МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 01.00.00.000 ПЗ

Група ІПм-22-4

Зікратий Віктор

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Зікратий Віктор Сергійович
(прізвище, ім'я, по батькові)

УДК ___004.942_
(індекс)

МАГІСТЕРСЬКА РОБОТА

Методології побудови оптимальних рекреаційних маршрутів
(назва роботи)

Інженерія програмного забезпечення
(назва освітньої програми)

121 - Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____Зікратий В.С.
(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник__Піх Володимир Ярославович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту
В.о. завідувача кафедри

доц._____Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Рецензент

_____
(посада) (підпис) (дата) (ініціали та прізвище)

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

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
В.о. зав. кафедрою ІПЗ

доц. В.В. Бандура

“ 04 ” вересня 2023 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Зікратому Віктору Сергійовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи Методології побудови оптимальних рекреаційних маршрутів
керівник проекту (роботи) Піх Володимир Ярославович, к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджена наказом закладу вищої освіти від “ 18 ” грудня 2023 р. № 738/7

2. Строк подання студентом проекту (роботи) 15 січня 2024 р.

3. Вихідні дані до проекту (роботи) Алгоритми побудови оптимальних маршрутів, перелік рекреаційних зон в заданій локації

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

1 Методологія побудови оптимальних маршрутів
2 Аналіз існуючих рішень
3 Розробка веб застосунку для побудови оптимальних рекреаційних маршрутів
4 Проектування інтерфейсу веб застосунку
5 Тестування веб застосунку

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

1 Діаграма класів веб застосунку
2 Діаграма застосування веб застосунку
3 Діаграма послідовності побудови маршруту
4 Діаграма додавання місця відпочинку
5 Схема бази даних

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Нормоконтроль	доц. к.т.н. Вовк Р. Б.	
Перевірка на плагіат	доц. к.т.н. Вовк Р. Б.	

7. Дата видачі завдання 04 вересня 2023 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури	20.09.2023	виконано
2	Аналіз підходів до побудови оптимальних маршрутів	01.10.2023	виконано
3	Вибір алгоритму побудови оптимальних маршрутів та його вдосконалення на основі заданих параметрів	20.10.2023	виконано
4	Вибір інструментів розробки вебзастосунку та його реалізація	15.11.2023	виконано
5	Тестування роботи вебзастосунку	03.12.2023	виконано
6	Затвердження пояснювальної записки роботи завідувачем кафедри	15.01.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 85 с., 34 рис., 14 табл., 28 джерел.

Тема: Методології побудови оптимальних рекреаційних маршрутів.

Об’єкт дослідження: методи побудови оптимальних маршрутів між заданими локаціями.

Мета роботи: створення конкурентоспроможної системи для побудови оптимальних маршрутів відвідування рекреаційних зон в межах певної території.

Предмет дослідження: алгоритм побудови оптимальних маршрутів з врахуванням додаткової інформації про відвідувані місця

Результати дослідження:

Розроблено алгоритм побудови маршрутів між рекреаційними зонами з врахуванням їх рейтингу, запланованих видів відпочинку та можливих подій що можуть бути в них заплановані. На основі розробленого алгоритму розроблено вебдодаток для побудови оптимальних рекреаційних маршрутів.

Висновок:

В результаті досліджень було запропоновано алгоритм побудови рекреаційних маршрутів та реалізовано його у вигляді вебдодатку.

КЛЮЧОВІ СЛОВА: ОПТИМАЛЬНИЙ МАРШРУТ, ЗАДАЧА КОМІВОЯЖЕРА, ВЕБЗАСТОСУНОК, РЕКРЕАЦІЙНА ЗОНА

ANNOTATION

Master's thesis: 85 pages, 34 figures, 14 tables, 28 sources.

Topic: methodologies for constructing optimal recreational routes.

Research object: methods for constructing optimal routes between specified locations.

Objective: creating a competitive system for building optimal routes to visit recreational areas within specific territory.

Subject of research: алгоритм побудови оптимальних маршрутів з врахуванням додаткової інформації про відвідувані місця

Research results:

There was developed an algorithm for constructing routes between recreational areas, considering their rating, planned types of recreation and possible events that may be scheduled in them. Based on the developed algorithm a web application has been created for building optimal recreational routes.

Conclusion:

As a result of the research, an algorithm for building recreational routes was proposed and implemented in a form of web application.

KEYWORDS: OPTIMAL PATH, TRAVELLING SALESMAN PROBLEM, WEB APPLICATION, RECREATIONAL ZONES

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

TSP – Traveling Salesman Problem, завдання комівояжера

NP - Nondeterministic Polynomial time, недетермінований поліноміальний час

CRUD – Create/Read/Update/Delete, створення, читання, зміна та видалення певної сутності

REST – Representational State Transfer, передача репрезентативного стану

СУБД – Система управління базою даних

AJAX – Asynchronous JavaScript transfer, асинхронна передача коду на JavaScript

HTML – Hyper Text Markup Language, мова розмітки гіпертексту

CSS – Cascading Style Sheets, каскадні таблиці стилів

DTO – Data transfer object, об’єкт для передачі даних

ВСТУП

Актуальність роботи

Рекреаційна зона – це спеціально відведене та облаштоване місце для відпочинку, розваг та туризму. Вони можуть бути як природними, так і штучними, і розташовуватися в різноманітних ландшафтах: від міських парків до гірських курортів.

До природних рекреаційних зон можна віднести: ліси, парки, озера, річки, пляжі та інші природні місця, які використовуються для відпочинку та розваг.

До штучних рекреаційних зони відносяться парки розваг, аквапарки, музеї, театри, кінотеатри, спортивні комплекси та інші місця, які були створені руками людини для відпочинку та розваг.

До комбінованих рекреаційних зон відносяться зони, що включають в себе як природні, так і штучні елементи. Наприклад, парк, який розташований на території лісу, або пляж, який обладнаний кабінками для переодягання і душем.

Рекреаційні зони відіграють важливу роль у житті людей. Вони сприяють:

– відпочинку та розслабленню: рекреаційні зони забезпечують людям можливість відпочити від роботи, навчання чи інших повсякденних справ, розслабитися і відновити сили.

– зміцненню здоров’я: активний відпочинок у рекреаційних зонах допомагає зміцнити здоров’я, поліпшити фізичну форму та самопочуття.

– розвитку туризму: рекреаційні зони приваблюють туристів, сприяючи розвитку місцевої економіки.

– збереженню природних ресурсів: рекреаційні зони допомагають зберігати природні ресурси, оскільки вони зазвичай розташовані в екологічно чистих місцях.

Сьогодні вільний час людини стає все цінніше, а чим більшою стає кількість місць для відпочинку, тим важче обрати конкретну рекреаційну зону. Проблема стає ще актуальнішою, коли хочеться відвідати декілька місць. Також дуже часто виникає така ситуація, коли людина приїжджає в нове місто і не знає

як розпланувати свою подорож. Тому пошук певних нових місць для проведення відпочинку стає все більш затратним по часу.

У той же час інтерактивні системи, що спрощують певні задачі стають все популярнішими. І щоб зекономити час користувачу при плануванні свого дозвілля, було вирішено створити систему побудови оптимального маршруту по рекреаційних зонах у вигляду WEB-застосунку.

На сьогоднішній день в Україні схожих рішень є досить мало і основним недоліком аналогів є неможливість побудови оптимального маршруту з місць відпочинку. Також немає системи яка б об’єднала місця громадського харчування і місця для проведення дозвілля в одну базу.

Зв'язок роботи з науковими програмами, планами, темами

Дана робота відповідає сучасним планам та програмам галузі, так як вона розвиває тему ефективного використання інженерії програмного забезпечення в інших галузях, зокрема у сфері туризму. Також практична актуальності запропонованих алгоритмів та моделей є важливою для вирішення потреб галузі.

Мета і задачі дослідження

Метою даної роботи є створення конкурентоспроможної системи для побудови оптимальних маршрутів відвідування рекреаційних зон в межах певної території.

Створена система є корисною жителям міст, які б хотіли спланувати відвідування певних рекреаційних зон, з врахуванням їх призначення (активний відпочинок, плавання, можливість організації пікніка і т.п.).

Об'єктом дослідження даної магістерської роботи є розробка методології та підходів до побудови і оптимізації маршрутів відвідування рекреаційних зон з врахуванням заданих обмежень.

Предметом дослідження даної магістерської роботи є алгоритми пошуку оптимальних маршрутів.

Методи дослідження. Були використані наступні методи у реальному проекті:

- Методика аналізу складності алгоритмів

- Основи проектування баз даних

Наукова новизна одержаних результатів полягає в тому що отримали розвиток технології актуальні для багатьох галузей.

Практичне значення одержаних результатів.

Результат даної роботи має потенціал пришвидшити розвиток туристичної галузі держави, а також залучити кошти іноземного капіталу для масштабування даної системи в інших країнах.

Особистий внесок.

Проведено вдосконалення відомого алгоритму пошуку оптимальних маршрутів з врахуванням заданих обмежень, здійснено розробку веб додатку на основі розроблено алгоритму та проведено його тестування.

Структура та обсяг магістерської роботи.

Магістерська робота представлена на 86 сторінках друкованого тексту, який включає в себе вступ, три розділи, висновки та список використаних джерел (42 найменування).

1 МЕТОДОЛОГІЯ ПОБУДОВИ ОПТИМАЛЬНИХ МАРШРУТІВ

1.1 Основні характеристики завдання комівояжеру

Комбінаторика – це розділ математичної науки, що вивчає комбінації та їх кількість, складені з тих чи інших умов із заданих об'єктів. Вперше питання, які вивчає комбінаторика, були підняті в працях математиків Стародавньої Греції, Індії та Китаю. Інтерес до предмету збільшився в 19-му і 20-му століттях у зв'язку з розвитком теорії графів.

