар між усіма парами вузлів, навіть якщо вони не всі з’єднані.

1.3.4 Генетичні алгоритми

Генетичний алгоритм (GA) — це алгоритм пошуку, натхненний процесом природного відбору. Він використовується для пошуку оптимальних рішень широкого кола проблем, включаючи оптимізацію, машинне навчання та інженерний дизайн.

GA працює шляхом ітеративного створення та оцінки сукупності рішень, які називаються хромосомами. Кожна хромосома представляє потенційне рішення проблеми. Алгоритм починається з початкової популяції хромосом, а потім неодноразово застосовує набір генетичних операторів для створення нових популяцій хромосом. Генетичні оператори базуються на принципах природного відбору, таких як відбір, кросинговер і мутація.

- Відбір: для створення нових хромосом відбираються найкращі хромосоми в популяції. Це вибирає найбільш перспективні рішення проблеми, забезпечуючи фокусування алгоритму на перспективних областях простору пошуку.

- Кроссовер: Вибираються дві хромосоми, і їх генетична інформація обмінюється для створення нових нащадків. Це рекомбінує інформацію від двох батьків, що потенційно призводить до нових і кращих рішень.

- Мутація: випадкові зміни вводяться в хромосоми, щоб представити новий генетичний матеріал і запобігти сходженню алгоритму до локального оптимуму.

Алгоритм продовжує генерувати нові популяції хромосом, доки не буде виконано критерій зупинки. Критерієм зупинки може бути максимальна кількість поколінь, певний рівень придатності або відсутність удосконалення найкращих рішень.

Ось покрокове пояснення того, як працює генетичний алгоритм:

1. Ініціалізація популяції: Створення популяції хромосом, представленої у вигляді послідовностей генів. Гени можуть представляти параметри проблеми, або вони можуть представляти різні рішення проблеми.

2. Оцінюється популяцію: Оцінюється кожна хромосома в популяції, щоб визначити її відповідність. Фітнес — це міра того, наскільки добре хромосома вирішує проблему.

3. Вибераються батьки: Виберються дві або більше хромосом із популяції на основі їх відповідності. Найімовірніше, що найбільш підходящі хромосоми будуть обрані, гарантуючи, що алгоритм зосереджується на перспективних областях простору пошуку.

4. Кроссовер: Створення нових нащадків шляхом обміну генетичною інформацією між вибраними батьками. Така рекомбінація інформації може призвести до нових і кращих рішень.

5. Мутувати нащадків: Внести випадкові зміни в хромосоми нащадків. Це може запобігти сходженню алгоритму до локальних оптимумів.

6. Замінити популяцію: Замінити поточну популяцію новим нащадком. Це створює нове покоління хромосом і продовжує пошук кращих рішень.

7. Продовження до припинення: Повторюються кроки 2-6, доки не буде виконано критерій припинення. Критерієм зупинки може бути максимальна кількість поколінь, певний рівень придатності або відсутність удосконалення найкращих рішень.Генетичні алгоритми є потужним інструментом для вирішення широкого кола проблем. Вони особливо добре підходять для проблем, які важко вирішити за допомогою традиційних методів оптимізації.

Ось деякі з переваг генетичних алгоритмів:

1) Їх можна використовувати для вирішення широкого кола проблем.

2) Вони не чутливі до початкових умов.

3) Вони можуть знайти кілька рішень проблеми.

Ось деякі з недоліків генетичних алгоритмів:

1) Вони можуть бути обчислювально дорогими.

2) Їх може бути важко налаштувати.

3) Вони не завжди можуть сходитися до оптимального рішення.

Загалом, генетичні алгоритми є універсальним і потужним інструментом, який можна використовувати для вирішення широкого кола проблем. Вони є цінним доповненням до набору інструментів будь-якого фахівця з оптимізацій.

1.3.5 Алгоритми імітації відпалу

Імітований відпал (SA) — це метаевристичний алгоритм, натхненний процесом відпалу в металургії. Він використовується для знаходження наближених глобальних оптимумів для широкого кола задач оптимізації.SA працює шляхом повторного переходу від одного рішення до іншого, оцінюючи кожне нове рішення, щоб визначити, чи воно краще за поточне. Якщо нове рішення краще, воно приймається. Якщо він гірший, його все одно можна прийняти з ймовірністю, яка залежить від різниці в якості між двома розчинами та параметром температури. Температурний параметр поступово зменшується з часом, що змушує алгоритм досліджувати простір рішень більш агресивно на ранній стадії та ретельніше, коли він наближається до потенційного рішення.

Покрокові дії алгоритму:

1. Ініціалізація температури: Встановлюється параметр початкової температури на високе значення.

2. Виберіть сусіднє рішення: Довільно вибереться сусіднє рішення для поточного рішення.

3. Оцініть сусідній розчин: Обчислюються якість сусіднього розчину.

4. Прийняти або відхилити сусіднє рішення:

   a. Якщо якість сусіднього рішення краща, ніж поточне рішення, приймається сусіднє рішення та встановлюється поточне рішення на сусіднє рішення.

   b. Якщо якість сусіднього розчину гірша за поточне, приймається сусіднє рішення з імовірністю e^((якість поточного розчину - якість сусіднього розчину) / температура).

в. Якщо сусіднє рішення не прийнято, збережіть поточне рішення.

5. Спад температури: Поступово знижуються температурний параметр відповідно до графіка.

6. Продовжується до припинення: Повторюються кроки 2–5, доки не буде виконано критерій припинення. Критерієм зупинки може бути максимальна кількість ітерацій, певний рівень збіжності або відсутність покращення найкращого рішення.

Імітований відпал є потужним алгоритмом для вирішення широкого кола задач оптимізації. Він особливо добре підходить для проблем із багатьма локальними оптимумами, оскільки він може уникнути застрягання в локальних оптимумах і знайти глобальний оптимум.

Переваги моделювання відпалу:

1) Його можна використовувати для вирішення широкого кола проблем.

2) Це дуже ефективно для проблем із багатьма локальними оптимумами.

3) Це відносно легко реалізувати.

Недоліки імітованого відпалу:

1) Це може бути обчислювально дорогим для проблем із багатьма змінними.

2) Може бути складно налаштувати параметр температури.

3) Це не завжди може знайти глобальний оптимум.

Загалом імітований відпал — це універсальний і потужний алгоритм, який можна використовувати для вирішення широкого кола задач оптимізації. Це цінне доповнення до набору інструментів будь-якого спеціаліста з оптимізації.

1.3.6 Алгоритм ATSP

Алгоритм ATSP — це універсальний алгоритм, який можна використовувати для вирішення широкого спектру завдань. Деякі з його поширених застосувань включають:

- Маршрутизація транспортних засобів у транспортній мережі: Алгоритм ATSP можна використовувати для пошуку найкоротших маршрутів для транспортних засобів між різними точками транспортної мережі. Це можна використовувати для таких програм, як служби доставки, служби таксі та громадський транспорт.

- Планування завдань у розподіленій обчислювальній системі: Алгоритм ATSP можна використовувати для планування виконання завдань на різних комп'ютерах у розподіленій обчислювальній системі. Це можна використовувати для підвищення ефективності системи шляхом мінімізації обсягу зв'язку між комп'ютерами.

- Оптимізація логістики та ланцюгів постачання: Алгоритм ATSP можна використовувати для оптимізації логістики та ланцюгів постачання шляхом пошуку найкоротших маршрутів для транспортування товарів між різними місцями. Це може допомогти зменшити витрати та підвищити ефективність.

Алгоритм ATSP є потужним інструментом для вирішення задач оптимізації. Однак важливо зазначити, що алгоритм може бути дорогим у плані обчислень, особливо для великих графіків. Існує ряд евристичних і апроксимаційних алгоритмів, які можна використовувати для більш ефективного вирішення проблеми ATSP, але ці алгоритми не завжди можуть знайти оптимальне рішення.

Сильні сторони:

- Можна використовувати для вирішення широкого спектру проблем
- Може обробляти графіки з негативними вагами ребер
- Можна використовувати для пошуку кількох рішень

Слабкі сторони:

- Для великих графіків може бути дорого за обчисленням
- Не завжди можна знайти оптимальне рішення
- Потрібне матричне представлення графіка

Пояснення алгоритму ATSP із покроковими інструкціями:

1. Ініціалізація матриці відстані:

Створюється матриця відстані D, де D[i][j] представляє найкоротший шлях між вузлом i та вузлом j. Спочатку всі відстані в матриці встановлені на нескінченність, за винятком діагональних елементів, які встановлені на 0. Це означає, що відстань між вузлом і самим собою дорівнює 0.

2. Розслабляються краї:

Перехід по всіх ребрах на графіку та для кожного ребра (i, j) оновлення відстані у матриці відстаней, якщо нова відстань коротша за поточну:

$$D[i][j] = \text{хв}\,(D[i][j], D[i][k] + D[k][j])$$

де k — проміжний вузол між вузлами i та j.

3. Перевірка наявність негативних циклів:

Якщо відстань між вузлом і самим собою стає негативною після оновлення матриці відстані, це вказує на наявність негативних циклів у графі. Негативні цикли можуть призвести до того, що алгоритм ATSP знайде нескінченний цикл, що не є дійсним рішенням. Щоб впоратися з цим, перевіряється наявність негативних циклів за допомогою алгоритму Беллмана-Форда або подібного алгоритму.

4. Витягуються найкоротші шляхи:

Після того, як усі ребра були розслаблені та негативні цикли не виявлені, матриця відстані міститиме найкоротші відстані між усіма парами вузлів. Витягуються найкоротші шляхи з матриці відстані, повертаючись від вузла призначення до вузла джерела, дотримуючись країв мінімальної вартості на кожному кроці.

5. Обробка кілької рішень і обмежень:

Для деяких застосувань може знадобитися розглянути кілька рішень або включити додаткові обмеження, окрім відстаней найкоротшого шляху. Цього можна досягти шляхом модифікації алгоритму ATSP або використання більш спеціалізованого алгоритму, адаптованого до конкретної проблеми.

В цілому, алгоритм ATSP є потужним інструментом для вирішення задач оптимізації. При виборі алгоритму важливо ретельно враховувати обчислювальну складність задачі та бажану точність рішення.

1.3.7 Алгоритм ACO

ACO — це метаевристичний алгоритм, натхненний поведінкою мурах у пошуку їжі. Він був успішно застосований для широкого кола проблем оптимізації, включаючи планування маршруту, планування та розподіл ресурсів.

Алгоритм ACO імітує поведінку мурах у відкладенні слідів феромонів і випаровуванні феромонів, коли вони шукають джерела їжі. Під час руху мурахи виділяють феромони, і ці феромони збільшують ймовірність того, що інші мурахи підуть тим самим шляхом. Концентрація феромонів з часом слабшає, що означає згасання інформації про багатообіцяючі шляхи.

Ключові компоненти алгоритму ACO:

1. Феромонна матриця: Матриця, що представляє графік, де кожна комірка містить значення феромону, яке вказує на бажаність відповідного краю.
2. Правило оновлення феромонів: Правило, яке оновлює значення феромонів у матриці на основі вибору мурашок і отриманої якості шляху.
3. Поведінка мурах: Набір правил, які визначають спосіб переміщення мурашок по графіку, вибираючи ребра на основі значень феромонів і евристичної інформації.
4. Локальний пошук: механізм, що дозволяє мурахам досліджувати альтернативні шляхи, долаючи локальні оптимуми та покращуючи загальне рішення.
5. Випаровування феромонів: процес, який поступово зменшує значення феромонів, що відображає розпад інформації про старі шляхи.

Сильні сторони:

-Стійкий до шумів і збоїв: ACO здатний адаптуватися до мінливих умов і збоїв у навколишньому середовищі, що робить його придатним для застосування в реальних умовах.

- Ефективність для складних проблем: ACO може вирішувати складні проблеми оптимізації з кількома обмеженнями та залежностями.

Обмеження:

- Повільніше, ніж точні алгоритми: ACO зазвичай повільніше, ніж точні алгоритми, як-от алгоритм Дейкстри для пошуку оптимального рішення.

- Може бути чутливим до налаштування параметрів: Продуктивність ACO залежить від значень певних параметрів, і ці параметри можуть вимагати ретельного налаштування для досягнення оптимальної продуктивності.

ACO застосовувався для широкого кола проблем, зокрема:

- Планування: Планування завдань і ресурсів у системах виробництва, обслуговування та охорони здоров'я.

- Розподіл ресурсів: Розподіл ресурсів на діяльність або вимоги.

ACO можна поєднувати з іншими алгоритмами для покращення його продуктивності та вирішення більш складних проблем. Наприклад, ACO можна поєднати з A*, щоб забезпечити гібридний підхід, який поєднує в собі сильні сторони обох алгоритмів.

Покрокові інструкції щодо реалізації алгоритму ACO:

1. Ініціалізація феромонної матриці:

Створюється феромонну матрицю з тими ж розмірами, що й графік, що представляє проблему. Встановлюється значення феромонів на невелике позитивне значення, наприклад 0,01.

2. Створюється початковий набір мурах:

Створюється набір мурах, кожен з яких представлятиме потенційне рішення проблеми. Випадково ініціалізуйте кожну мураху в початковому вузлі на графіку.

3. Повторення наступниї дій для кожної мурашки:

a. Виберіається ребро для проходження:Для кожного сусіднього вузла поточного вузла обчислюється ймовірність переходу для цього краю. Імовірність переходу є зваженою сумою значення феромону на краю та евристичного значення краю. Випадково вибереться ребро відповідно до ймовірностей переходу.

b. Перейти до вибраного вузла: Переміщення мурашки до вибраного сусіднього вузла.

в. Феромон депозиту: Нанесення значення феромону на пройдений край. Значення феромону зазвичай залежить від якості розчину, знайденого мурахою.