У комбінаторики є багато застосувань в інших областях математики, включаючи теорію графів, програмування, криптографію і теорію ймовірності. Серед провідних математиків цій галузі можна виділити Блеза Паскаля, Якоб Бернуллі, Леонхард Ейлера і Пала Ердеша [5].

Завдання комівояжера (Travelling salesman problem, TSP) – класична задача комбінаторики. Щоб наочніше показати її значимість, нижче наведена коротка історія вирішення поставленого завдання.

Математичні проблеми, пов'язані із завданням комівояжера, були вивчені в 1800-их роках ірландським математиком сером Вільямом Роуеном Гамільтоном і британським математиком Томасом Пенінгтон Кіркменом. У 1857 Гамільтон створив гру Ікосіан, в правилах якої сказано, що учасники повинні з'єднати 20 точок додекаедру так, щоб кожна точка використовувалася не більше одного разу, також кінцева точка шляху повинна збігатися з вихідною. Ця гра лягла в основу появи гамільтонова графа [6].

Завдання пошуку оптимально шляху була вперше розглянута з математичної точки зору в 1930-их роках математиком і економістом Карлом менджер у Відні. У 1940-их статистики Механобіс, Джессен, Гош і Маркс намагалися знайти застосування задачі комівояжера в сільськогосподарському секторі, що призвело до її популяризації починаючи з середини 1950-х роках методи розв'язання задачі стали публікуватися в наукових журналах.

Хоча завдання комівояжера проста для розуміння, її вкрай складно вирішити [7]. У 1972 Річард М. Короп показав, що завдання Гамільтона циклу належить до класу NP-повних задач (Nondeterministic Polynomial time), які має на увазі NP-складність завдання TSP [8]. Це відкриття дало наукове пояснення очевидною обчислювальної труднощі знаходження оптимальних маршрутів. Методи рішення TSP ставали більш складними, кількість міст зростала.

Нижче представлена коротка історія етапів розв'язання задачі комівояжера.

У 1954 Данциг, Фалкерсона і Джонсон видали опис методу для вирішення TSP і практично проілюстрували його, вирішивши завдання з 49 містами, з'єднавши таким чином міста кожного штату Америки. У 1971 році Гельд і Карпо вирішили задачу пошуку оптимального шляху між 64 містами.

Пізніше в 1977 році, Мартін Гротчел побудував шлях між 120 німецькими містами. У 1973 Лін і Керниган знайшли застосування завдання TSP в інженерії – вони поєднали 318 пунктів, отриманих в результаті різання лазером.

У 1987 році Голландія і Гроешел була вирішена задача з 666 містами, пізніше в цьому ж році Падберг і Рінальді знайшли оптимальний тур, що зв'язує вже 2 392 міста (рис. 1). У 1994 році кількість міст збільшилася до 7397, через 10 років кількість міст досягло до 24978. У 2006 було отримано рішення задачі комівояжера з 85900 містами [5].

1.2 Методики побудови маршрутів між рекреаційними зонами

Розробка туру включає такі етапи: вибір пунктів маршруту, ієрархізація цих пунктів, вибір пунктів початку та закінчення маршруту, після чого провадиться розробка схеми маршруту та його оптимізація. Розробка схеми маршруту та його програмне забезпечення є нерозривно пов'язаними паралельними процесами. Розрахунок вартості туру грунтується на проведених організаційно-технічних заходах з пошуку партнерів та укладених з ними угод.

Етап І. Вибір пунктів маршруту. Критерієм відбору є привабливість об'єктів показу для задоволення мети подорожі, можливість забезпечити

різноманітність програми перебування в даному населеному пункті, його транспортна доступність та забезпеченість послугами гостинності.

Етап II. Ієрархізація пунктів маршруту провадиться за вказаними вище критеріями, при цьому основна увага повинна бути звернута на клас гостинності та транспортну доступність обраних пунктів маршруту. Метою даного етапу є виділення диференціація пунктів маршруту відповідно до мети подорожі з виділенням пунктів дислокації з тривалим програмним забезпеченням та екскурсійних пунктів.

Етап III. Вибір початкового та кінцевого пунктів маршруту здійснюється за показником транспортної доступності, тобто зв'язності з місцем постійного проживання потенційних туристів (зоною дії туроператора), взаємозамінності видів транспорту, типу транспортних засобів.

Етап IV. Розробка схеми маршруту. Схема маршруту залежить від обраної форми. Маршрут - це напрямок переміщення туриста. За схемою маршрути можуть бути лінійні, кільцеві, радіальні та комбіновані, (рис. 1.1). Вибір пунктів по маршруту узгоджується з програмою відповідно до виду туризму, терміну та класу обслуговування.

На рис.1.1 прийняті наступні умовні позначення: Пп - початковий пункт маршруту; Пр – проміжний пункт з порядковим номером 1,2,...n; Кп – кінцевий пункт маршруту; → напрямок руху

Вибір схеми маршруту залежить від транспортної системи: конфігурації транспортної мережі, її густоти та технічного стану, рівня розвитку окремих видів транспорту, рівня розвитку транспортної інфраструктури, що забезпечує надійність та безпеку роботи транспорту.

Найпоширенішим випадком при розробці схеми є варіант, коли за обмежений термін при мінімізації витрат часу на переміщення між основними пунктами маршруту бажано забезпечити максимально можливу інформативність подорожі, тобто охопити якнайбільше об'єктів показу задля задоволення пізнавальної мети. Такий варіант отримав назву «задача комівояжера».

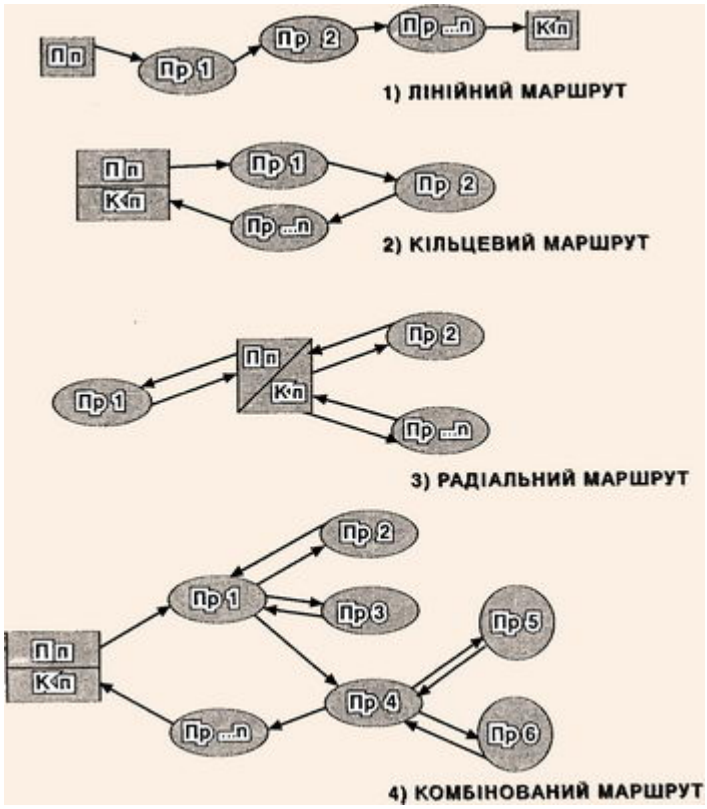

Рис. 1.1 Види маршрутів

При вирішенні цієї задачі можна застосовувати методичний апарат теорії графів. Графо-аналітичні методи дають можливість виокремити найсуттєвіші елементи, унаочнити наявну інформацію, обгрунтувати відбір та ієрархізацію пунктів маршруту. Задача зводиться до побудови графа – математичного відображення просторової організації туристичного продукту. Граф являє собою певним чином організовану кінцеву множину вершин і множинність ребер. Вершини і ребра є елементами графа, що роблять його зв'язним. Граф називається метризованим, коли його елементам надається певне значення, наприклад, метризація вершин задається атрактивністю об'єктів відвідування в балах, місткістю та класністю готельної бази, бальною оцінкою різнорідності програми тощо; ребра графа відображають наявний зв'язок між вершинами і в нашому випадку це перш за все транспортний зв'язок, який відтворює або тільки наявність шляхів сполучення, або їх категорію чи полімагістральність. Значення ребер може задаватися відстанню (в км) між пунктами маршруту або часом, необхідним для його долання тим чи іншим видом транспорту (такий варіант кращий, коли немає взаємозамінності транспортних засобів). Тобто сам процес

побудови графа є певним впорядкуванням та генералізацією інформації відповідно до мети подорожі. Таким чином, ми маємо графічне зображення територіальної структури майбутнього турпродукту, що задана елементами графу. Вершинами графа (V) є пункти, обрані за визначеними критеріями, а ребрами (E) – наявний зв'язок між ними, шляхи сполучення певної категорії.

Оцінка положення кожної вершини в графі, визначення місця в загальній системі атрактивних ресурсів, а також оцінка графа в цілому (його зв'язність, конфігурація) і його параметризація за витратами часу провадиться за допомогою топологічних мір. Ці міри визначаються на множині відношень між елементами графа. Виділяють міри концентрації та диференціації, за якими оцінюється положення вершин в графі, і міри інтеграції та композиції, що дають змогу оцінити граф в цілому.

Розглянемо умовний приклад. На певній території за умов атрактивності туристичних ресурсів, рівня розвитку рекреаційних зон та транспортного забезпечення виділені п'ять пунктів, що зв'язані між собою автошляхами та залізничним сполученням (рис.1.2). Треба оцінити положення кожної вершини в графі. Для цього використовується показник центральності (A), що визначається за кількістю інциденцій (це кількість ребер, що виходять з даної вершини) - таблиця 1.1.

Таблиця 1.1

Показники оцінки вершин графа

Показники	**1**	**2**	**3**	**4**	**5**
1) центральності	1	3	3	1	2
2) ієрархічності	4	1	2	4	3

Як видно з рис.1.2, найкраще положення в графі займають пункти V2 та V3, але в пункті V2 наявний міжнародний аеропорт, тому за показником ієрархічності він займає перше місце і визначений як початковий пункт

маршруту. Пункти V1 та V4, які мають найнижчі показники центральності та ієрархічності, обираються як екскурсійні пункти, що забезпечують програму перебування відповідно в пунктах V2 та V3. Схема маршруту в даному випадку складається як комбінована і має два варіанти руху: 1) V2 - V5 - V3 чи 2) V2 - V3 - V5.

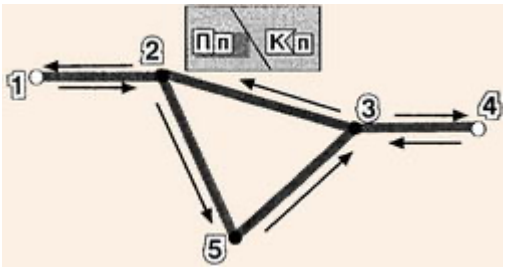

Рис 1.2 Приклад прокладання туристичного маршруту
1-5 – пункти маршруту, з них 2,3,5 – пункти ночівлі,
1,4 – екскурсійні пункти; стрілками показаний напрямок руху

Варіативність побудови маршруту визначається за показником цілісності (μ), який в теорії графів називається цикломатичним числом і показує кількість замкнених циклів в графі (чим більше μ – тим цілісніше є граф і більше варіантів маємо при побудові схеми маршруту). Оскільки маємо різні варіанти побудови схеми руху по маршруту, суттєвим є оцінка графа в цілому задля порівняльної параметризації пропонованих схем. Для цього використовується показник зв'язності (β) який обраховується як відношення суми ребер графа до суми його вершин:

$$\beta = E / V \qquad (1.1)$$

Показники міри композиції дають змогу оцінити конфігурацію та компактність пропонованої схеми маршруту. Загальну конфігурацію графа оцінюють за показником форми (π), який обраховується за формулою:

$$\pi = E / \delta \qquad (1.2)$$

де δ – діаметр графа або мінімальна кількість ребер, що з'єднує максимально віддалені його вершини. Чим більше значення показника δ, тим більш компактну форму має схема маршруту. Щоб оцінити саме компактність пропонованої схеми використовуюсь показник компактності (η), який визначається за формулою:

$$\eta = \varepsilon / V \qquad (1.3)$$

де ε – периметр графа, тобто сукупність ребер, що є зовнішньою гранню графа. Показник компактності оцінює протяжність графа – чим менше його значення, тим компактніший граф, тобто тим менші витрати часу на долання відстаней між пунктами маршруту (таблиця 1.2).

Таблиця 1.2

Оцінка схеми маршруту (відповідно до рис. 1.2)

μ	β	δ	π	ε	η
1	1	3	1,7	4	0,8

Показники задаються по кожному варіанту схеми маршруту. Таким чином, обирається комбінована схема маршруту першим варіантом руху (V2 - V5 - V3).

Етап V. Оптимізація машруту полягає у встановлені ряду об'єктивних та суб'єктивних обмежень. Об'єктивні обмеження виходять з умов сегментації ринку, а суб'єктивні визначаються можливостями туроператора. Обмеження визначаються перш за все цільовим споживчим сегментом (вік, життєвий цикл сім'ї, стиль та рівень життя тощо). Часові обмеження стосуються сезонності та терміну подорожі. Економічні обмеження виходять з умов функціонування

ринку (кон'юнктура ринку, стан конкурентного середовища, цінова політика тощо).

Основним обмеженням на початковому етапі розробки туру є обмеження в часі та засобах по забезпеченню комфортності подорожування. Саме часові обмеження визначають вибір транспортних засобів та їх тип під час проходження маршруту. Вибір транспортного засобу слід розглядати як введення певних обмежень в організаційні параметри туру (кількість туристів в групі і відповідно вибір форми туру впливають на його організацію та вартість).

1.3 Аналіз алгоритмів побудови оптимальних маршрутів

Алгоритми для вирішення завдання комівояжера можна розділити на точні (exact algorithm) і неточні (non-exact algorithm). Точні алгоритми включають в себе перебір всіх можливих варіантів, в окремих випадках рішення можуть бути швидко знайдені, але в цілому здійснюється перебір $n!$ циклів. Другі в загальних випадках застосовуються для задач, які неможливо вирішити точно (обчислення певних інтегралів, рішення нелінійних рівнянь, витяг квадратного кореня.), якщо існуючі точні рішення вимагають значних і невиправданих витрат часу при високій складності завдання, а також як частина більш складного алгоритму, з допомогою якого завдання вирішується точно [11].

1.3.1 Основні характеристики точних алгоритмів

В загальному існує дві групи точних алгоритмів. Перша група використовує методи релаксації лінійного програмування TSP (Travelling salesman problem;), наприклад, алгоритм Гомори, метод внутрішньої точки, метод гілок і меж. Друга група, використовує методи динамічного програмування. Характерною особливістю методів обох груп є гарантія знаходження оптимальних рішень при загальній трудомісткості процесу [11].

Опис алгоритму повного перебору (Brute Force)

Один з найбільш очевидних методів вирішення задачі комівояжера є метод повного перебору або грубої сили. Його суть полягає в переборі всіх можливих варіантів шляхів, алгоритм вирішення можна записати як:

– визначити загальне число можливих гамільтонових циклів;

– визначити вагу кожного гамільтонового циклу, склавши вага всіх його ребер;

– вибрати гамільтонів цикл з мінімальною вагою, який і буде оптимальним.

Га́мільтонів шля́х – шлях, що містить кожну вершину графу рівно один раз. Гамільтонів шлях, початкова і кінцева вершини якого збігаються, називається гамільтоновим циклом.

Метод повного перебору має низку переваг. Він гарантує знаходження рішення задачі TSP, при цьому він прямолінійний і простий у виконанні. У той же час, алгоритм вважається неефективним при роботі з великим об'ємом даних, так як для знаходження оптимального маршруту вимагає знаходження ваг $(n – 1)!$ гамільтонових циклів.

Опис методу гілок і меж (Branch and Bound)

Алгоритм гілок і меж, запропонований у [11] і докладно розглянутий у [12], є одним із ряду евристичних алгоритмів, запропонованих для вирішення задачі комівояжера.

У процесі рішення задачі комівояжера будується дерево пошуку рішення. При побудові дерева на кожному кроці вибирається деяка гілка. Вибір гілки здійснюється на основі оцінки. Оцінка визначає, на скільки збільшиться шуканий шлях, якщо гілка, що оцінюється, не буде використано в цьому шляху. З оцінок гілок піддерева формується оцінка піддерева і в процесі пошуку рішення поточна оцінка порівнюється з оцінками інших піддерев і при перевищенні або рівності, алгоритм передбачає продовження пошуку з використанням інших, більш

вигідних (з точки зору оцінки, що використовується) піддерев. Тобто, алгоритм гілок та меж є варіантом пошуку з поверненням.

При виборі гілки в процесі пошуку рішення використовується гілка, що дає найбільше зростання оцінки піддерева у разі її використання. Вибір такої гілки здійснюється за наступним алгоритмом: у вихідній матриці цін з кожного рядка та кожного стовпця віднімається мінімальний елемент. Отримуємо перетворену матрицю. При цьому в кожному рядку та стовпці з'являється елемент нульової вартості (довжини). Гілки нульової вартості розглядаються як можливі гілки для побудови шуканого шляху [7].

Далі кожен нульовий елемент матриці замінюється на нескінченність і визначається мінімальний елемент у кожному рядку та стовпці.

Отримуємо оцінку кожної з гілок нульової вартості, тобто, оцінюємо програш у разі не використання оцінюваної гілки.

Тобто, для оцінки гілки можна записати

$$\Delta a_{ij} = max\left\{\min\left\{\begin{matrix} a_{ik} \\ k \neq j \end{matrix}\right\} + min\left\{\begin{matrix} a_{kj} \\ k \neq i \end{matrix}\right\}\right\} \quad (1.4)$$

$k = 1,2,3,K,(n-1), n$

де Δa_{ij} – оцінка гілки

Вираз (1.4) враховує елементи *i*-го рядка і *j*-того стовпця матриці цін, відповідні аналізованої гілки. Оцінку гілки можна розширити, враховуючи більшу кількість елементів матриці цін, а саме гілки, що входять у вузол і, і гілки, що виходять із вузла *j*, тобто, *j*-тий рядок і *i*-ий стовпець і збільшити, таким чином, вдвічі число аналізованих елементів матриці цін. Це відповідає іншим гілкам суміжним з гілкою a_{ij} і поки не використовується для оцінки гілки.