4. Оновлення матриці феромонів: Коли всі мурахи пройдуть свої шляхи, оновіть значення феромонів у матриці на основі якості шляхів, знайдених мурахами. Як правило, значення феромонів оновлюються за такою формулою:

$$\tau(i,j) == (1 - \rho) * \tau(i,j) + \Delta\tau(i,j)$$

де:

- τ(i, j) – значення феромону на краю (i, j) в оновленій матриці
- ρ – швидкість випаровування феромону
- Δτ(i, j) – феромон, який осідає мурахами на краю (i, j)

5. Повторення кроків 2-4 для вказаної кількості ітерацій: Матриця феромонів поступово спрямовуватиме мурах до кращих рішень у міру проходження ітерацій.

6. Вибереться найкраще рішення: Після того, як алгоритм завершить задану кількість ітерацій, найкраще рішення, знайдене будь-яким із мурах, може бути обране як остаточне рішення.

7. Локальний пошук: Для подальшого вдосконалення рішення алгоритм локального пошуку може бути застосований до найкращого рішення, знайденого алгоритмом ACO. Це може допомогти уникнути локальних оптимумів і знайти ще кращі рішення.

8. Впровадити гібридні підходи: ACO можна поєднувати з іншими алгоритмами оптимізації, такими як алгоритм A*, для створення гібридних підходів, які поєднують сильні сторони обох алгоритмів. Це може призвести до більш ефективних і ефективних рішень.

Поточні дослідження зосереджені на розробці більш ефективних алгоритмів ACO, покращенні їх здатності вирішувати великі та складні проблеми та розробці гібридних підходів з іншими методами оптимізації.

Ці алгоритми та підходи представляють лише деякі з можливих методів планування оптимальних маршрутів. Вибір конкретного алгоритму залежить від характеру задачі, вимог до точності, обмежень та інших факторів. У сучасних системах планування маршрутів також можуть використовуватися комбінації різних алгоритмів для досягнення оптимальних результатів.

1.4 Дизайн з орієнтацією на користувача та інтерфейс у подорожніх додатках

Дизайн та інтерфейс у подорожніх додатках відіграють важливу роль у створенні зручного та задовільного досвіду для користувачів. Подорожі завжди були і залишаються популярним видом відпочинку та ділових поїздок, і в сучасному світі маємо доступ до різноманітних технологічних рішень для їх планування та виконання. Завдяки розвитку мобільних додатків, сьогодні можна легко планувати подорожі, бронювати готелі, купувати квитки та отримувати інші корисні послуги. Проте успіх таких подорожніх додатків значною мірою залежить від того, наскільки добре вони призначені для користувача та наскільки зручним та інтуїтивним є їхні інтерфейси. Дизайн, орієнтований на користувача, передбачає створення продукту, який враховує потреби, очікування та звички користувачів.

Інтуїтивність та простота є першими та найважливішими принципами дизайну. Інтерфейс додатку має бути настільки простим та зрозумілим, що користувачам не потрібно докладати надмірних зусиль для того, щоб зрозуміти, як

ним користуватися. Всі елементи та функції повинні бути розташовані логічно та послідовно.

Уникається перевантажений інтерфейс та мінімізується кількість надмірних кнопок та тексту, щоб спростити користування додатком. Особливу увагу слід приділяти навігації - меню та структура додатку повинні бути легкими у використанні, і важливі функції повинні бути доступні на перший погляд.
Персоналізація та рекомендації грають також важливу роль у дизайні подорожніх додатків. Користувачі люблять, коли додаток адаптується до їхніх уподобань і потреб. Можливість налаштовувати досвід подорожі, отримувати рекомендації та персоналізовану інформацію - це ключ до задоволення користувачів.

У подорожніх додатках важливо використовувати інтерактивні карти та візуалізацію даних. Користувачі часто використовують карти для планування маршрутів та пошуку цікавих місць. Інтерактивність карт, можливість збільшення та зменшення масштабу, а також маркування цікавих точок роблять користування додатком більш зручним і цікавим.

Додавання фотографій та зображень до інформації про місця та об'єкти інтересу допомагає користувачам краще ознайомитися з ними та зробити більш інформований вибір.

Зручність для мобільних пристроїв є також ключовим аспектом дизайну. Більшість користувачів використовують подорожні додатки на мобільних пристроях, тому важливо забезпечити зручність на малих екранах. Адаптивний дизайн, який підлаштовується під різні розміри екранів та орієнтацію, допомагає забезпечити зручність користувачам.

Офлайн доступ - це ще один важливий аспект дизайну подорожніх додатків. Користувачам може знадобитися використовувати додаток без доступу до Інтернету, особливо під час подорожі. Тому додавання можливості завантажувати дані для офлайн-використання є доречним рішенням.

Безпека та приватність - це критично важливі аспекти дизайну. Захист особистих даних користувачів та їхньої геолокації є обов'язковим завданням.

Користувачі повинні мати можливість контролювати свої налаштування приватності та доступ до геолокації.

Для глобального успіху додатку, підтримка іншомовних користувачів та різних культур - це важливий аспект дизайну. Додається можливість використання різних мов та культур у додатку.

Також необхідно надавати підтримку користувачам та можливість отримання допомоги та інструкцій у випадку питань або проблем. Введення чат-підтримки допомагає вирішувати питання у режимі реального часу.

Тестування з користувачами допомагає виявити незручності та проблеми в дизайні та інтерфейсі та виправити їх до випуску продукту.

Узагальнюючи, дизайн та інтерфейс у подорожніх додатках повинні бути орієнтовані на користувача і враховувати їхні потреби, очікування та зручність. Від якості дизайну значною мірою залежить популярність та успіх таких додатків на ринку. Інтуїтивність, персоналізація, візуалізація даних, оцінки користувачів та багато інших аспектів грають важливу роль у створенні досконалого подорожнього додатка.

РОЗДІЛ 2. ОГЛЯД МЕТОДОЛОГІЙ

Огляд методологій відіграє важливу роль у забезпеченні фундаментального розуміння різних підходів до управління проектами. Цей розділ має на меті охопити широкий спектр методологій, аналізуючи їх історію, ключові концепції, особливості, а також сценарії застосування. Від традиційних методів, як-от Waterfall, до більш адаптивних та гнучких підходів, таких як Agile, Scrum чи Lean, кожна методологія розглядається через призму її внеску в ефективність проектного управління. Крім того, обговорення включатиме вплив цих методологій на різні аспекти проекту, від планування та реалізації до контролю якості та оцінки ризиків.

2.1 Методологія збору даних та розробки програмного забезпечення

Збір даних є ключовим етапом в розробці веб-додатку для оптимізації маршрутів подорожей. Для досягнення успіху в даному проєкті необхідно глибше розуміти потреби користувачів, їх очікування та унікальні вимоги до продукту. Для цього ми використовуємо ефективні методи збору даних, які дозволяють нам отримувати цінну інформацію для подальшого розроблення та вдосконалення веб-додатку для оптимізації маршрутів подорожей.

У цьому розділі розглядаються різноманітні методи збору даних, які надають цінну інформацію для подальшого розроблення та вдосконалення веб-додатку для оптимізації маршрутів подорожей. Розглянуті як кількісні, так і якісні підходи до збору даних, а також способи аналізу та інтерпретації отриманої інформації. Використання різноманітних методів дозволяє створити продукт, який відповідає реальним потребам користувачів і вигідно виділяється на ринку подорожніх додатків.

2.1.1 Методи збору даних

Незалежно від того, чи є досвід у розробці аналогічних додатків, чи це перший крок у цій галузі, правильно обрані методи збору даних визначать успіх вашого проєкту та спрямують роботу на правильний шлях. Розглянуті методи надають необхідні інструменти для збору об'єктивної та цінної інформації, яка буде в основі нашого дизайну та функціональності веб-додатку.

1. Опитування користувачів:

- Створення опитника з питаннями про поточні проблеми та вимоги користувачів у сфері маршрутів подорожей.

- Розповсюдження опитника серед цільової аудиторії через веб-сайт, соціальні мережі або електронну пошту.

- Аналіз отриманих відповідей та виділення ключових вимог та проблем користувачів.

2. Спостереження та аналіз поведінки користувачів:

- Встановлення аналітичних інструментів на веб-сайті або додатку для відстеження поведінки користувачів.

- Аналіз даних про взаємодію користувачів з додатком, таких як сторінки, які вони відвідують, функції, які використовують, та тривалість сесій.

- Виявлення основних шляхів користувачів та можливих точок труднощі.

3. Інтерв'ю з користувачами:

- Проведення інтерв'ю з представниками цільової аудиторії, щоб зрозуміти їхні потреби, думки та вимоги.

- Створення структурованих питань або тематичних інтерв'ю, щоб детальніше розібратися у вимогах користувачів.

4. Аналіз конкурентів:

- Вивчення конкурентів та їхніх веб-додатків для оптимізації маршрутів подорожей.

- Порівняння функціональності та можливостей конкурентів з тим, що планується розробити.

- Виявлення переваг та недоліків конкурентів та визначення можливостей для покращення.

5. Фокус-групи:

- Складання групи представників цільової аудиторії та проведення фокус-груп для обговорення їхніх потреб та очікувань.

- Використання модератора для керування обговоренням та реєстрації відгуків учасників.

6. Аналіз відгуків та рейтингів:

- Моніторинг відгуків та рейтингів користувачів в інтернет-магазинах, маркетплейсах або соціальних мережах, якщо аналогічні додатки вже існують на ринку.

- Звертання уваги на позитивні та негативні відгуки, а також на конкретні пропозиції та скарги користувачів.

7. Аналіз анкет та звітів:

- Вивчення попередніх аналітичних звітів, опитників та досліджень, які проводилися в галузі оптимізації маршрутів подорожей.

- Використання існуючих даних для порівняльного аналізу та виявлення наявних тенденцій.

8. Внутрішні дані та статистика:

- Використання внутрішніх даних та статистики з попередніх версій додатку, якщо такі є.

- Аналіз використання функцій та поведінки користувачів у попередніх версіях, для розуміння, які аспекти можуть бути покращені.

Збір даних може бути комбінованим, і важливо підібрати методи, які найкраще відповідають поставленним цілям та можливостям. Після збору даних вони повинні бути аналізовані та інтерпретовані для формування вимог та рекомендацій для подальшого розроблення веб-додатку для оптимізації маршрутів подорожей.

2.1.2 Методологія розробки програмного забезпечення

Розробка програмного забезпечення є сучасним інженерним процесом, який вимагає високої організації, методичності та системності для досягнення успішних результатів. У даному розділі ми глибше дослідимо методології розробки програмного забезпечення, їхні концепції та практичні застосування.

Сучасний світ неможливо уявити без програмного забезпечення, яке використовується в усіх сферах життя: від комунікації та розваг до бізнес-процесів та наукових досліджень. Програмне забезпечення виконує різноманітні завдання, спрощуючи та автоматизуючи багато процесів. Однак його розробка - це складний та багатоетапний процес, який потребує глибокого розуміння завдань, технічної компетенції та правильного керування.

Методологія розробки програмного забезпечення - це систематичний підхід до планування, реалізації та управління процесом створення програмних продуктів. Вона визначає порядок виконання завдань, ролі учасників, інструменти та практики, необхідні для досягнення мети. Вибір методології розробки впливає на

якість, швидкість та вартість розробки, тому це важливий аспект у процесі роботи над програмними проєктами.

У цьому розділі буде проаналізован світ методологій розробки програмного забезпечення та розглянемо різні підходи до організації розробки проектів. Буде розкрита суть традиційних методологій, таких як Waterfall та V-модель, а також вивчино сучасні та гнучкі підходи, такі як Agile та Scrum. Також проведені дослідження щодо основних принципів кожної методології, їхні переваги та недоліки, а також сфери застосування.

Правильний вибір методології розробки визначає успіх проєкту, забезпечуючи ефективне управління ресурсами, збільшення продуктивності та досягнення бажаних результатів. Крім того, повинні бути розглянуті практичні аспекти впровадження методологій розробки, включаючи інструменти для керування проектами та спільною роботою, а також підходи до оцінки якості програмного забезпечення.

Мета цього розділу – отримати чітке розуміння різних методологій розробки програмного забезпечення та їхню роль у сучасній індустрії. Також допоможе визначити, яка методологія найкраще підходить для конкретного проєкту, враховуючи його характеристики та завдання. Безсумнівно, знання методологій розробки є важливим інструментом для кожного професіонала в галузі програмного забезпечення, незалежно від рівня досвіду.

2.1.3 Waterfal

Методологія Waterfall — це традиційний підхід до розробки програмного забезпечення, який слідує за лінійним послідовним процесом. Він ділить проект на окремі етапи:

1. Збір вимог: Проект починається зі збору та документування потреб і очікувань зацікавлених сторін. Цей етап передбачає розуміння проблеми, яку потрібно вирішити, визначення цільової аудиторії та окреслення бажаних функцій програмного забезпечення.

2. Проектування: На основі зібраних вимог етап проектування зосереджується на створенні детального плану програмної системи. Це включає визначення архітектури, інтерфейсу користувача, структур даних і алгоритмів. Чітка та вичерпна документація є важливою на цьому етапі.

3. Розробка: Після завершення розробки починається фаза розробки, на якій відбувається фактичне впровадження програмного забезпечення. Розробники пишуть код, ітеративно тестуючи кожен модуль, щоб переконатися в функціональності та дотриманні специфікацій дизайну.

4. Тестування: Ретельне тестування є невід'ємною частиною методології Waterfall. Етап тестування включає кілька рівнів тестування, починаючи з модульних тестів для перевірки окремих компонентів і переходячи до інтеграційних тестів, системних тестів і приймальних тестів, щоб переконатися, що система відповідає загальним вимогам.

5. Розгортання: Останній етап, розгортання, передбачає розміщення розробленого програмного забезпечення у виробництві, що робить його доступним для користувачів. Це включає в себе налаштування необхідної інфраструктури, налаштування процедур розгортання та забезпечення повної інтеграції з існуючими системами.

Методологія Waterfall пропонує кілька переваг:

- Структура та чіткість: лінійний підхід забезпечує чітку дорожню карту для проекту, гарантуючи, що кожна фаза завершена перед переходом до наступної. Це сприяє організованості та зменшує ризик пропущених термінів або розповзання обсягу.

- Акцент на документацію: Методологія Waterfall наголошує на комплексній документації протягом життєвого циклу проекту. Ця документація служить орієнтиром для зацікавлених сторін, розробників і майбутніх команд з обслуговування.

- Передбачуваність і контроль: модель Waterfall забезпечує передбачуваний графік завершення проекту, що дозволяє краще розподіляти ресурси та оцінювати витрати.

Однак методологія Waterfall також має деякі обмеження:

- Негнучкість: лінійна природа моделі водоспаду ускладнює внесення змін під час проекту, оскільки зміни можуть вимагати переробки попередніх етапів.
- Повільні цикли доставки: послідовний підхід може призвести до більш тривалих циклів розробки, особливо для складних проектів із кількома ітераціями.
- Труднощі з адаптацією до мінливих вимог: Жорстка структура Waterfall може важко адаптуватися до мінливих ринкових умов або потреб зацікавлених сторін.

Підводячи підсумок, методологія Waterfall — це структурований підхід, який добре підходить для чітко визначених проектів зі стабільними вимогами та чітким обсягом. Його акцент на документації та передбачуваних термінах робить його придатним для проектів із чіткими цілями та обмеженими очікуваннями змін. Однак для проектів з динамічними вимогами або потребою у швидкому прототипуванні методології Agile можуть бути більш доцільними.

2.1.4 Agile

Гнучкі методології пропонують гнучкий і адаптований підхід до розробки програмного забезпечення. На відміну від лінійної моделі Waterfall, методології Agile розбивають проекти на менші ітераційні цикли, відомі як спринти. Кожен спринт зосереджується на наданні певного набору функцій, що дозволяє швидко створювати прототипи, часті відгуки та постійне вдосконалення.

Ключові принципи гнучких методологій:

1. Клієнтська орієнтація: Гнучкі методології підкреслюють тісну співпрацю з клієнтами протягом усього процесу розробки. Відгуки користувачів збираються завчасно та часто, щоб гарантувати, що програмне забезпечення відповідає реальним потребам і очікуванням.

2. Ітеративна розробка: Проекти розбиваються на короткі спринти, кожен з яких забезпечує приріст робочого продукту. Цей ітеративний підхід забезпечує швидкий зворотний зв'язок і адаптацію до мінливих вимог.

3.Команди, що самоорганізуються: Гнучкі команди самокеровані, уповноважені приймати рішення та адаптуватися до мінливих ситуацій. Це сприяє автономії та заохочує робоче середовище для співпраці.

4. Регулярне обмірковування: Гнучкі команди проводять регулярні ретроспективи, щоб переглянути свій прогрес, визначити області для вдосконалення та вдосконалити свої процеси. Це сприяє постійному навчанню та вдосконаленню.

Поширені гнучкі методики:

1. Scrum: Scrum — це популярна гнучка структура, яка наголошує на спринтах, щоденних зустрічах та історіях користувачів, щоб спрямовувати розвиток.

2. Kanban: Kanban фокусується на візуалізації робочого процесу та керуванні завданнями за допомогою візуальної дошки. Він підкреслює безперервний потік і мінімізує незавершену роботу.

3. Екстремальне програмування (XP): XP — це гнучка методологія, відома своїм акцентом на тестуванні, парному програмуванні та безперервній інтеграції. Він зосереджений на постачанні високоякісного програмного забезпечення на ранній стадії та вчасно.

Переваги гнучких методологій:

1. Швидке створення прототипів і зворотній зв’язок: Гнучкі методології забезпечують швидке створення прототипів і часті відгуки від користувачів, дозволяючи раннє виявлення проблем і постійне вдосконалення продукту.

2. Пристосовуваність до змін: Гнучкі команди здатні впоратися зі змінами та адаптуватися до мінливих вимог, завдяки чому вони добре підходять для динамічних проектів.

3. Підвищена задоволеність клієнтів: Зосередженість Agile на залученні клієнтів і швидкому зворотному зв’язку підвищує ймовірність відповідати очікуванням користувачів і досягати задоволеності клієнтів.

4. Підвищення морального духу команди: Акцент Agile на самоорганізації, співпраці та постійному вдосконаленні сприяє створенню позитивного та активного робочого середовища.

Проблеми гнучких методологій:

1. Потрібна міцна комунікація: Ефективний Agile вимагає чіткої та прозорої комунікації між членами команди та зацікавленими сторонами.

2. Потенціал розповзання масштабу: Акцент Agile на адаптації до змін може призвести до розповзання масштабу, якщо ним не керувати ефективно.

3. Потрібна дисципліна та відданість: Гнучкі команди повинні підтримувати дисципліну та відданість ітераційному процесу та постійному вдосконаленню.

4. Може підходити не для всіх проектів: Agile може не підходити для проектів із дуже складними або жорсткими вимогами.

Підсумовуючи, Agile-методології пропонують гнучкий і адаптований підхід до розробки програмного забезпечення, завдяки чому вони добре підходять для динамічних проектів із змінними вимогами. Їх наголос на клієнтоорієнтованості, ітерації та постійному вдосконаленні може призвести до високоякісного програмного забезпечення, задоволених клієнтів і залучених команд. Однак Agile також вимагає міцного спілкування, дисципліни та відданості ітераційному процесу. Ретельний розгляд характеристик проекту та динаміки команди має важливе значення, коли вирішуєте, чи підійде методологія Agile.

2.1.5 V-модель

V-модель, також відома як V-цикл, — це методологія розробки програмного забезпечення, яка наголошує на структурованому та дисциплінованому підході до

розробки програмного забезпечення. Це розширення традиційної моделі Waterfall, яка відома своїм лінійним послідовним процесом.

Ключові характеристики V-моделі:

1. Паралельна розробка та тестування: V-модель розбиває процес розробки на окремі фази та вводить відповідні фази тестування, які виконуються паралельно. Це гарантує інтеграцію тестування протягом усього процесу розробки, дозволяючи раннє виявлення дефектів і постійне вдосконалення.

2. Візуальне представлення: V-подібна діаграма представляє зв'язок між фазами розробки та відповідними фазами тестування. Це візуальне представлення забезпечує чіткий огляд процесу розробки та допомагає виявити потенційні прогалини або збіги.

3. Акцент на документацію: V-модель приділяє значну увагу всебічній документації. Кожен етап процесу розробки має відповідну документацію, яка описує вимоги, дизайн, код і процедури тестування. Ця документація служить орієнтиром для зацікавлених сторін, розробників і команд обслуговування.

Фази V-моделі:

1. Збір і аналіз вимог: Проект починається зі збору та аналізу вимог до програмної системи. Це передбачає розуміння проблеми, яку потрібно вирішити, визначення цільової аудиторії та окреслення бажаних функцій програмного забезпечення.

2. Функціональний дизайн: На основі зібраних вимог етап функціонального проектування зосереджується на визначенні архітектури високого рівня, інтерфейсу користувача, структур даних і алгоритмів. Ця фаза забезпечує концептуальний план програмної системи.

3. Детальний проект: Етап детального проектування додатково вдосконалює функціональний дизайн, створюючи більш детальні специфікації для програмної системи. Це включає визначення архітектури низького рівня, структур даних, алгоритмів і об'єктних моделей.

4. Впровадження: Етап впровадження включає кодування програмного забезпечення відповідно до детальних специфікацій проекту. Розробники пишуть

код, ітеративно тестуючи кожен модуль, щоб переконатися в функціональності та дотриманні специфікацій дизайну.

5. Інтеграція та тестування:** Етап інтеграції та тестування передбачає інтеграцію окремих модулів, розроблених на етапі впровадження, і проведення ретельного тестування, щоб переконатися, що система функціонує належним чином. Цей етап включає модульне тестування, інтеграційне тестування, системне тестування та приймальне тестування.

6. Розгортання та обслуговування: Етап розгортання передбачає розміщення розробленого програмного забезпечення у виробництві, що робить його доступним для користувачів. Це включає в себе налаштування необхідної інфраструктури, налаштування процедур розгортання та забезпечення повної інтеграції з існуючими системами. Етап обслуговування передбачає постійну підтримку, виправлення помилок і вдосконалення програмного забезпечення.

Переваги V-моделі:

1. Структура та дисципліна: V-модель забезпечує структурований та дисциплінований підхід до розробки програмного забезпечення, сприяючи чіткості, організації та дотриманню визначених процесів.

2. Раннє виявлення дефектів: Підхід до паралельної розробки та тестування забезпечує інтеграцію тестування впродовж усього процесу розробки, дозволяючи раннє виявлення дефектів і запобігаючи тому, що їх подальше усунення стане більш дорогим.

3. Зниження ризику помилок: Акцент на документуванні та структурованих процесах зменшує ризик помилок, упущень і непорозумінь.

4. Покращене управління проектами: V-модель надає чітку дорожню карту для проекту, сприяючи кращому управлінню проектом і розподілу ресурсів.

5. Підходить для складних проектів: Структурований підхід V-моделі добре підходить для складних проектів із чітко визначеними вимогами та чіткими цілями.

Недоліки V-моделі:

1. Негнучкість до змін: Жорстка структура V-моделі може ускладнити врахування змін у вимогах або обсягах під час процесу розробки.

2. Може не адаптуватися до динамічних середовищ: V-модель може не підходити для проектів у динамічних середовищах із частими змінами або зміною вимог.

3. Може призвести до затримки тестування: Паралельний підхід до розробки та тестування може призвести до затримок у тестуванні, якщо є проблеми або переробка на етапі розробки.

4. Може не сприяти постійному вдосконаленню: Структурований характер V-моделі може не сприяти постійному вдосконаленню та адаптації до мінливих обставин.

2.1.6 Методологія DevOps

DevOps — це методологія розробки програмного забезпечення, яка долає розрив між командами розробки (Dev) і операцій (Ops). Він наголошує на співпраці, автоматизації та безперервній інтеграції та доставці (CI/CD) для оптимізації процесу доставки програмного забезпечення.

Ключові принципи DevOps:

1. Співпраця: DevOps сприяє тісній співпраці між командами розробників і операторів, щоб розірвати розбіжності та забезпечити безперебійне спілкування та обмін знаннями.

2. Автоматизація: DevOps використовує інструменти та процеси автоматизації для автоматизації повторюваних завдань, зменшення ручного втручання та підвищення ефективності.

3. Постійна інтеграція та доставка (CI/CD): DevOps використовує методи CI/CD, де код інтегрується та часто розгортається, що забезпечує швидкий зворотний зв’язок і постійне вдосконалення.

Основні практики DevOps:

1. Інфраструктура як код (IaC): Інфраструктура визначається та керується як код за допомогою таких інструментів, як Terraform або Ansible, що дозволяє автоматизувати надання і оновлення інфраструктури.

2. Контейнеризація: Програмне забезпечення упаковується в легкі контейнери за допомогою таких інструментів, як Docker, що сприяє переносимості та узгодженим середовищам у розробці, тестуванні та виробництві.

3. Моніторинг і спостережуваність: системи оснащені інструментами моніторингу для збору й аналізу показників, журналів і трасування, що дозволяє в режимі реального часу бачити стан і продуктивність системи.

4. DevSecOps: Безпека інтегрована в конвеєр DevOps, гарантуючи, що методи безпеки впроваджуються протягом усього процесу розробки та розгортання.

Переваги DevOps:

1. Швидший час виходу на ринок: DevOps прискорює цикли доставки програмного забезпечення, забезпечуючи швидші випуски та кращу реакцію на вимоги ринку.

2. Зменшення кількості помилок та інцидентів: Автоматизація, часте тестування та безперервний моніторинг допомагають завчасно виявляти та виправляти проблеми, зменшуючи ризик збоїв у виробництві та зводячи до мінімуму простої.

3. Покращена якість і надійність: Безперервне тестування та цикли зворотного зв’язку сприяють вищій якості та надійності програмного забезпечення, що призводить до задоволених користувачів і зниження витрат на підтримку клієнтів.

4. Зниження витрат: DevOps може призвести до економії коштів завдяки підвищенню ефективності, скороченню часу простою та зниженню витрат на обслуговування.

Проблеми DevOps:

1. Культурний зсув і співпраця: Впровадження DevOps вимагає культурних зрушень всередині організації, сприяння співпраці та усунення розривів між командами розробників і операторів.

2. Розвиток навичок: Впровадження DevOps вимагає спеціальних навичок у автоматизації, хмарних технологіях і методах безперервної доставки.

3. Інструменти та інтеграція: Інтеграція різних інструментів DevOps і забезпечення бездоганної інтеграції на різних платформах може бути складним і трудомістким.

4. Постійне вдосконалення: DevOps — це безперервний шлях, який вимагає постійної адаптації та вдосконалення, щоб йти в ногу з розвитком технологій і вимогами ринку.

Підсумовуючи, DevOps — це трансформаційна методологія, яка усуває розбіжності між командами розробників і операторів, спрощуючи процес доставки програмного забезпечення та прискорюючи інновації. Завдяки автоматизації, безперервній інтеграції та доставці, а також інфраструктурі як коду, організації DevOps можуть надавати високоякісне програмне забезпечення швидше, надійніше та за нижчою ціною.