Тоді для розширеної оцінки можна записати

$$\Delta_p a_{ij} = max\left\{\min\left\{\begin{matrix} a_{ik} \\ k \neq j \end{matrix}\right\} + \min\left\{\begin{matrix} a_{kj} \\ k \neq i \end{matrix}\right\} - \min\left\{\begin{matrix} a_{jk} \\ k \neq i \end{matrix}\right\} - \min\left\{\begin{matrix} a_{ki} \\ k \neq j \end{matrix}\right\}\right\} \quad (1.5)$$

$k = 1,2,3,K,(n-1), n$

де $\Delta_p a_{ij}$ – розширена оцінка гілки

Вираз (1.5) враховує додаткові гілки, що у розширеній оцінці. Ці гілки на рис. 1.3 виділено зменшеною товщиною ліній.

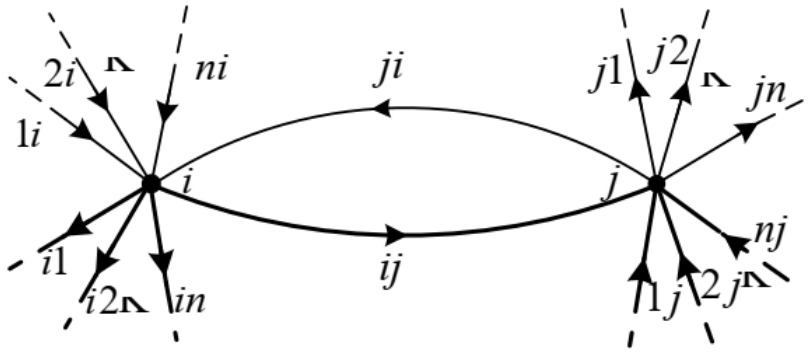

Рис 1.3 Додаткові гілки, що використовуються у розширеній оцінці

Додаткові гілки, що використовуються у розширеній оцінці (1.5), дають можливість виділити гілки, що мають найкраще продовження, тобто продовження мінімальної довжини.

Кількість можливих рішень одно (n – 1)! / 2, для n = 50 це приблизно 3 × 1062. Цей метод найбільш часто використовується при кількості вузлів від 40 до 60 [17].

Необхідність цілком вирішувати завдання лінійного програмування у всій області допустимих рішень можна вважати головним недоліком описаного вище методу. Для задач з великим об'ємом даних метод гілок і меж є невиправдано трудомістким, в той же час алгоритм є надійним методом вирішення цілочислових задач.

Опис алгоритму Гомори (The Cutting Plane)

У 1954 році була представлена робота Данцига, Фалкерсона і Джонсон, що описує новий метод розв'язання задачі комівояжера, який також може бути використаний для вирішення будь-якої проблеми

$$\min c^T x \; subject \; to \; x \; \in S \tag{1.6}$$

де $c \neq 0$, S – кінцеве підмножина деякого простору, і таким чином це зможемо знайти точки S. Це ітераційний алгоритм. Кожне повторення починається з лінійної програмної релаксації. Запишемо у вигляді формули:

$$\min c^T x \; subject \; to \; Ax \; \leq b \tag{1.7}$$

де багатогранник P, такий що $x \in Ax$, містить S і обмежений. Так як P обмежений, можемо знайти оптимальне рішення x^* як екстремальну точку P. Якщо x^* належить S, то оптимальне рішення знайдено (1.7); в іншому випадку деяка лінійна нерівність задовольняє всі точки S і порушує x [13].

Даний метод використовується для побудови точних або наближених задач, особливо часто зустрічається в поєднанні з методом гілок і меж. Обидва методи засновані на вирішенні послідовності релаксованих підзадач лінійного програмування.

В алгоритмі Гоморі релаксовані підзадачі поступово покращують апроксимацію цілочисельного завдання, зменшуючи околицю оптимального рішення. Якщо оптимальність не вдалося отримати, тоді шукається наближене рішення з похибкою.

У методу відсікань є перевага над методом гілок і меж – перший більш зручний для апаратного обчислення, так як для його рішення не потрібен великий обсяг оперативної пам'яті для зберігання дерева рішень [12].

Опис методу динамічного програмування (Dynamic Programming)

Розглянемо задачу з n містами і відстанями d_{ij} між будь-якими двома містами, шлях починається і закінчується в місті n. TSP може бути вирішена за

час $O(n!)$ методом повного перебору, але алгоритм динамічного програмування дозволяє скоротити час до $O(n*627)$.

Визначимо підзадачу: нехай S буде підмножиною міст, що містить 1 і як мінімум ще одне місто, j буде містом відмінним від 1, позначимо через $C(S, j)$ найкоротший шлях, що починається в 1, що проходить всі міста S і закінчується в j.

Запишемо алгоритм у вигляді псевдо-коду.

```
for all j do C({1,j},j) := d1j
for s:= 3 to n do (the size of the sunsets considered this round)
for all subsets S of {1,…, n} of size n and containing 1 do
for all j ∈ S, j ≠ 1 do
C(S, j) := min i ≠ j, i ∈ S [C(S – {j}, i) + dij]
opt := min j≠1 [C({1,2,…, n}, j) + dj1]
```

Даний метод використовується для підвищення ефективності обчислювальних повторень, зберігаючи проміжні результати і знову використовуючи їх при необхідності [10].

1.3.2. Основні характеристики неточних алгоритмів

В цілому алгоритми даної групи пропонують потенційно неоптимальні, але швидкі рішення. У свою чергу наближені алгоритми можна розділити на дві категорії: наближені (Approximation Algorithms) і евристичні (Heuristic Algorithms).

Опис алгоритму Крістофідеса (Christofides 'Algorithm)

Алогорітм Крістофідеса використовується для вирішення метричних TSP – з додатковою умовою, що для матриці відстаней виконано нерівність трикутника:

$$\forall i, j, k \; d_{ik} \geq d_{ij} + d_{jk} \tag{1.8}$$

Велика частина евристичних алгоритмів належать до 2 наближеного класу. Професор Нікос Крістофайдс в 1976 році допрацював один з існуючих алгоритмів (метод подвійного мінімального остовного дерева, $O(n^6 log_6(n))$) так, що час вирішення завдання не перевищує оптимальний час більш ніж на 3/2 [9].

Рішення оригінального алгоритму можна записати так: знайти мінімальне дерево з множини всіх міст, продублювати всі ребра і побудувати граф Ейлера, побудувати гамільтонів цикл, пройшовши кожен вузол тільки один раз і вибираючи найкращий шлях, що веде з кожного вузла.

Алгоритм Крістофідеса складається з послідовності наступних дій:

– визначити мінімальне дерево з множини всіх міст;

– визначити паропоєднання з мінімальною вагою множини вершин непарного степеня і побудувати граф Ейлера;

– визначити ейлерів обхід і побудувати гамільтонів цикл, уникаючи відвідуваних вузлів [14].

Основна відмінність – додаткове обчислення паропоєднання з мінімальною вагою. Ця частина також найбільш трудомістка, тому час виконання алгоритму зростає до $O(n^1)$. Проведені тести показали, що алгоритм Крістофідеса на 10% вище нижньої межі Хелд-Карп [15].

Опис алгоритму найближчого сусіда (NearestNeighbour)

Один з найпростіших евристичних методів рішення TSP. Головне правило алгоритму – завжди вибирати сусіднє місто (сусіда). Рішення завдання складається з наступних кроків:

– вибрати будь-яке місто;

– визначити сусіднє місто, не включений в маршрут, і перейти в нього;

– перевірити чи залишилися міста, не включені в маршрут, якщо відповідь позитивна – повторити другий крок.

– щоб завершити тур додати ребро між останнім визначеного міста і першим.

У загальному випадку трудомісткість рішення задачі дорівнює O(n^6). Нижня межа вартості оптимального маршруту на 10% вище нижньої межі Хелд-Карп [15].

Опис жадібного алгоритму (Greedy)

Щоб вирішити TSP з використанням жадібний алгоритм, досліджуємо всі ребра, що виходять з міста-вузла, і вибираємо n найкоротших дуг. Якщо ті n найкоротших дуг формують гамільтонів цикл, тоді знайдено оптимальне рішення [11].

Трудомісткість рішення задачі жадібним алгоритмом дорівнює O (n^6).

Нижня межа вартості оптимального маршруту вище нижньої межі Хелд-Карп на 15-20%.

Опис алгоритму Керніган - Ліна (Lin-Kernighan)

Алгоритм Керніган - Ліна вважається одним із найбільш ефективних методів пошуку оптимальних або майже оптимальних рішень задачі комівояжера. Однак розробка і реалізація алгоритму проста лише на перший погляд, так як алгоритм складається з множини кроків, більшість з яких сильно впливає на роботу алгоритму [16]. Створення алгоритму Керніган було натхненне припущенням, що статичне *K* в оптимальному методі не дає найкраще рішення.

З'явилася ідея використовувати різні стадії оптимального методу у виконанні евристичного алгоритму. На практиці було показано, що практично неможливо заздалегідь передбачити яке *K* слід використовувати, щоб досягти

кращого компромісу між трудомісткістю і якістю рішення. Лін і Керниган прибрали цей недолік, ввівши оптимальну змінну, таким чином значення K змінюється під час виконання алгоритму. Трудомісткість при цьому дорівнює O ($n^{6.6}$) [16].

Опис алгоритму пошуку з заборонами (TabuSearch)

Головна проблема алгоритму найближчого сусіда полягає в частому застряванні в точці локального оптимуму. Даний метод дозволяє переходити від одного локального оптимуму до іншого в пошуку глобального оптимуму, після переходу ребро потрапляє в список заборон і повторно не використовується, крім випадків, коли воно може поліпшити побудований оптимальний шлях. На практичному рівні заборонений набір зберігається як комбінація раніше відвідуваних кроків, який дозволяє побудувати подальший шлях щодо поточного рішення і сусідніх вузлів.