2.1.7 Методологія швидкої розробки (RAD)

Швидка розробка програм (RAD) — це методологія розробки програмного забезпечення, яка наголошує на швидкості та гнучкості. Він розбиває проект на менші ітераційні цикли, використовуючи прототипи та відгуки користувачів для швидкої доставки робочого програмного забезпечення.

Ключові принципи RAD:

1. Зосередження на залученні користувачів: RAD приділяє значну увагу залученню користувачів протягом усього процесу розробки. Користувачі беруть участь у зборі вимог, створенні прототипів і тестуванні, гарантуючи, що програмне забезпечення відповідає їхнім потребам і очікуванням.

2. Ітераційна та поетапна розробка: RAD дотримується ітераційного підходу, розбиваючи проект на менші цикли або фази. Кожен цикл зосереджується на створенні робочого прототипу певного набору функцій, що забезпечує швидкий зворотний зв’язок і адаптацію до мінливих вимог.

3. Акцент на створення прототипів: RAD використовує методи створення прототипів для створення візуальних представлень програмного забезпечення, що

дозволяє користувачам візуалізувати систему та взаємодіяти з нею до її повної розробки.

4. Дизайн, орієнтований на користувача: RAD використовує підхід, орієнтований на користувача, зосереджуючись на розумінні потреб користувачів і розробці програмного забезпечення для ефективного задоволення цих потреб.

Основні практики RAD:

1. JAD (Спільна розробка додатків): Сеанси JAD об'єднують зацікавлених сторін, включаючи користувачів, розробників і бізнес-аналітиків, для спільного збору й уточнення вимог.

2. Розробка прототипу: Команди RAD створюють прототипи користувацького інтерфейсу програмного забезпечення та функціональних можливостей для збору відгуків користувачів і вдосконалення дизайну.

3. Поступова поставка: Робоче програмне забезпечення постачається поетапно, що дозволяє користувачам отримувати ранні відгуки та постійне вдосконалення.

4. Тестування та відгуки: Ретельне тестування та цикли зворотного зв'язку інтегровані протягом усього процесу розробки, щоб забезпечити відповідність програмного забезпечення стандартам якості та вимогам користувачів.

Переваги RAD:

1. Прискорена розробка: ітераційний підхід RAD забезпечує швидшу доставку робочого програмного забезпечення, дозволяючи організаціям швидко реагувати на зміни ринку та конкурентний тиск.

2. Зменшений ризик: Залучення користувачів на ранній стадії та створення прототипів допомагають завчасно виявляти та вирішувати потенційні проблеми, зменшуючи ризик дорогої переробки пізніше в процесі розробки.

3. Підвищена задоволеність користувачів: Орієнтація RAD на залучення користувачів і дизайн, орієнтований на користувача, призводить до програмного забезпечення, яке краще відповідає потребам і очікуванням користувачів, що призводить до більшої задоволеності користувачів.

4. Гнучкість для адаптації: Ітеративний підхід RAD забезпечує гнучкість у пристосуванні до змін у вимогах або зміні ринкових умов.

Виклики RAD:

1. Чіткість високих вимог: RAD вимагає чітких і чітко визначених вимог із самого початку, що може вимагати додаткових зусиль.

2. Надійне управління проектами: Ефективне управління проектами має вирішальне значення для забезпечення добре структурованого та ефективного процесу RAD, управління розподілом ресурсів і запобігання затримкам.

3. Технічний досвід: Команди RAD повинні володіти сильними технічними навичками в інструментах прототипування, дизайні інтерфейсу користувача та методології швидкого прототипування.

4. Підходить для конкретних проектів: RAD може не підходити для всіх проектів, особливо тих, що мають дуже складні або жорсткі вимоги.

Підсумовуючи, RAD є потужною методологією для швидкої та ефективної доставки програмного забезпечення, особливо для проектів з орієнтованими на користувача вимогами та потребою в гнучкості. Однак його успіх залежить від чітких вимог, ефективного управління проектами та команди з необхідним технічним досвідом. Ретельний розгляд характеристик проекту та динаміки команди має важливе значення, коли вирішуєте, чи підходить RAD.

2.2 Вибір правильної методології

Вибір методології залежить від різних факторів, зокрема:

- Розмір та складність проекту
- Стабільність і мінливість вимог
- Командна динаміка та досвід
- Терміни та бюджетні обмеження

Загалом Waterfall підходить для чітко визначених, стабільних проектів із чітким масштабом. Гнучкі методології чудово працюють у динамічному середовищі, де вимоги змінюються, а безперервний зворотний зв’язок має важливе

значення. DevOps ефективний в організаціях, які прагнуть подолати розрив між розробкою та операціями, спрощуючи доставку програмного забезпечення. RAD ідеально підходить для проектів зі стислими термінами та зосередженістю на взаємодії з користувачем.

Методології розробки програмного забезпечення забезпечують основу для планування, організації та ефективного виконання проектів. Розуміючи сильні сторони та обмеження кожного підходу, команди можуть вибрати методологію, яка найкраще відповідає цілям, вимогам і динаміці проекту. Зокрема в цій роботі буде використовуватися методологія Waterfall, оскільки вона написаний однією людиною, мала чіткі строки та масштаби ще на початку планування.

2.3 Вибір алгоритмів та стратегії впровадження

1. Алгоритм Дейкстри:

Якщо програма в основному зосереджена на оптимізації пішохідних маршрутів у містах, алгоритм Дейкстри є відповідним вибором завдяки його простоті та ефективній обробці незважених графіків. Він ефективно знаходить найкоротший шлях між одним вихідним вузлом і всіма іншими вузлами на графі, що робить його ідеальним для пошуку найпряміших пішохідних маршрутів.

Однак алгоритм Дейкстри не підходить для оптимізації автомобільних чи залізничних маршрутів, оскільки він не може врахувати різні автомобільні та залізничні мережі та відповідний час у дорозі та відстані. Крім того, він не може впоратися з від’ємними вагами країв, які можуть виникнути при розгляді таких факторів, як затори на дорогах або закриті дороги.

2. A* Алгоритм:

Для оптимізації автомобільних і залізничних маршрутів між містами та всередині них алгоритм A* є більш потужним і універсальним вибором. Це варіант алгоритму Дейкстри, який використовує евристичну функцію для керування пошуком, що робить його ефективнішим для великих графів і здатним обробляти

зважені ребра. Це дає змогу враховувати такі фактори, як час у дорозі, відстань і умови руху під час визначення оптимального маршруту.

Алгоритм A* також ефективно обробляє від'ємні ваги країв, що робить його придатним для включення даних про дорожній рух у реальному часі або врахування таких факторів, як перекриття доріг або об'їзди. Крім того, його можна змінити, щоб включити додаткові обмеження, такі як уникнення певних доріг або максимізація мальовничої краси маршруту.

3. Оптимізація колонії ANT (ACO):

Алгоритм ANT Colony Optimization (ACO) може бути цінним інструментом для оптимізації маршрутів у складних і динамічних середовищах, таких як міські райони з різними схемами трафіку. Він імітує поведінку мурах у пошуках їжі для пошуку ефективних шляхів, роблячи його стійким до мінливих умов і здатним визначати альтернативні маршрути, коли виникають перешкоди чи збої.

Алгоритм ACO може бути особливо корисним для оптимізації маршрутів, які включають як пішохідну, так і автомобільну подорож, оскільки він може збалансувати найкоротший шлях із такими факторами, як зручні для пішоходів маршрути та умови руху. Його також можна налаштувати з урахуванням конкретних уподобань, наприклад мінімізації впливу шуму або уникнення зайнятих місць.

4. Гібридні алгоритми:

У випадках, коли програма охоплює широкий діапазон режимів руху та обмежень, поєднання кількох алгоритмів може забезпечити більш комплексне та гнучке рішення. Наприклад, початкова оптимізація з використанням алгоритму Дейкстри для пішохідних маршрутів може супроводжуватися пошуком A* для автомобільних маршрутів, включаючи дані про дорожній рух і дорожні мережі.

Крім того, ACO можна включити для обробки динамічних змін у режимі реального часу, змінюючи маршрути пішоходів і транспортних засобів на основі оновлених умов дорожнього руху або несподіваних подій. Цей гібридний підхід використовує сильні сторони кожного алгоритму для забезпечення більш надійного та адаптивного рішення.

Вибираючи відповідний алгоритм для свого проекту, було враховано наступні фактори:

- Режими пересування: Визначено основні способи пересування, які оптимізує ваша програма, наприклад пішохідний, автомобільний або потяговий, щоб вибрати алгоритми, які відповідають їхнім характеристикам.
- Географічне охоплення: Визначено географічне охоплення вашої програми, чи вона зосереджена на внутрішньоміських поїздках, міжміських маршрутах або комбінації обох. Це вплине на складність представлення графа та придатність алгоритмів.
- Оновлення в режимі реального часу: Оцінено, чи потребує ваша програма оновлення в режимі реального часу з урахуванням умов дорожнього руху, закритих доріг або інших динамічних факторів. Якщо так, то такі алгоритми, як ACO або гібридні підходи, можуть бути кращими.
- Бажана точність: Визначено рівень точності, необхідний для вашої програми. Для критичних програм можуть знадобитися точні алгоритми, такі як алгоритм Дейкстри. Для менш критичних програм наближені алгоритми можуть забезпечити компроміс між точністю та ефективністю.
- Обмеження та переваги: Визначено будь-які конкретні обмеження чи переваги, які потрібно враховувати, як-от уникання певних доріг, мінімізація впливу шуму або максимізація мальовничої краси. Вибрані алгоритми повинні ефективно включати ці фактори.

Оскільки програма в основному зосереджена на оптимізації маршрутів поїздів та іншого трансопрту як у містах, так і між ними, алгоритм *A* з оновленнями в реальному часі є найбільш прийнятним вибором. Він поєднує в собі ефективність A* для пошуку найкоротшого шляху з можливістю включати дані про дорожній рух у реальному часі та збої.

Алгоритм A*, як обговорювалося раніше, є універсальним і ефективним алгоритмом для пошуку найкоротшого шляху у зважених графіках. Він може обробляти такі фактори, як час у дорозі, відстань і умови руху, що робить його ідеальним для оптимізації маршрутів поїздів.

Вимога щодо оновлень у реальному часі вимагає використання алгоритму, який може динамічно адаптуватися до мінливих умов. Алгоритм A* можна адаптувати для включення даних у реальному часі, таких як розклад руху поїздів, зміни платформи та затримки.

Підхід

1. Збирається дані в реальному часі: Встановлюється підключення до каналу даних у реальному часі, який надає актуальну інформацію про розклад руху поїздів, зміни платформи та затримки.

2. Інтегрується дані в алгоритм A*: Запроваджується механізм для інтеграції даних у реальному часі в обчислення алгоритму A*. Це може включати оновлення графічного представлення або зміну евристичної функції.

3. Постійне оновлення: Постійно оновлюється алгоритм A* останніми даними в реальному часі, щоб гарантувати, що він надає найточніші та найновіші рекомендації щодо подорожей.

Цей підхід дозволить програмі надавати користувачам оптимальні маршрути поїздів, які відображають зміни та збої в реальному часі, забезпечуючи безперебійну та ефективну подорож.

2.4 Тестування на користування та оцінка

Тестування на користування та оцінка – важливий етап у розробці програмного забезпечення, спрямований на забезпечення задоволення користувачів та відповідність продукту їхнім потребам і очікуванням. Ця тема охоплює широкий спектр методів, підходів та інструментів, що використовуються для перевірки та оцінки функціональності, ефективності та якості програмного забезпечення з точки зору кінцевого користувача.

Сучасне програмне забезпечення стикається з різноманітними вимогами і викликами, і тестування на користування є важливим інструментом для забезпечення його успіху на ринку. При цьому важливо не тільки переконатися, що програма виконує свої функції правильно, але і зробити це так, щоб вона була

зручною, ефективною та задовольняла потреби та очікування кінцевого користувача.

Тестування на користування охоплює різні аспекти, такі як тестування інтерфейсу користувача, перевірка взаємодії з програмою, тестування продуктивності та навантаження, а також забезпечення безпеки та надійності програмного продукту.

У цьому розділі досліджується тема тестування на користування та оцінок, розглянемо різні аспекти цього процесу, включаючи стратегії та методи тестування, вибір інструментів, проведення користувацьких тестів та збір фідбеку від користувачів. Також буде розглянуто питання оцінки якості та ефективності програмного забезпечення, використовуючи різні метрики та підходи.

Мета цього розділу - надати глибше розуміння процесу тестування на користування та оцінки, а також допомогти зробити програмні продукти більш зручними, ефективними та задовольняти потреби користувачів. Тестування на користування - це важлива складова частина процесу розробки програмного забезпечення, яка вимагає уважної уваги та аналізу.

РОЗДІЛ 3. АНАЛІЗ ВИМОГ ТА РОЗРОБКА ДОДАТКУ

Розділ Аналіз вимог та розробка охоплює всебічний процес від ідентифікації та аналізу вимог до втілення цих вимог у функціональну розробку продукту. Визначення як функціональних, так і нефункціональних вимог є фундаментальним кроком, який встановлює основу для подальшого проектування та розробки. Важливість детального розуміння потреб користувачів та бізнес-цілей є ключовою для створення високоефективного та відповідного продукту. Розділ також звертає увагу на важливість правильного вибору технологій, архітектури системи та стратегій імплементації, що забезпечують необхідну гнучкість, масштабованість, та безпеку веб-додатку. Велика увага приділяється інтерфейсу користувача, його інтуїтивності та ергономіці, а також аспектам тестування продукту для забезпечення його надійності та якості.

3.1 Специфікація функціональних вимог

Для проєкту, пов'язаного з розробкою програмного забезпечення для оптимізації маршрутів подорожей та навігації, рекомендовано розробити докладну і структуровану Специфікацію функціональних вимог (SRS - Software Requirements Specification). SRS є ключовим документом, який деталізує всі функції, функціональність та вимоги до програмного продукту.

Проєкт "Розробка web застосунку для оптимізації маршрутів подорожей та навігації" спрямований на створення високоефективного та зручного у використанні інструменту, який допоможе користувачам планувати та оптимізувати свої подорожі, забезпечуючи найкращий шлях до пункту призначення та забезпечуючи навігаційну підтримку під час подорожі.

Цей проєкт має наступні основні цілі:

1. Забезпечити користувачам зручну та ефективну можливість планування маршрутів: Розробити програмний продукт, який дозволить користувачам вводити

початкові та кінцеві точки подорожі та отримувати оптимальні маршрути на основі різних параметрів, таких як відстань, час подорожі, вид транспорту та інші.

2. Забезпечити можливість оптимізації маршрутів у реальному часі: Програмний продукт має мати можливість моніторити умови дорожнього руху та інші змінні фактори та надавати альтернативні маршрути у реальному часі для уникнення заторів та збереження часу користувачів.

3. Забезпечити зручний та інтуїтивно зрозумілий інтерфейс: Програмний продукт має бути легким у використанні та надавати інтуїтивно зрозумілий інтерфейс для користувачів різного рівня експертизи.

4. Забезпечити підтримку різних типів транспорту: Програмний продукт повинен підтримувати різні види транспорту, включаючи автомобільний, громадський, велосипедний та пішохідний, і надавати відповідні маршрути для кожного виду.

Обсяг проєкту включає в себе наступні ключові компоненти:

1. Розробка основного програмного забезпечення: Створення програмного продукту, який забезпечить функціональність планування маршрутів та навігації.

2. Реалізація системи моніторингу умов дорожнього руху: Розробка системи, яка буде відстежувати стан доріг, дорожні затори та інші фактори, що можуть впливати на маршрутизацію.

3. Розробка інтерфейсу користувача: Створення зручного та інтуїтивно зрозумілого інтерфейсу, який дозволить користувачам взаємодіяти з програмним продуктом.

5. Реалізація можливості оновлення в реальному часі: Додавання можливості оновлення маршрутів та навігаційних інструкцій у реальному часі на основі змінних умов.

6. Тестування та якість: Проведення різних видів тестів для забезпечення якості програмного продукту та відповідності вимогам.

7. Документація та підтримка: Підготовка документації для користувачів та надання технічної підтримки після випуску продукту.

8. Інтеграція з зовнішніми сервісами та джерелами даних: Інтеграція з GPS, картографічними сервісами та іншими зовнішніми джерелами даних.

9. Забезпечення безпеки та конфіденційності даних користувачів: Захист особистих даних користувачів та забезпечення безпеки використання програми.

10. Оцінка та вдосконалення продукту: Збір відгуків користувачів, аналіз даних використання та внесення змін для покращення продукту.

Цей опис загальної мети та обсягу проєкту надає загальне уявлення про те, що передбачається у процесі розробки програмного продукту для оптимізації маршрутів подорожей та навігації. Конкретні деталі та вимоги будуть визначені у специфікації функціональних вимог та інших відповідних документах проєкту.

Завершуючи розробку специфікації функціональних вимог, важливо зробити її доступною для всіх членів команди проєкту та зацікавлених сторін. Цей документ стане основою для розробки, тестування та оцінки вашого програмного продукту, а також інструментом для уникнення недорозумінь і невідповідностей у процесі розробки.

3.2 Специфікація нефункціональних вимог проєкту

Специфікація нефункціональних вимог визначає параметри, які характеризують якість, продуктивність та інші нефункціональні аспекти програмного продукту "Розробка програмного забезпечення для оптимізації маршрутів подорожей та навігації". Ці параметри важливі для забезпечення задоволення користувачів та ефективної роботи продукту. Нижче подано докладні специфікації нефункціональних вимог:

1. Продуктивність:

- Час відгуку: Середній час відгуку програми на запити користувачів не повинен перевищувати 2-3 секунди.

- Ефективність використання ресурсів: Програма повинна ефективно використовувати ресурси (процесор, пам'ять), і завантаження системи повинно бути прийнятним для різних пристроїв.

2.Надійність:

- Стійкість до помилок: Програмний продукт повинен бути стійким до помилок та незапланованих аварій, а також може відновлювати свою роботу після збоїв.

- Резервне копіювання даних: Система повинна мати механізм резервного копіювання даних користувачів для уникнення втрати інформації в разі аварій.

3. Безпека:

- Захист особистих даних: Програма повинна гарантувати конфіденційність і безпеку особистих даних користувачів.

- Захист від вразливостей: Програмний продукт має бути захищений від різних видів атак, таких як SQL-ін'єкції, переповнення буфера і т. д.

- Аутентифікація та авторизація: Забезпечити надійну систему аутентифікації та авторизації користувачів.

4. Сумісність:

- Сумісність з різними платформами: Програмний продукт має бути сумісним з різними операційними системами, такими як Windows, macOS, Android та iOS.

- Сумісність з різними браузерами: Якщо передбачається веб-версія, то вона має бути сумісною з різними браузерами, такими як Chrome, Firefox, Safari, Edge тощо.

5. Інтерфейс користувача:

- Інтуїтивно зрозумілий інтерфейс: Інтерфейс повинен бути інтуїтивно зрозумілим для користувачів різного рівня експертизи.

- Доступність: Забезпечити доступність для людей з обмеженими можливостями.

6. Масштабованість:

- Можливість масштабування: Програмний продукт повинен бути готовим до масштабування для підтримки росту користувачів та обсягів даних.

7. Документація та навчання:

- Документація для користувачів: Забезпечення наявності документації для користувачів щодо використання продукту.

- Навчання та підтримка: Надання навчальних матеріалів та технічної підтримки користувачам.

8. Вимоги до документації та тестування:

- Документація розробників: Забезпечити наявність документації для розробників, включаючи коментарі до коду та технічні описи.

- Тестова документація: Підготовка тестової документації та забезпечення проведення всебічних тестів.

Ця Специфікація нефункціональних вимог дозволяє визначити і виміряти аспекти продукту, які не відносяться безпосередньо до його функціональності, але важливі для його успішного впровадження та використання. Відповідність цим нефункціональним вимогам допоможе забезпечити високу якість та задоволення користувачів.

3.3 Аналіз вимог для профіля користувача та використання

Для розробки програмного продукту, спрямованого на оптимізацію маршрутів подорожей та навігацію, важливо провести аналіз вимог щодо профілю користувача та способів використання продукту. Цей аналіз допоможе зрозуміти потреби користувачів і забезпечити їхню задоволеність та зручність використання програмного продукту.

1. Профіль користувача:

- Індивідуальні користувачі: Основна аудиторія продукту включає в себе індивідуальних користувачів, які шукають оптимальні маршрути для

особистих подорожей на різних видів транспорту, таких як автомобіль, велосипед, громадський транспорт або пішки.

- Бізнес-користувачі: Бізнес-користувачі можуть використовувати програмний продукт для планування подорожей до зустрічей або доставки товарів. Вони можуть вимагати додаткових функціональних можливостей, таких як оптимізація маршрутів для бізнес-потреб та статистика витрат.

- Туристи та мандрівники: Ця група користувачів може використовувати програму для планування туристичних подорожей та відвідування цікавих місць. Вони можуть вимагати інформації про туристичні об'єкти та альтернативні маршрути.

- Водії громадського транспорту: Користувачі громадського транспорту можуть шукати оптимальні маршрути та розклади для їхніх поїздок.

- Велосипедисти та пішоходи: Для цих користувачів важливо враховувати специфічні особливості маршрутів для велосипедистів та пішоходів, такі як велосипедні шляхи та тротуари.

2. Способи використання

- Планування подорожей: Користувачі будуть вводити початкові та кінцеві точки своєї подорожі та отримувати оптимальні маршрути в залежності від їхніх вимог і умов.

- Навігація під час подорожі: Під час подорожі користувачі отримуватимуть навігаційні вказівки та інформацію про повороти та відстань до пункту призначення.

- Моніторинг умов дорожнього руху: Програмний продукт може надавати користувачам інформацію про дорожні затори, аварії та інші фактори, що можуть впливати на маршрут.

- Збереження обраного маршруту: Користувачі можуть зберігати обрані маршрути для майбутнього використання та планування.

- Історія подорожей*: Система може зберігати історію подорожей користувачів для аналізу та підрахунку витрат.

Аналіз вимог для профілю користувача та способів використання дозволяє належним чином спроектувати функціональність програмного продукту та забезпечити задоволення потреб і очікувань різних категорій користувачів.

3.4 Архітектура системи та огляд компонентів

Керована подіями архітектура – це парадигма розробки програмного забезпечення, у якій компоненти спілкуються через події. Події – це сповіщення, які зазвичай публікує виробник і використовують один або кілька підписників. Цей підхід сприяє слабкому зв’язку між компонентами, дозволяючи їм розробляти, розгортати та масштабувати незалежно один від одного.

Ключові характеристики керованої подіями архітектури:

1. Слабкий зв’язок: Компоненти спілкуються через події, а не через прямий виклик методу. Це відокремлює компоненти, дозволяючи їх розробляти, розгортати та масштабувати незалежно.

2. Асинхронний зв’язок: Події публікуються асинхронно, що дозволяє компонентам реагувати на події, не чекаючи, поки видавець завершить свою роботу. Це покращує швидкість реакції та дозволяє керованим подіями архітектурам обробляти великі обсяги даних.

3. Потоки подій: Події можна організовувати в потоки, що дозволяє компонентам підписуватися на певні події або групи подій. Це спрощує обробку подій і забезпечує більш складну обробку даних подій.

4. Посередники подій: Посередники подій відіграють центральну роль в керованих подіями архітектурах, діючи як посередники між виробниками та передплатниками. Вони керують маршрутизацією повідомлень, надійністю та збереженням повідомлень.

Переваги архітектури, керованої подіями:

1. Масштабованість: Архітектури, керовані подіями, дуже масштабовані, оскільки компоненти можна масштабувати незалежно, а події можна розподіляти між кількома серверами.

2. Відмовостійкість: Архітектури, керовані подіями, можуть більш витончено переносити збої, оскільки компоненти можуть виходити з ладу, не впливаючи на систему в цілому.

3. Гнучкість: Архітектури, керовані подіями, сприяють гнучкості, оскільки компоненти можна легко додавати або видаляти, не порушуючи роботу всієї системи.

4. Швидкість реагування: Архітектури, керовані подіями, можуть реагувати на події в реальному часі, що робить їх добре підходящими для додатків, які потребують швидкої та своєчасної відповіді.

Приклади використання архітектури, керованої подіями:

1. Обробка даних у реальному часі: Отримання, обробка та аналіз даних у реальному часі з датчиків, пристроїв Інтернету речей, соціальних мереж та інших джерел.

2. Розподілені програми: Відокремлюється та розподіляйте компоненти великомасштабних програм для покращення масштабованості та відмовостійкості.

3. Автоматизація робочого процесу: Керується складними робочими процесами, запускаючи події та надсилаючи повідомлення між різними компонентами.

4. Передбачувана аналітика: Моніторинг потоків даних і ініціювання сповіщень або дій на основі моделей і подій у реальному часі.

5. Інтелектуальні програми: Розроблюється інтелектуальні програми, які реагують на події та приймають рішення на основі даних у реальному часі.

Архітектура, керована подіями, — це потужний і універсальний підхід до проектування програмного забезпечення, який можна застосовувати до широкого кола програм. Його здатність обробляти дані в реальному часі, відокремлювати компоненти та ефективно масштабувати робить його цінним інструментом для створення сучасних та адаптивних додатків.

3.4.1 Компоненти

Будуть використані наступні компоненти системи:

- Виробник події: Виробник події відповідає за створення подій, які представляють сповіщення або повідомлення, що описують певну подію. У контексті оптимізації маршруту виробником подій може бути компонент інтерфейсу користувача, який отримує дані користувача, алгоритм маршрутизації, який генерує оптимізовані маршрути, або джерело даних трафіку, яке надає інформацію про трафік у реальному часі.

- Посередник подій: Посередник подій діє як посередник між організаторами подій і підписниками. Він отримує події від виробників, зберігає їх у довготривалій черзі повідомлень і доставляє передплатникам. У контексті оптимізації маршруту посередником подій може бути система обміну повідомленнями, наприклад Apache Kafka або RabbitMQ.

- Підписник подій: Підписник подій — це компонент, який отримує й обробляє події. У контексті оптимізації маршруту підписником події може бути служба оптимізації маршруту, яка обчислює найкоротший шлях між двома точками, служба обробки даних про трафік, яка оновлює прогнози маршруту, або компонент інтерфейсу користувача, який відображає оптимізовані маршрути для користувача.

- Обробники подій: Обробники подій – це компоненти, які запускаються подіями. Вони інкапсулюють логіку обробки та реагування на події. У контексті оптимізації маршруту обробники подій можуть бути функціями, які витягують відповідну інформацію з подій, оновлюють структури даних або запускають інші події.

- Сховища даних: Сховища даних використовуються для зберігання постійних даних і керування ними. У контексті оптимізації маршрутів сховищами даних можуть бути бази даних, файлові системи або хмарні служби зберігання.

- Сервери додатків: Сервери додатків відповідають за виконання логіки додатків і керування ресурсами. У контексті оптимізації маршрутів сервери

додатків можуть бути веб-серверами, платформами мікросервісів або безсерверними платформами.

- Мережева інфраструктура: Мережева інфраструктура використовується для з’єднання компонентів архітектури на основі подій. Це може включати брандмауери, маршрутизатори та балансувальники навантаження.

- Компонент візуалізації: Цей компонент можна використовувати для відображення користувачеві оптимізованих маршрутів і інформації про дорожній рух.