Головним недоліком цього методу є його час виконання, трудомісткість алгоритму оцінюється як O ($n!$) [17].

Опис мурашиного алгоритму (Ant Colony Optimization)

Мурашиний алгоритм – ефективний поліноміальних алгоритм, натхненний поведінкою справжніх мурах. Вперше його принципи були описані в 1991 Марко Дориго. Мурашкам властиво співпрацювати в пошуках харчових ресурсів, тому вони залишають слід хімічної речовини, феромонів, на їх шляху від гнізда до джерела їжі [19]. Цей тип невербальної комунікації називають стігмергія – стимуляція, заснована на досвіді попередніх мурах і спрямована на підвищення продуктивності [18].

Для вирішення завдання комівояжера як правило використовують близько 20 мурах. Їх розміщують у випадкові міста і відправляють в інші міста. Їм не дозволяють двічі відвідувати один і той же місто, тільки якщо вони не

завершують маршрут. Той мураха, який вибрав найкоротший тур, буде залишати слід феромонів обернено пропорційний довжині маршруту.

Цей слід феромонів буде зчитуватися наступним мурахою при виборі міста, і з великою ймовірність він піде тим же шляхом, ще сильніше зміцнивши слід. Цей процес буде багаторазово повторений поки не буде знайдено маршрут, досить короткий, щоб бути оптимальним.

Серед недоліків алгоритму хочеться виділити, що перше отримане рішення може виявитися одним з найгірших в плані оптимізації, однак при повторному рішенні метод видає досить точний результат [19].

Опис алгоритму нижня межа Хелд-Карпа (The Held-Karp Lower Bound)

Найпоширеніший спосіб виміряти ефективність евристичного алгоритму для вирішення TSP – це порівняти результати з нижньою границею методу Хелд-Карпа. Ця нижня межа є рішенням TSP, знайденим за поліноміальний час за допомогою симплекс-методу. Нижня межа Хелд-Карпа приблизно на 0.8% нижче оптимальної тривалості туру. У той же час вона гарантовано не перевищує оптимальний час більш ніж на 2/3 [14].

На даний час алгоритм Крістофідеса вважається самим ефективним методом для вирішення завдання комівояжера на загальних метричних просторах, хоча відомі кращі наближення для окремих випадків. Також добре в тестах себе показали алгоритм Керніган-Ліна і жадібний евристичний алгоритм.

1.4 Постановка задачі дослідження

Система призначена для створення бази із рекреаційних зон і побудови на її основі маршруту, який проходитиме по місцям з найвищим рейтингом, заданими видами відпочинку і найменшою відстанню між собою.

Основні цілі системи:

– економія часу користувача при виборі способу проведення свого дозвілля;

– обирати оптимальні маршрути до рекреаційних зон;

– зробити роботу менеджерів (адміністраторів) закладів простішою, завдяки зручним інструментам редагування і добавлення заходів у закладах;

– збільшити обсяг відвідувачів рекреаційних зон.

Для реалізації поставлених цілей система має виконувати наступні функції для користувача:

– додавання, редагування та видалення рекреаційних зон і заходів у них;

– визначення відстані між рекреаційними зонами;

– визначення рейтингу рекреаційними зонами та видів відпочинку в них;

– побудова оптимального маршруту;

– перегляд списку рекреаційних зон та доступних в них видів відпочинку;

– оцінка рекреаційної зони;

– надання певних прав доступу іншим користувачам;

– надання докладної інформації для потенційного клієнта про рекреаційні зону та її можливості по видах і способах відпочинку;

– залишення відгуку про рекреаційну зону.

Система побудови оптимального маршруту по рекреаційних зонах у заданій локації може використовуватися на комп'ютерах і ноутбуках з операційною системою Windows 10 - 11, дистрибутивах Linux і Mac OS. Також адаптивна версія системи працює на смартфонах і планшетах на платформі Android або iOS. Сайт запускається і правильно працює в браузерах Google Chrome, Mozilla Firefox, Opera різних версій.

Основні характеристики системи:

– максимальний час завантаження сторінки 2 с;

– середній час роботи алгоритму 3 с;

– час аутентифікації і входу в систему становить 1 с;

– максимально можлива кількість збереженої в базі даних інформації 4 ГБ;

1.5 Аналіз наявних аналогів

На сьогодні в інтернеті є досить мало сайтів, які надають такі ж можливості. Переваги та недоліки наявних аналогів представлені у таблиці 1.3.

Таблиця 1.3

Переваги та недоліки наявних аналогів

Назва	Переваги	Недоліки
OutdoorsGPS [19]	Сервіс дозволяє вибрати з уже побудованих маршрутів. Є можливість переглянути довжину маршруту.	Сервіс розрахований в основному тільки на велосипедистів і альпіністів.
Strava[20]	Сервіс зберігає пройдені маршрути і дає можливість оцінити їх. Можливість аналізу пройдених маршрутів і рекомендація нових, відповідно до результатів аналізу.	Відсутність рейтингу кожного з відвіданих місць.
Plot a Route[21]	Дозволяє користувачу самому будувати маршрути на карті і зберігати їх	Не надає можливості автоматичної побудови маршруту, оцінки відвіданих закладів і залишення свого відгуку.
Google Maps[22]	Зручна карта і можливість прокладати маршрут, наявність контактних даних закладу, а також можливість залишати і читати відгуки користувачів.	Відсутня можливість оцінити заклад. Також прокладання маршруту не бере до уваги рейтинг закладу. Немає можливості добавляти і переглядати заходи. За детальними даними необхідно звертатися на офіційний сайт закладу.
Speedy Route	Зрозумілий інтерфейс; можна розрахувати	Сервіс налаштовано виключно для автомобілістів;

Назва	Переваги	Недоліки
	найбільш ефективний за кількістю витрат палива маршрут між декількома локаціями	труднощі при пошуку україномовних адрес; мінімальна кількість пунктів маршруту рівна 5; платна підписка
CargoApps	Можна оптимізувати маршрут з закріпленими початковою та фінішною або тільки початковою точками; можна обирати принцип оптимізації – за довжиною шляху або вартістю проїзду	Безкоштовно користуватись можна тільки 14 днів; труднощі при пошуку україномовних адрес; сервіс орієнтований на перевезення вантажів

Загалом функціонал описаних сайтів і систем є досить обмеженим і орієнтований на певну категорію людей і задач.

Головна перевага даної розробки перед аналогами – це побудова маршруту, залежного від відстані рекреаційними зонами, їх рейтингу та наявними видами відпочинку. Такий алгоритм дозволяє побудувати найкращий маршрут і забезпечує максимальну ефективність потраченого користувачем часу.

Іншою перевагою є можливість залишати відгуки про рекреаційні зони, наявні в них заклади, оцінювати їх, а також отримувати інформацію про заходи, які у них відбуваються.

Для того щоб система була конкурентоспроможною необхідно визначити вимоги до її функціоналу та інтерфейсу і реалізувати їх.

1.6 Вимоги до функціоналу

Система побудови оптимального маршруту повинна реалізовувати система авторизації для забезпечення необхідного рівня безпеки при

використанні застосунку. Персональна інформація користувача вказується при реєстрації у застосунку і доступна для читання тільки користувачу. Шляхом введення авторизації забезпечується обмеження доступу до конфіденційної інформації користувачів.

Дана система розрахована як на користувачів, які бажають отримати маршрут по рекреаційних зонах, розважальних заходах, які там можуть відбуватися, так і на менеджерів закладів, які добавлятимуть і редагуватимуть заходи у закладах, цим самим рекламуватимуть свої заклади. Тому внутрішня логіка системи підтримує роботу з різними правами доступу до даних. Завдяки такому підходу є можливість надавати різний функціонал для користувачів і менеджерів, що в свою чергу ще більше захищає дані системи.

Більш детальні вимоги по функціоналу наведено в табл. 1.4.

Таблиця 1.4.

Вимоги до функціоналу застосунку

Функція	**Вимоги**
Реєстрація	Надати користувачу можливість зареєструватися в системі, вказавши свій email, повне ім’я та пароль. Після цього для користувача створюється обліковий запис у системі.
Аутентифікація	Надати користувачу можливість аутентифікуватись у системі, використовуючи вже створений профіль. Аутентифікація – це процес перевірки відповідності введеного логіну і паролю до логіну і паролю, що збережені у базі даних. Якщо така пара є в БД системи, то користувачу надається доступ до внутрішніх даних та сервісів.
Редагування профілю	Додати функціонал для редагування персональної інформації користувача. Змінити можна лише певні дані профілю, відповідно наданій користувачу інформації.

Продовження таблиці 1.4

Надання користувачу прав менеджера	Забезпечити можливість адміністратору надати конкретному користувачу права менеджера.
Перегляд сторінки профілю	Надати можливість переглядати інформацію у своєму профілі
Перегляд списку рекреаційних зон, можливостей відпочинку, розважальних закладів, закладів харчування	Добавити можливість переглянути усі рекреаційні зони та заклади, які зареєстровані у системі, з основною інформацією про них. Дана функція повинна підтримувати пагінацію для розбиття масиву закладів на сторінки. Пагінація – це посторінковий вивід інформації, тобто показ обмеженої частини інформації на одній web-сторінці.
Добавлення рекреаційної зони, закладу	Реалізувати можливість створення нової рекреаційної зони, нового закладу із заповненням детальної інформації про них. Така функція повинна бути тільки на рівні менеджера чи адміністратора.
Перегляд конкретної рекреаційної зони, закладу	Добавити можливість перегляду конкретної рекреаційної зони, закладу з детальнішою інформацією і зі списком видів (можливостей) відпочинку, подій, які відбуватимуться у рекреаційній зоні (закладі) найближчим часом.
	Також на сторінці рекреаційної зони (закладу) повинен бути список відгуків користувачів, які відвідали дане місце (заклад), а також повинен відображатись середній рейтинг рекреаційної зони (закладу).
Редагування інформації про існуючі рекреаційні зони (заклади)	Надати змогу змінювати інформацію про уже введені в систему рекреаційні зони чи заклади. Така функція повинна бути доступна тільки для тих менеджерів, що прив’язані до рекреаційної зони (закладу), який необхідно редагувати, а також для адміністраторів.
Видалення рекреаційних зон (закладів)	Надати змогу видаляти інформацію про конкретні рекреаційні зони (заклади). Така функція повинна бути доступна тільки для тих менеджерів, що прив’язані до рекреаційної зони (закладу), яку необхідно редагувати, а також для адміністраторів.