3.5 Дизайн інтерфейсу користувача та створення прототипів

Для розробки ефективного та зручного інтерфейсу користувача (UI) для програмного продукту "Розробка програмного забезпечення для оптимізації маршрутів подорожей та навігації", необхідно враховувати потреби та очікування різних категорій користувачів, а також спростити процес планування маршрутів та навігації. Нижче наведено опис основних елементів дизайну інтерфейсу та створення прототипів.

1. Головний екран:

- Головний екран має містити поля для введення початкової та кінцевої точки подорожі.

- Кнопка "Пошук маршруту" дозволяє користувачеві знайти оптимальний маршрут.

2. Меню навігації:

- Верхнє меню містить розділи для доступу до різних функцій, таких як "Мої маршрути", "Карта", "Пошук цікавих місць", "Налаштування" та інші.

- Меню також може включати значки для швидкого доступу до функцій.

3. Карта:

- Візуалізація маршруту на карті, яка показує дороги, повороти та іншу інформацію про маршрут.

- Можливість масштабування карти та переміщення для перегляду деталей маршруту.

4. Результати пошуку:

- Перелік результатів пошуку, які включають в себе різні варіанти маршрутів.

- Детальна інформація про кожен маршрут, включаючи відстань, час подорожі, вид транспорту та інші параметри.

5. Навігація під час подорожі:

- Вказівки щодо поворотів та інформація про відстань до наступного повороту.

- Повідомлення про дорожні затори та інші умови на дорозі.

6. Додаткові функції:

- Функції для пошуку цікавих місць, такі як ресторани, готелі, пам'ятки архітектури та інші.

- Можливість зберігання обраних маршрутів та перегляду історії подорожей.

Створення прототипів:

Для розробки прототипу інтерфейсу можна використовувати спеціалізовані програми, такі як Figma, Adobe XD або Sketch. Прототип дозволяє візуалізувати функціональність та інтерфейс програми до фактичного програмування. Прототип може бути використаний для збору відгуків від користувачів та вдосконалення дизайну.

Загальний дизайн інтерфейсу має бути інтуїтивно зрозумілим і зручним для користувачів, незалежно від їхнього досвіду використання схожих програмних продуктів. Прототипи дозволяють виявити потенційні проблеми та внести необхідні зміни перед розробкою фінального інтерфейсу.

3.6 Розробка та інтеграція алгоритмів

У цьому ключовому розділі дипломної роботи вивчається процес розробки та інтеграції алгоритмів, що визначають ефективне планування оптимальних маршрутів для подорожей. На даному етапі дослідження відбувається перехід від

теоретичних засад до практичної реалізації інструментів, що забезпечують найкращі результати у реальних умовах.

Засновок цього розділу полягає у ретельній розробці та інтеграції компонентів, що є структурними елементами алгоритмічного планування маршрутів. Основною фігурою цього розділу є клас `Node`, який визначає вузли у просторі пошуку. Кожен вузол представляє собою окрему точку на карті і має властивості, такі як координати "x" і "y", посилання на "батьківський" вузол, вартість "g" та значення "h", що визначаються під час роботи алгоритму A*. Метод `__lt__` для класу `Node` дозволяє проводити порівняння вузлів на основі їхньої суми `g` і `h`, що є ключовим для визначення оптимального маршруту.

Ключовою функцією, яка дозволяє реалізувати алгоритм A*, є функція `astar`. Вона отримує на вхід граф, вузол-початок і вузол-ціль, і відповідає за пошук найкращого маршруту в графі. Функція `astar` використовує два набори вузлів: "відкриті" та "закриті", щоб систематично досліджувати та оцінювати вузли в процесі пошуку маршруту. Вона постійно витягує вузли з набору "відкриті" та оновлює їхні значення `g` та `h` для сусідніх вузлів, щоб визначити найкращий маршрут до цілі.

Важливою складовою є також евристична функція, яка оцінює відстань від поточного вузла до цілевого вузла. У нашому випадку використовується манхеттенська відстань, яка допомагає алгоритму A* вибирати оптимальний шлях до цілі, враховуючи обмеження на переміщення в горизонтальному та вертикальному напрямках.

Крім того, в цьому розділі докладно аналізується клас `Graph`, який представляє собою граф з взаємозв'язками між вузлами. Клас `Graph` надає методи для додавання ребер, отримання сусідів, отримання ваги ребра та встановлення значення `g` для вузлів. Цей клас є ключовим для створення та управління графом, який використовується алгоритмом A* для знаходження оптимального маршруту.

Загальна мета цього розділу - продемонструвати як розроблені алгоритми і компоненти взаємодіють між собою для досягнення ефективного та оптимального планування маршрутів для подорожей.

3.6.1 Функція astar

Ця функція реалізує алгоритм A* для пошуку оптимального маршруту в графі. Її реалізацію на мові Python можна побачити на рисунку 3.1.

```python
def a_star(start, goal, grid):
    close_set = set()
    came_from = {}
    gscore = {start: 0}
    fscore = {start: heuristic(start, goal)}
    open_set = []

    heapq.heappush(open_set, (fscore[start], start))

    while open_set:
        current = heapq.heappop(open_set)[1]

        if current == goal:
            data = []
            while current in came_from:
                data.append(current)
                current = came_from[current]
            return data[::-1]

        close_set.add(current)
        for neighbor in get_neighbors(current, grid):
            if neighbor in close_set:
                continue
            tentative_g_score = gscore[current] + heuristic(current, neighbor)

            if neighbor not in [i[1] for i in open_set] or tentative_g_score < gscore[neighbor]:
                came_from[neighbor] = current
                gscore[neighbor] = tentative_g_score
                fscore[neighbor] = tentative_g_score + heuristic(neighbor, goal)
                heapq.heappush(open_set, (fscore[neighbor], neighbor))

    return False
```

Рис. 3.1 Реалізація алгоритму A* на мові Python

Опис роботи цієї функції:

1.Початкова налаштування: Функція приймає на вхід початковий вузол `start`, цільовий вузол `goal` і двовимірний граф `grid`. Також вона ініціалізує ряд важливих структур даних, таких як `close_set` (набір відвіданих вузлів), `came_from` (словник, що відстежує, звідки прийшли до кожного вузла), `gscore` (словник, що зберігає вартість шляху від початкового вузла до кожного вузла) і `fscore` (словник, що зберігає значення f-функції для кожного вузла, де f = g + h).

2. Додавання початкового вузла в `open_set`: Початковий вузол `start` додається до пріоритетного черги `open_set` з вагою `fscore[start]`, де `fscore[start]` обчислюється за допомогою евристичної функції `heuristic`.

3. Основний цикл пошуку: Поки черга `open_set` не порожня, алгоритм продовжує шукати оптимальний маршрут. На кожній ітерації вибирається вузол `current` з найменшим значенням `fscore` із черги `open_set`.

4. Перевірка, чи досягнуто цільового вузла: Якщо поточний вузол current дорівнює цільовому вузлу goal, алгоритм знаходить оптимальний маршрут, будуючи його зворотньо від цільового вузла до початкового, використовуючи структуру came_from. Маршрут зберігається у зворотньому порядку і повертається як результат.

5. Оновлення close_set і обчислення сусідів: Поточний вузол current` додається до close_set, щоб позначити, що він був відвіданий. Далі обчислюються сусіди поточного вузла current за допомогою функції get_neighbors: Для кожного сусіда виконується наступне:

- Перевіряється, чи сусід уже входить до close_set. Якщо так, то алгоритм переходить до наступного сусіда.

- Обчислюється тимчасова вартість tentative_g_score для поточного сусіда на підставі вартості шляху до поточного вузла current і евристичного значення heuristic.

- Перевіряється, чи сусід не входить до черги open_set або tentative_g_score менше, ніж gscore для сусіда. Якщо це так, то оновлюється інформація про сусіда, включаючи came_from, gscore і fscore, і сусід додається до черги open_set з оновленим пріоритетом.

6. Повернення результату: Якщо черга open_set опустіла, і цільового вузла не було досягнуто, функція повертає значення False, що вказує на те, що маршрут до цілі не знайдено.

У підсумку, функція astar використовує алгоритм A* для знаходження оптимального маршруту в графі з початкового вузла start до цільового вузла goal. Вона ефективно використовує структури даних та евристичні функції для

здійснення цього пошуку і повертає знайдений маршрут у вигляді послідовності вузлів.

3.6.2 Евристична функція

Реалізація евристики на мові Python написана на рисунку 3.2.

```
def haversine(point1, point2, unit=Unit.KILOMETERS, normalize=False, check=True):
    # unpack latitude/longitude
    lat1, lng1 = point1
    lat2, lng2 = point2

    # normalize points or ensure they are proper lat/lon, i.e., in [-90, 90] and [-180, 180]
    if normalize:
        lat1, lng1 = _normalize(lat1, lng1)
        lat2, lng2 = _normalize(lat2, lng2)
    elif check:
        _ensure_lat_lon(lat1, lng1)
        _ensure_lat_lon(lat2, lng2)

    return get_avg_earth_radius(unit) * _haversine_kernel(lat1, lng1, lat2, lng2)
```

Рис.3.2 Реалізація евристики на мові Python

Дані, що функція приймає:

- point` та point2 є кортежами, що представляють широту та довготу двох точок відповідно.

- unit визначає одиницю вимірювання для відстані (наприклад, кілометри, милі). За замовчуванням використовуються кілометри.

- normalize - це прапорець для нормалізації значень широти та довготи.

- check - прапорець для перевірки значень широти та довготи.

Функція починається з розпакування кортежів point1 і point2 в lat1, lng1, `lat2` і lng2.

Якщо normalize є True, вона нормалізує точки, щоб вони були в межах допустимого діапазону для широти (-90 до 90 градусів) і довготи (-180 до 180 градусів). Якщо normalize є False, але check є True, функція переконується, що

значення широти та довготи знаходяться в межах допустимого діапазону без нормалізації.

Потім функція обчислює відстань, використовуючи формулу haversine, яка включає тригонометрію для врахування кривизни Землі.

get_avg_earth_radius(unit) отримує середній радіус Землі в зазначеній одиниці.

_haversine_kernel(lat1, lng1, lat2, lng2) виконує основний розрахунок формули haversine, використовуючи надані широти та довготи.

Формула haversine визначається як:

$$a = sin^2(\Delta lat/2) + cos(lat_1) * cos(lat_2) * sin^2(\Delta lon/2)$$

$$c = 2 * atan2\left(\sqrt{a}, \sqrt{1-a}\right)$$

$$d = R * c$$

Де Δlat - це різниця в широтах, а Δlon - різниця в довготах, R - радіус Землі, а d - відстань між двома точками.

У реалізації передбачається, що інші допоміжні функції, як-от _normalize, _ensure_lat_lon, get_avg_earth_radius, і _haversine_kernel, визначені в коді у додатку. Ці функції обробляють специфіку нормалізації, перевірки та основного розрахунку haversine.

3.6.3 Клас графа:

Клас Graph представляє граф, який шукає алгоритм A*. Він має методи для додавання ребер, отримання сусідів, отримання ваги ребра та встановлення значення `g` вузла.

Функція calculate_distance_between_cities приймає два об'єкти міста (city1 та city2), отримує їхні координати (широту та довготу) і повертає відстань між ними, використовуючи функцію haversine. Вона зображена на рисунку 3.3.

```python
def calculate_distance_between_cities(city1, city2):
    city1_coords = (city1.latitude, city1.longitude)
    city2_coords = (city2.latitude, city2.longitude)
    return haversine(city1_coords, city2_coords)
```

Рис. 3.3 Реалізація функції підрахунку відстані між містами

Функція create_graph створює граф, в якому вершини – це міста, а ребра – це відстані між містами. Порядок дій так сама функція зображена на рисунку 3.4

1. Створюється порожній граф.

2. Всі міста з бази даних додаються як вершини.

3. Між кожною парою міст додається ребро із вагою, що дорівнює відстані між містами.

```python
def create_graph():
    G = nx.Graph()

    # Retrieve all city instances from your database
    cities = City.query.all()

    # Add each city as a node in the graph
    for city in cities:
        G.add_node(city.name, pos=(city.latitude, city.longitude))

    # Add edges between each pair of cities
    for city1 in cities:
        for city2 in cities:
            if city1 != city2:
                distance = calculate_distance_between_cities(city1, city2)
                G.add_edge(city1.name, city2.name, weight=distance)

    return G
```

Рис. 3.4 Реалізація створення графу на мові Python

Функція find_nearest_station знаходить найближчу станцію до заданої локації користувача. Її реалізація зображена на рисунку 3.5.

- Перебираються всі станції, порівнюючи відстані до локації користувача.

- Повертає найближчу станцію і відстань до неї.

```
2 usages
def find_nearest_station(user_location, stations):
    nearest_station = None
    min_distance = float('inf')
    for station in stations:
        distance = haversine(user_location, (station.latitude, station.longitude))
        if distance < min_distance:
            nearest_station = station
            min_distance = distance
    return nearest_station, min_distance
```

Рис. 3.5 Реалізація функції знаходження найближчої станції на мові Python

Функція find_path_in_network використовує алгоритм A* для знаходження найкоротшого шляху у мережі поїздів та автобусів між двома станціями. Її реалізовано на рисунку 3.6.

```
def find_path_in_network(start_station, end_station, train_network_graph):
    # Use A* to find the shortest path
    return astar(train_network_graph, source=start_station.name, target=end_station.name, weight='weight')
```

Рисунок 3.6 Реалізація знаходження шляху між заданими точками на мові Python

Функція calculate_path об'єднує кілька кроків для обчислення загального шляху від локації користувача до пункту призначення, включаючи використання поїздів і пішохідну частину маршруту.

- Знаходить найближчу станцію до локації користувача.
- Визначає шлях.
- Обчислює відстань від кінцевої станції до пункту призначення.
- Об'єднує все для отримання загального шляху та відстані.

Функція calculate_path та її реалізація зображена на рисунку 3.7

```python
def calculate_path(user_location, destination, train_network_graph, stations):
    # Step 1: User Location to Nearest Station
    start_station, distance_to_start_station = find_nearest_station(user_location, stations)

    # Step 2: Train Network Pathfinding
    end_station, distance_to_end_station = find_nearest_station(destination, stations)
    train_path = find_path_in_network(start_station, end_station, train_network_graph)

    # Step 3: Final Leg by Foot
    distance_from_end_station_to_destination = haversine((end_station.latitude, end_station.longitude), destination)

    # Combine results
    total_distance = distance_to_start_station + nx.path_weight(
        train_network_graph,
        train_path,weight='weight') + distance_from_end_station_to_destination

    return {
        "path": train_path,
        "total_distance": total_distance
    }
```

Рисунок 3.7 Реалізація підрахунку відстані на мові Python

Кожна з цих функцій виконує важливу роль у створенні та обчисленні маршрутів у мережі громадського транспорту, включаючи інтеграцію з геолокаційними даними та маршрутизацією.

3.7 Структура бази даних та обробка даних

Ефективна обробка та зберігання даних є ключовим аспектом розробки програми для оптимізації маршрутів подорожей та навігації. У цьому проекті база даних використовується для зберігання географічних даних, інформації про користувачів, історії подорожей та інших даних, необхідних для роботи програми.

3.7.1 Структура бази даних

База даних повинна бути ретельно спроектованою для забезпечення швидкого доступу до даних та ефективного використання ресурсів. Основні таблиці бази даних можуть включати такі:

- Користувачі: Таблиця, яка містить інформацію про зареєстрованих користувачів, включаючи ідентифікатор, ім'я, електронну пошту, хеш пароля та інші відомості.

- Маршрути: Таблиця, яка зберігає дані про створені користувачами маршрути, включаючи назву маршруту, початкову та кінцеву точки, дату створення та інші атрибути.

- Історія подорожей: Таблиця, що містить інформацію про історію подорожей користувачів, включаючи дані про маршрути, дату подорожі, час витрат та інші показники.

- Географічні дані: Для ефективної навігації та оптимізації маршрутів може бути використана таблиця з географічними даними, яка містить координати доріг, місцевості та цікавих місць.

- Додаткові дані: Інші таблиці для зберігання інформації про цікаві місця, ресторани, готелі та інші об'єкти, які можуть бути важливими для користувачів.

3.7.2 Обробка даних

Програмний продукт може підключатися до зовнішніх джерел даних, таких як географічні сервіси та бази даних для оновлення інформації про дороги, дорожні затори та інші дані. Також для перетворення адрес та місцезнаходження на координати (і навпаки) може бути використана геокодувальна служба. За допомогою географічних даних та алгоритмів планування маршрутів можна обчислювати оптимальні маршрути для користувачів.

Програмний продукт може аналізувати дані про дорожні затори, аварії та інші умови на дорозі для надання актуальних рекомендацій користувачам.

Важливо забезпечити безпеку та конфіденційність даних користувачів шляхом шифрування та інших заходів безпеки.

Програмний продукт може взаємодіяти з іншими онлайн-сервісами, такими як сервіси бронювання готелів чи ресторанів, для надання додаткової інформації користувачам.

Зберігання та обробка даних є фундаментальною частиною розробки програмного продукту для оптимізації маршрутів подорожей та навігації.

Ефективна робота з даними дозволить надати користувачам точні та актуальні інформаційні послуги та покращити їх досвід використання програми.

3.8 Оцінка та результати

Оцінка ефективності програмного продукту важлива для забезпечення його якості та задоволення потреб користувачів. В цьому контексті розглянемо основні метрики та критерії оцінки для програми для оптимізації маршрутів та навігації:

1. Швидкість та продуктивність:

- Час розрахунку маршруту: Метрика, яка вимірює час, який потрібен для обчислення оптимального маршруту. Користувачі очікують швидких результатів.

- Час відгуку системи: Тривалість очікування відповіді від системи навігації. Зменшення часу відгуку покращує досвід користувачів.

2. Точність маршрутів:

- Похибка маршруту: Відстань між обчисленим маршрутом та фактичним маршрутом. Маршрут повинен бути максимально точним.

3. Зручність та користувальницька зв'язок:

- Інтерфейс користувача (UI) / Досвід користувача (UX)**: Оцінка вигляду і функціональності інтерфейсу користувача. Важливо, щоб програма була зрозумілою та зручною для використання.

- Наявність інструкцій та довідки: Оцінка доступності інструкцій і довідки для користувачів.

4. Доступність:

- Підтримка різних платформ: Можливість використовувати програму на різних платформах (смартфони, планшети, веб-версія).

-Бар'єри для осіб з обмеженими можливостями: Чи забезпечує програма доступність для осіб з різними видами обмежених можливостей.

5. Надійність:

- Стабільність та відсутність збоїв: Відсутність некоректної роботи програми та падінь.

- Реакція на внутрішні та зовнішні помилки: Як програма реагує на помилки вводу, втрату зв'язку тощо.

6. Функціональність:

- Можливості вибору маршруту: Доступність різних альтернативних маршрутів та урахування різних видів транспорту.

- Наявність додаткових функцій: Наявність додаткових функцій, таких як пошук цікавих місць, інформація про дорожні умови тощо.

8. Безпека даних:

- Шифрування даних: Використання шифрування для захисту особистих даних користувачів.

- Захист від несанкціонованого доступу: Заходи для запобігання несанкціонованому доступу до особистих даних.

9. Продуктивність на великій кількості користувачів:

- Спроможність обробки багатьох запитів одночасно: Можливість обслуговування великої кількості користувачів без втрати швидкості та якості обслуговування.

Зворотній зв’язок показав, що веб застосунок працює, відгук залежить від інтернет в’єднання, але при стабільноній швидкості, запит повертається за 3 секунди, що є доволі швидким для проходження всіх єтапів від отримання сервером запиту до формування та надсилання відповіді користувачеві назад. Мрашрут є доволі точним, похибка від задовільної локації є до п’яти метрів, що є задоввільним, оскільки фінальний пункт є у зримій доступності. Користувачі були задоволені наявності альтернативи декількох побудованих маршрутів та зручності інтерфейсу. Персональні данні були зашифровані. Також було введено систему особистих профілей та паролей для захисту від несанкціонованого доступу. Кількість обробки запитів залежить від мишини, на якій додаток розгортується.

ВИСНОВКИ

У цій дипломній роботі було розглянуто процес розробки програмного продукту, спрямованого на оптимізацію маршрутів подорожей та навігацію. Розглянуті основні аспекти, включаючи аналіз вимог, проєктування системи, реалізацію програмного забезпечення, створення інтерфейсу користувача та обробку даних. Нижче подані основні висновки даної роботи:

1. Аналіз вимог: Перший етап розробки вимагає детального аналізу потреб користувачів та визначення функціональних і нефункціональних вимог до програмного продукту.

2. Проєктування системи: На цьому етапі була розроблена архітектура системи, обрані технології, визначені структура бази даних та алгоритми оптимізації маршрутів.

3. Реалізація програмного продукту: Програмне забезпечення було розроблено згідно з вимогами та архітектурним проектом. Важливим аспектом була оптимізація алгоритмів для забезпечення швидкості та точності обчислень.

4. Дизайн інтерфейсу користувача: Виконано дизайн інтерфейсу, спрямований на зручність користувача та забезпечення інтуїтивно зрозумлих функцій.

5. Обробка даних та база даних: Для забезпечення навігації та оптимізації маршрутів була створена структура бази даних та реалізована обробка географічних даних.

6. Метрики та критерії оцінки: Визначено метрики для оцінки ефективності програмного продукту, включаючи швидкість, точність, зручність використання, безпеку, доступність та інші.

В результаті роботи було створено програмний продукт, який забезпечує користувачам можливість швидкого та точного планування маршрутів подорожей. Перевірка продукту на визначені метрики показала його високу якість та ефективність.

Ця дипломна робота висвітлила важливість ретельного аналізу вимог, дизайну, розробки та тестування програмних продуктів у сфері навігації та маршрутного планування. Результати роботи можуть бути використані для подальших досліджень та розвитку програмних засобів у цій області.

ПЕРЕЛІК ПОСИЛАНЬ

1. Алгоритм імітації відпалу. *www.wikidata.uk-ua.nina.az*. URL: https://www.wikidata.uk-ua.nina.az/Алгоритм_імітації_відпалу.html.
2. Методологія RAD розробки інформаційних систем. *Реферати, курсові, дисертації, дипломи*. URL: https://ua-referat.com/Методологія_RAD_розробки_інформаційних_систем.[Android and iOS] - GIS Geography. *GIS Geography*. URL: https://gisgeography.com/gps-apps-navigation/.
3. 10 GPS Apps For Navigation [Android and iOS] - GIS Geography. *GIS Geography*. URL: https://gisgeography.com/gps-apps-navigation/.
4. Evolution of Navigation Systems - Inventionland. *Inventionland*. URL: https://inventionland.com/blog/evolution-of-navigation-systems/.
5. Genetic Algorithms - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/genetic-algorithms/.
6. GitLab. What is DevOps?. *The DevSecOps Platform | GitLab*. URL: https://about.gitlab.com/topics/devops/.
7. Introduction to A*. A Start algorithm URL: https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html.
8. Pandey M. Dijkstra's Algorithm - Scaler Topics. *Scaler Topics*. URL: https://www.scaler.com/topics/data-structures/dijkstra-algorithm/.
9. SDLC V-Model - Software Engineering - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/.
10. *SumDU Repository: Home*. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/84718/1/Korotenko_bac_rob.pdf;jsessionid=A691962ABDC981455FCEF04CDE35D76D.
11. The Floyd-Warshall Algorithm. *Visualizations of Graph Algorithms*. URL: https://algorithms.discrete.ma.tum.de/graph-algorithms/spp-floyd-warshall/index_en.html.

12. Waterfall metholgy for designing projects
URL: https://business.adobe.com/blog/basics/waterfall#:~:text=The%20Waterfall%20methodology%20–%20also%20known,before%20the%20next%20phase%20begins.
13. What Is Agile Methodology? (A Beginner's Guide) [2023] • Asana. *Asana*.
URL: https://asana.com/resources/agile-methodology.