Продовження таблиці 1.4

Оцінка якості рекреаційної зони (закладу)	Розробити функцію оцінки якості відвіданої рекреаційної зони (закладу). На основі цих оцінок має формуватись загальний рейтинг рекреаційної зони (закладу). Дана функція повинна бути доступною тільки для звичайних користувачів.
Залишення відгуку про рекреаційну зону (заклад)	Добавити можливість залишення відгуку на сайті про відвідану рекреаційну зону (заклад). Дана функція повинна бути доступною тільки для звичайних користувачів.
Створення події у рекреаційній зоні (закладі)	Реалізувати можливість добавлення нової події з її детальним описом і датою. Така функція повинна бути тільки у менеджерів, які прив'язані до рекреаційної зони (закладу), в якій подія створюється.
Редагування та видалення подій	Додати функцію редагування та видалення уже створених подій. Така можливість має бути тільки у менеджерів, які прив'язані до рекреаційної зони (закладу), в якій була створена конкретна подія.
Побудова маршруту із рекреаційних зон (закладів чи подій)	Надати користувачу сервісу можливість отримати згенерований маршрут по рекреаційних зонах (закладах, подіях) при заданій бажаній локації, її радіусі і кількості місць, які хочеться відвідати.
Вихід з акаунта	Добавити можливість виходу з акаунта.
Забезпечення коректності вводу всіх даних	Добавити перевірку коректності всіх введених даних, та, при помилках під час введення даних, повідомляти про це користувача.
Кешування інформації	Забезпечити можливість роботи в офлайн режимі шляхом кешування змін з синхронізацією в майбутньому.

1.7 Вимоги до інтерфейсу

Оскільки якість процесу інтерактивної взаємодії користувача із системою (швидкість, зручність, низький рівень втоми) пов'язана з такими психологічними характеристиками людини як короткострокова та середньострокова пам'ять, час

реакції, можливості сприйняття візуальної інформації, то при розробці інтерфейсу необхідно пам’ятати, що [41]:

– інтерфейс – сама важлива частина веб додатку з точки зору його реклами з метою продажу і з точки зору безпосереднього користувача системи, який може працювати з нею по декілька годин поспіль;

– інтерфейс впливає на характер рішень, які приймає ОПР, він може прискорювати час прийняття рішення та покращувати або погіршувати їх якість;

– який саме конкретний тип інтерфейсу можна створити за допомогою вибраних інструментальних засобів і які принципові можливості може надати інструментальна система.

Основними властивостями, яким повинні задовольняти інтерфейси, є такі [42]:

1) Адаптованість означає, що інтерфейс повинен бути:

– сумісним з потребами та можливостями користувача;

– забезпечувати простоту переходу від виконання однієї функції до іншої;

– забезпечувати користувача на високому рівні вказівками стосовно його можливих дій, а також генерувати належний зворотний зв’язок на його запити;

– надавати користувачу можливість відчувати себе повноправним керівником ситуації при розв’язанні всіх типів задач, тобто, забезпечувати його всією необхідною інформацією; користувач повинен бути впевненим, що він сам розв’язує поставлену задачу;

– забезпечувати користувача різними, взаємно доповнюючими формами представлення результатів в залежності від типу запиту або від характеру отриманого рішення;

– враховувати особливості користувачів різних рівнів;

2) Достатність інтерфейса означає:

– допустимі запити користувача повинні бути чіткими і однозначними для користувачів всіх рівнів, а також для прикладних задач всіх типів;

– реакція системи на всі типи запитів також повинна бути однозначною і зрозумілою і, по можливості, простою.

3) Дружність інтерфейсу Це максимальна простота його використання і готовність в повній мірі задовольнити запити користувача при розв’язанні визначеного класу задач.

4) Гнучкість інтерфейсу Гнучкість інтерфейсу – це можливість його адаптування до розв’язання конкретної задачі. Якщо розв’язувана задача дуже складна, то інтерфейс повинен полегшувати формулювання запитів і видавати результати у формі, яка легко і швидко сприймається користувачем. Тобто інтерфейс повинен буди максимально простим навіть у випадку, коли розв’язується дуже складна задача.

При цьому простота означає наступне:

– інтерфейс не повинен бути перевантажений деталями щодо представлення розв’язку поставленої задачі – користувач може не охопити всіх подробиць (і в цьому, як правило, немає потреби) – тобто нічого зайвого, крім того, що необхідно для розуміння результату;

– він не повинен містити зайвих декоративних деталей, які відволікають від головної задачі;

– інтерфейс повинен бути консистентним, тобто, ґрунтуватись на використанні відомих, загальноприйнятих методів і засобів представлення інформації;

– в ідеалі процес взаємодії користувача з системою не повинен представляти ніяких труднощів.

Основною вимогою до дизайну є те, що він повинен бути єдиним, незалежно від платформи. Користувачі відвідують сайт з різних пристроїв: стаціонарного комп’ютера, ноутбука, планшета чи телефона. Важливою частиною роботи над дизайном є забезпечення кожному користувачу однакової версії сайту, незалежно від пристрою. Тобто дизайн повинен бути «адаптивним», що забезпечить коректне відображення графічних елементів на будь-якому з дисплеїв.

Також важливою вимогою є простота і зрозумілість навігації. Навігація – це ключове поняття в зручності користувацького інтерфейсу. Саме тому у

кожного сайту повинна бути найпростіша з усіх можливих структура, навігація має бути очевидною і однаковою для схожих за сенсом сторінок. Проектувати навігацію потрібно так, щоб користувач завжди розумів де він, куди йому необхідно і як йому потрапити туди за найменшу кількість переходів.

1.8 Висновки

Отже, основним завданням роботи є створення конкурентоспроможної системи для побудови оптимальних маршрутів відвідування рекреаційних зон в межах певної території.

Ця система повинна бути зручною в використанні, доступною як для користувачів, так і для адміністраторів, а також як на екрані комп’ютеру, так і не екрані телефону.

Для того щоб система могла працювати, необхідно імплементувати ефективний алгоритм вирішення задачі комівояжера.

2 РОЗРОБКА ДІАГРАМ ДЛЯ СИСТЕМИ ТА БАЗИ ДАНИХ

2.1 Діаграма класів

Діаграма класів – це діаграма, яка демонструє класи системи, їх атрибути, методи і взаємозв'язки між ними. На діаграмі класи представлені у рамках, які містять три компоненти: у верхній частині написано ім'я класу, посередині знаходяться поля (атрибути) класу, а нижня частина містить методи класу [27].

Основні класи моделі системи наступні:

а) User – містить основну інформацію про користувача, а саме ім'я, прізвище, адресу електронної пошти, роль користувача, пароль ідентифікаційний номер. Також клас містить список оцінок і відгуків конкретного користувача.

б) Venue – містить інформацію про рекреаційну зону (відпочинковий заклад, розташований на її території), її назву, розташування, список подій, що відбуваються у на її території, список оцінок і відгуків про місце (події), а також має метод, який повертає середнє значення рейтингу рекреаційної зони.

в) Event – описує подію у рекреаційній зоні чи закладі і містить посилання на иісце проведення, а також назву, опис і дати початку та закінчення.

г) Rating – містить рейтинг рекреаційної зони чи закладу, посилання власне на місце (заклад) і на користувача, що поставив оцінку

д) Review – містить відгук про рекреаційну зону (заклад), посилання на заклад і на користувача, що залишив відгук.

е) RatingId – з'єднує між собою сутності Rating, Venue та User.

є) Location – широта та довжина місця знаходження рекреаційної зони

ж) Admin – розширює клас User, представляю користувача системи який матиме доступ до CRUD операцій над Venue.

Структурна схема класів компонентів системи зображена на рис. 2.1.