ДОДАТОК А: Макети інтерфейсу користувача

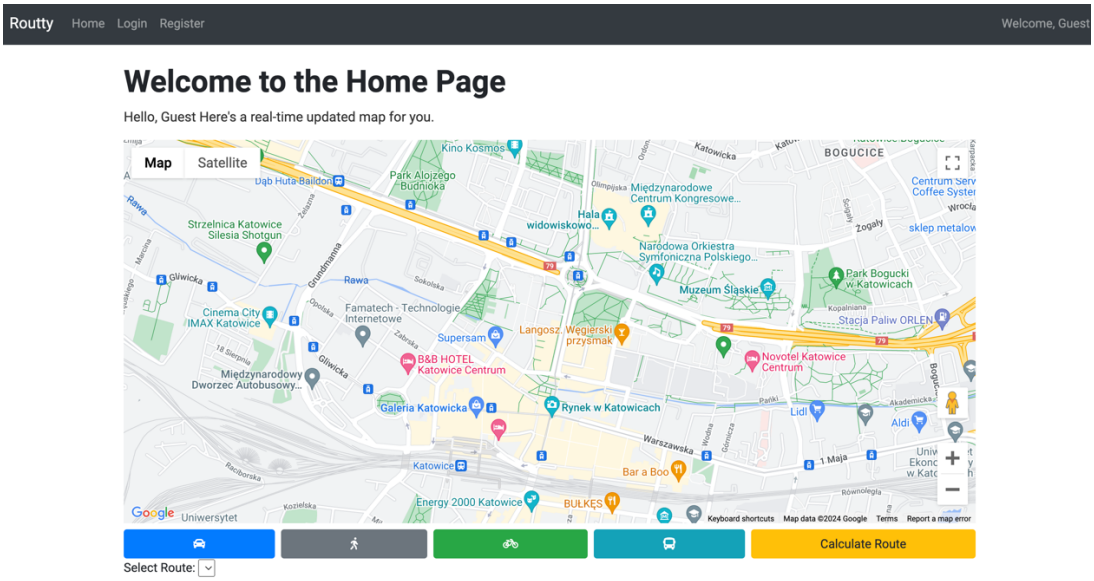

Рис. А.1 Головна сторінка веб-додатку

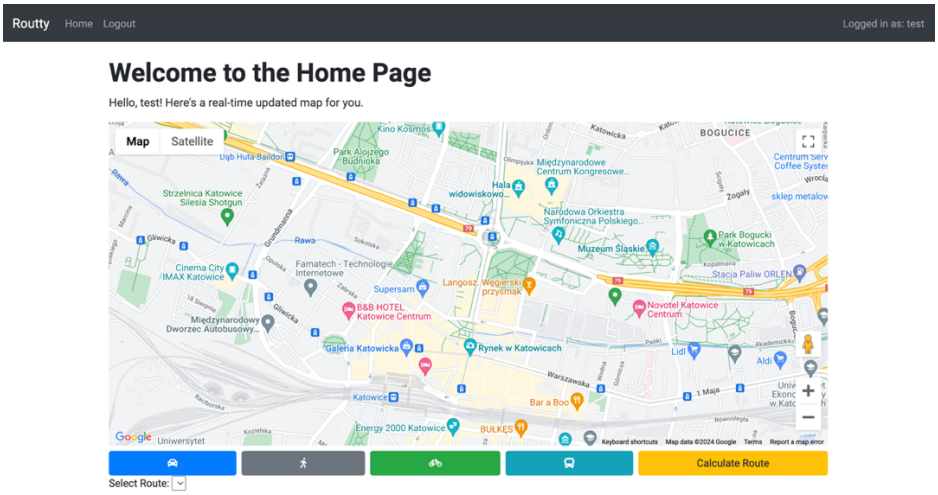

Рис. А.2 Головна сторінка, якщо на сайт зайшов авторизований користувач

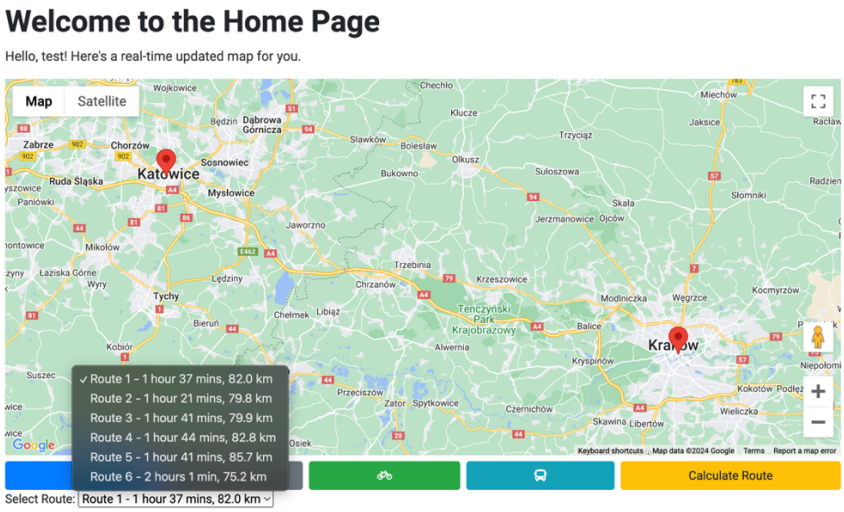

Рис. А.3 Вибір маршруту

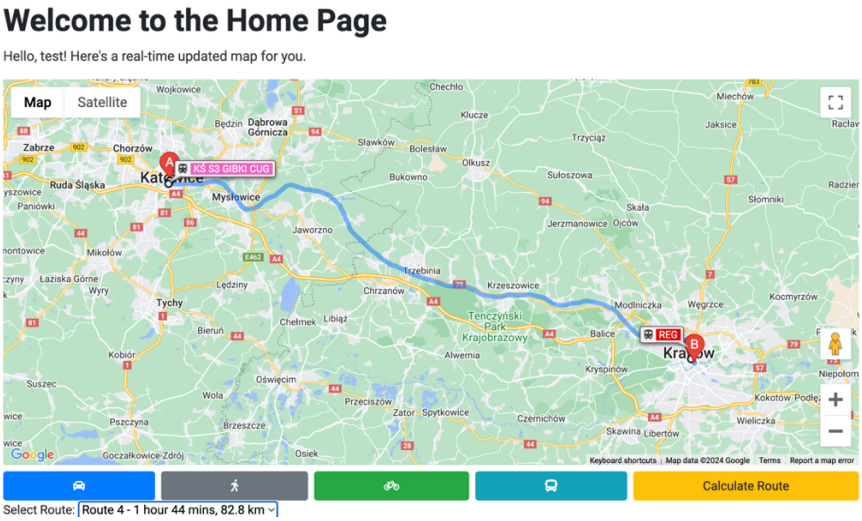

Рис А.4 Побудований маршрут. Спосіб мандрівки транзит

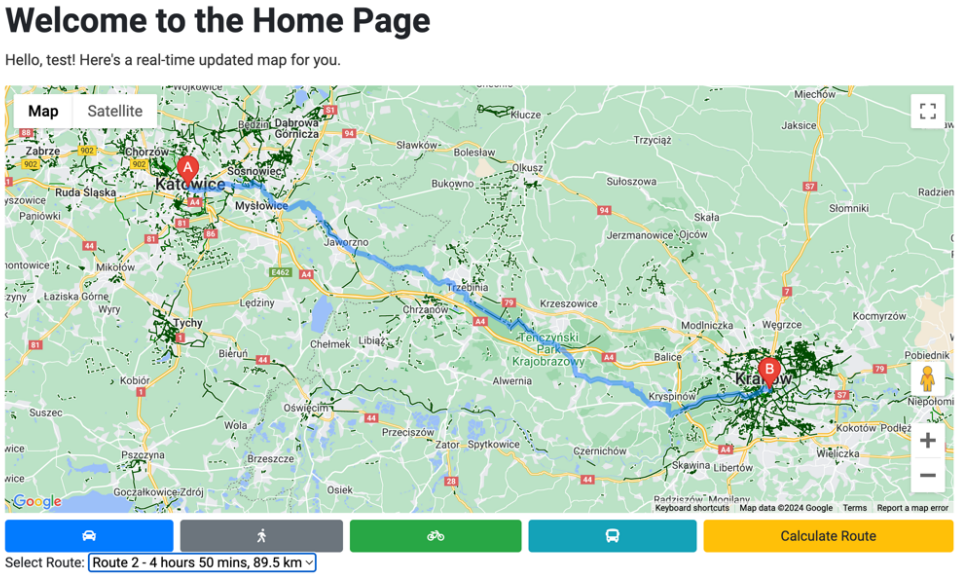

Рис. А.5 Побудований маршрут. Спосіб мандрівки велосипед

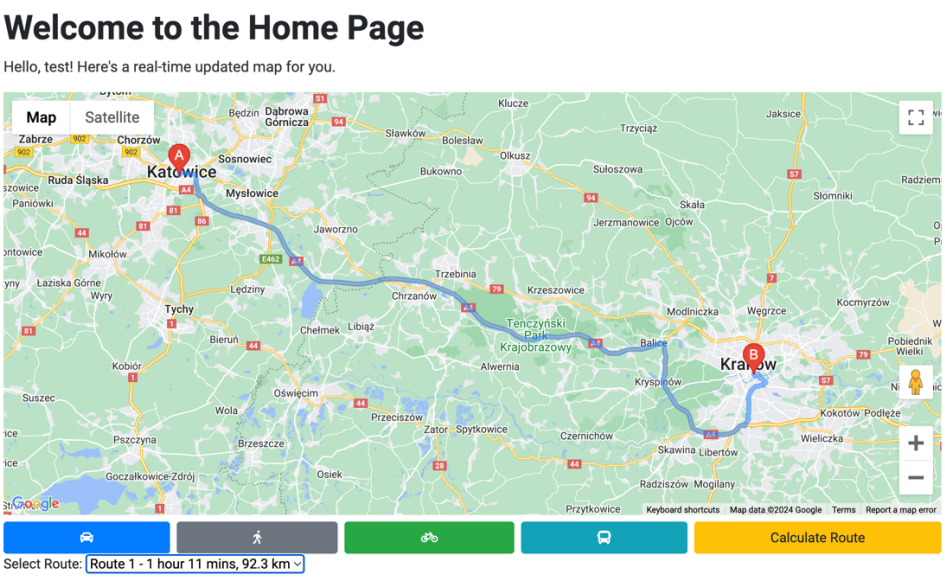

Рис. А.6 Побудований маршрут. Спосіб мандрівки авто

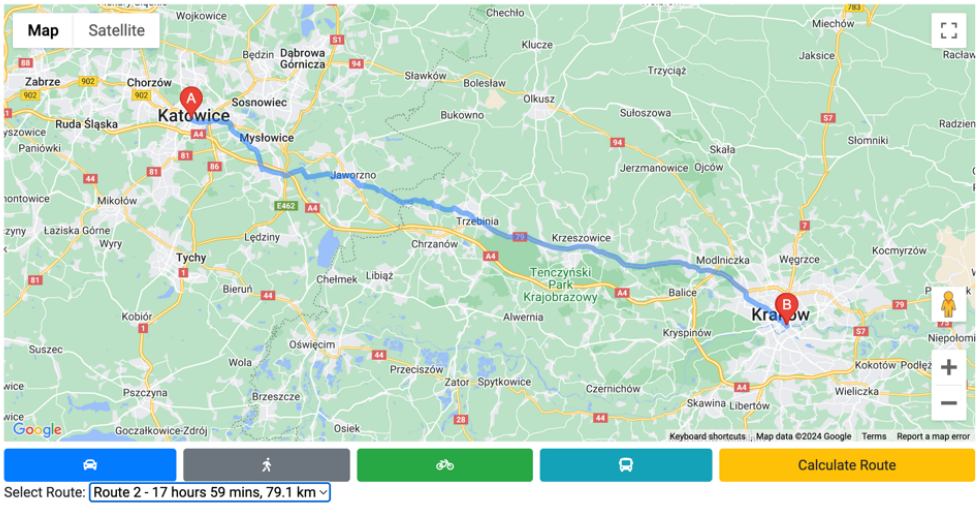

Рис. А.7 Побудований маршрут. Спосіб мандрівки пішки

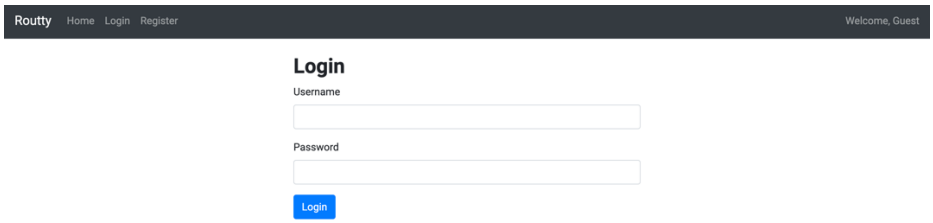

Рис. А.8 Сторінка входу в систему

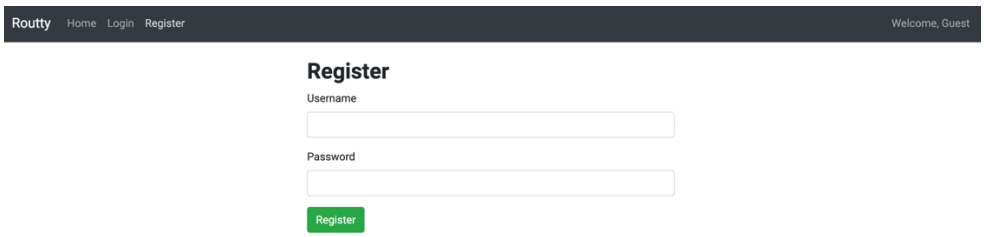

Рис. А.9 Сторінка реєстрації в системі

ДОДАТОК Б: Витрішки коду або псевдокод алгоритмів

```
function displaySelectedRoute() {
    const selectedRouteIndex = parseInt(document.getElementById("routeSelect").value);
    const selectedRoute = allRoutes[selectedRouteIndex];

    console.log("Selected Route:", selectedRoute);

    // Make sure the directionsRenderer is initialized
    if (!directionsRenderer) {
        directionsRenderer = new google.maps.DirectionsRenderer({
            map: map,
            provideRouteAlternatives: true
        });
    }

    // Update the directions renderer with the selected route
    if (selectedRoute) {
        const routeToDisplay = {
            geocoded_waypoints: selectedRoute.geocoded_waypoints,
            routes: [selectedRoute], // Put the selected route in an array
            request: {
                travelMode: selectedTravelMode,
                origin: selectedRoute.legs[0].start_location,
                destination: selectedRoute.legs[0].end_location
            }
        };

        directionsRenderer.setDirections(routeToDisplay);
    } else {
        console.error("Selected route is undefined or improperly formatted");
    }
```

Рис.Б.1 Функція для зображення маршруту

```
    function addMarker(location) {
    if (markers.length >= 2) {
        markers[0].setMap(null);
        markers.shift();
    }

    const marker = new google.maps.Marker({
        position: location,
        map: map,
    });
    markers.push(marker);

    if (markers.length === 2) {
        calculateAndDisplayRoute(false);
    }
}
1 usage
function getUserLocation(callback) {
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(function(position) {
        var userLocation = {
            lat: position.coords.latitude,
            lng: position.coords.longitude
        };
        callback(userLocation);
    }, function() {
        console.error("Error getting the user location");
        // Handle error or provide a default location
    });
} else {
    console.error("Geolocation is not supported by this browser.");
```

Рис. Б.2 Функції для додавання маркеру потребуємих локацій та отримання поточної локації користувача

```
from flask_login import UserMixin

from app import db


4 usages
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(15), unique=True, nullable=False)
    password = db.Column(db.String(80), nullable=False)

    def __repr__(self):
        return f"<Hello, {'guest!' if UserMixin.is_anonymous else self.username}>"
```

Рис. Б.3 Клас користувача

```python
from app import socketio


@socketio.on('connect')
def handle_connect():
    print('Client connected')


@socketio.on('disconnect')
def handle_disconnect():
    print('Client disconnected')


def emit_real_time_data(data):
    socketio.emit('real_time_data', data)
```

Рис. Б.4 Файл для отримання даних у режимі реального часу

```python
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import InputRequired, Length


2 usages
class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[InputRequired(), Length(min=4, max=15)])
    password = PasswordField('Password', validators=[InputRequired(), Length(min=8, max=80)])
    submit = SubmitField('Register')

2 usages
class LoginForm(FlaskForm):
    username = StringField('Username', validators=[InputRequired(), Length(min=4, max=15)])
    password = PasswordField('Password', validators=[InputRequired(), Length(min=8, max=80)])
    submit = SubmitField('Login')
```

Рис. Б.5 Файл форм реєстрації та входу користувача

```python
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))


2 usages
@app.route('/')
def home():
    return render_template('home.html')


1 usage
@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()

    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user and check_password_hash(user.password, form.password.data):
            login_user(user)
            return redirect(url_for('home'))

        flash('Invalid username or password')

    return render_template('login.html', form=form)
```

Рис. Б.6 Файл маршрутизації сайту

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