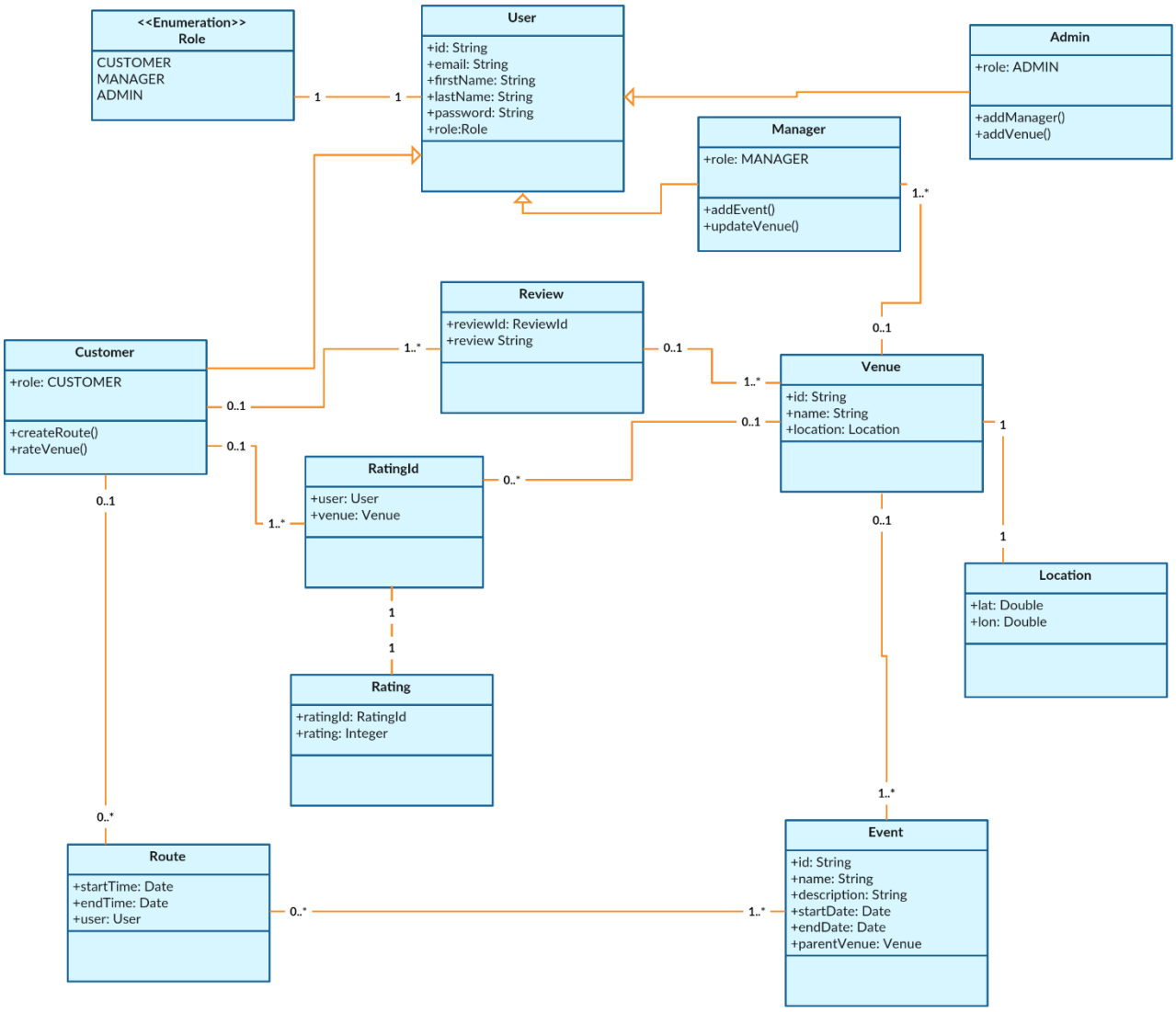

Рис. 2.1 Діаграма класів

3.2 Діаграма використання

Діаграма використання відображає відношення між акторами і прецедентами, вона описує що робить система в зовнішньому світі. На діаграмі використання застосовується 2 типу основних сутностей: варіанти використання (View Venues, Login та т.д.) і дійові особи (Customer, Manager), між якими встановлюються зв'язки [25].

Структурна схема використання представлена на рисунку 3.2. На ній представлені можливості використання системи користувачами з різними ролями, а саме адміністратором, менеджером і звичайним користувачем.

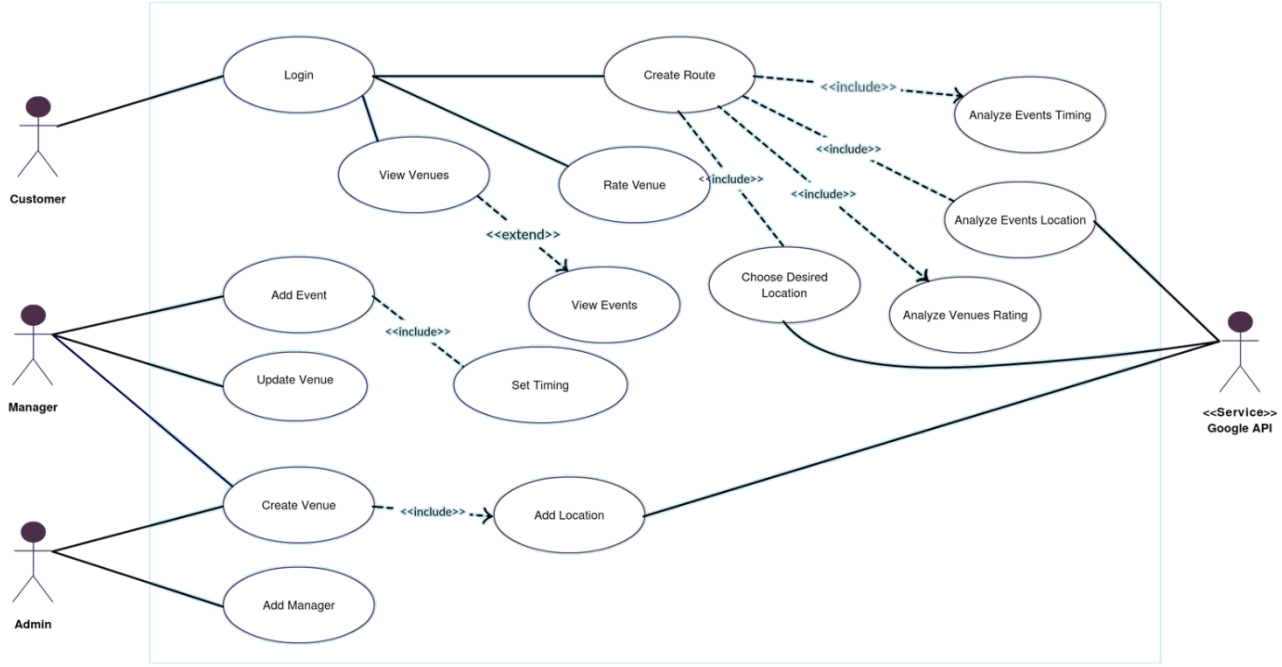

Рис. 2.2 Діаграма використання

2.3 Діаграма послідовності побудови оптимального маршруту

Діаграма послідовності – це діаграма, на якій для певного набору об’єктів на часовій осі показаний життєвий цикл певного об’єкту і взаємодія акторів у рамках визначеного прецеденту [24].

Діаграма послідовності побудови маршруту представлена на рисунку 3.3. На ній зображено процес передачі даних та виклик функцій при генерації маршруту. На ній видно, що процес починається коли користувач, після заповнення даних про локацію відпочинку і кількість місць відпочинку, які він хоче відвідати, натискає на кнопку побудови маршруту. Після цього виконується POST запит на сервер, де описані дані містяться у тілі запиту. Цей запит обробляється контролером на сервері, а він в свою чергу робить запит до сервісу, який відповідає за побудову шляху. Далі цей сервіс отримує з бази даних значення всіх рекреаційних зон, закладів чи подій, які в них відбуваються, які знаходяться в заданій локації, виконує алгоритм пошуку найкращого шляху і повертає отримані результати контролеру, який повертає їх на сторону клієнта, де ця інформація відображається на карті у вигляді шляху.

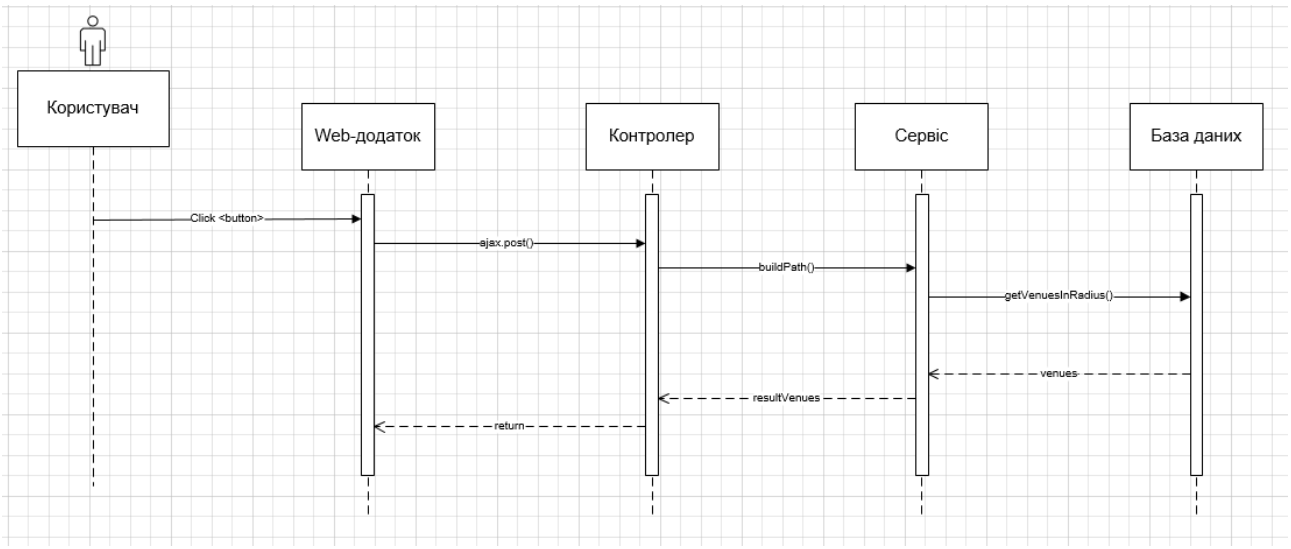

Рис. 2.3 Діаграма послідовності побудови маршруту

2.4 Діаграма послідовності створення місця відпочинку

Діаграма послідовності створення місця відпочинку представлена на рисунку 3.4. На ній представлено процес передачі даних та виклик функцій при створенні нового місця відпочинку (рекреаційної зони, закладу відпочинку).

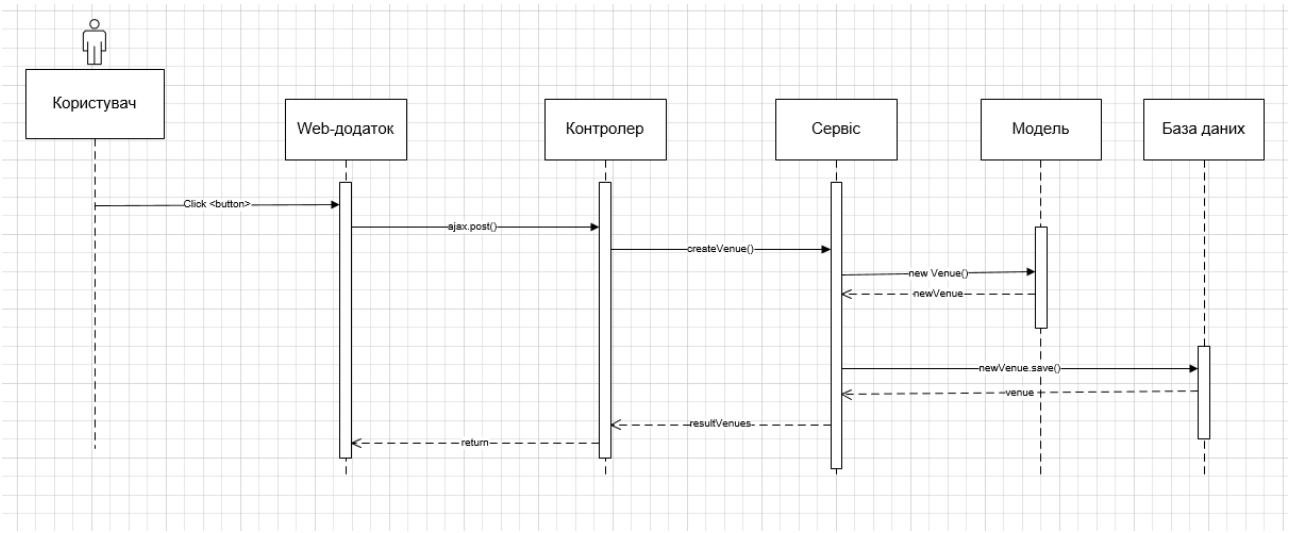

Рис. 2.4 Діаграма послідовності створення місця відпочинку

На ній видно, що процес починається коли менеджер заповнює інформацію про місце відпочинку та натискає на кнопку «Create». Після цього виконується

POST запит на сервер, де дані про заклад містяться у тілі запиту. Цей запит обробляється контролером на сервері, потім керування переходить до сервісу створення місць відпочинку. Потім цей сервіс звертається до моделі і в результаті отримує сутність нового місця відпочинку, яку після цього відправляє базі даних на збереження. У разі успішного виконання збереження, база даних передає назад збережене місце відпочинку.

2.5 Опис бази даних

Для роботи з базами даних було обрано систему керування базами даних MySQL. Для даної магістерської роботи структура бази даних зображена на рис. 2.5.

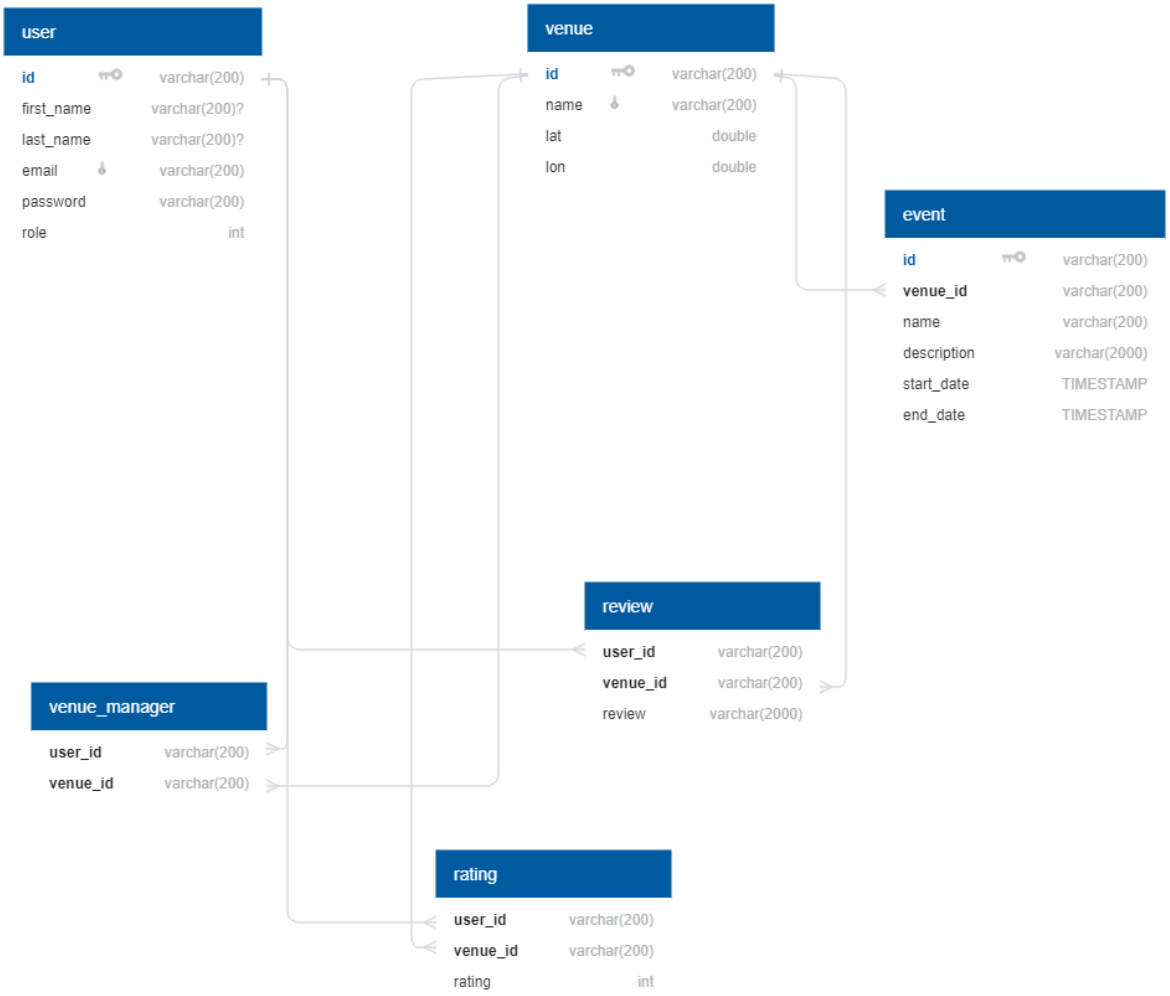

Рис. 2.5 Схема бази даних

MySQL СУБД є хорошим рішенням для малих та середніх застосунків. Завдяки вбудованому механізму багатопоточності у MySQL досить висока швидкодія, також ця СУБД є надійною і безпечною, так як є вбудовані функції для шифрування даних. Також MySQL є досить простою у використанні і у неї багатий функціонал. MySQL легко працює з великими об’ємами даних і легко масштабується, а спрощення деяких стандартів дозволяє MySQL значно збільшити продуктивність [29].

Далі у таблиці 2.1 наведено опис таблиць розробленої бази даних.

Таблиця 2.1.

Опис таблиць бази даних

Назва таблиці	Назва стовпця	Тип даних	Детальна інформація
user	Id	varchar(200)	Ідентифікаційний номер користувача
	first_name	varchar(200)	Ім’я користувача
	last_name	varchar(200)	Прізвище користувача
	Email	varchar(200)	Електронна пошта користувача
	Password	varchar(200)	Пароль користувача
	role	int	Роль користувача
venue	id	varchar(200)	Ідентифікаційний номер рекреаційної зони (закладу)
	lat	double	Значення широти, на якій знаходиться рекреаційна зона (заклад)
	lon	double	Значення довготи, на якій знаходиться рекреаційна зона (заклад)

Продовження таблиці 2.1

event	id	varchar(200)	Ідентифікаційний номер події
	venue_id	varchar(200)	Ідентифікаційний номер закладу, де проводиться подія
	name	varchar(200)	Назва події
	description	varchar(2000)	Опис події
	start_date	timestamp	Дата початку події
	end_date	timestamp	Дата закінчення події
venue_manager	venue_id	varchar(200)	Ідентифікаційний номер рекреаційної зони (закладу)
	user_id	varchar(200)	Ідентифікаційний номер менеджера (адміністратора), який прив’язаний до рекреаційної зони (закладу)
review	venue_id	varchar(200)	Ідентифікаційний номер рекреаційної зони (закладу)
	user_id	varchar(200)	Ідентифікаційний номер користувача, який залишив відгук
	review	varchar(2000)	Відгук користувача
rating	venue_id	varchar(200)	Ідентифікаційний номер рекреаційної зони (закладу)
	user_id	varchar(200)	Ідентифікаційний номер користувача, який оцінив рекреаційну зону (заклад, подію)

2.6 Робота з базою даних

Для зв’язку моделі з базою даних у магістерській роботі використано ORM Hibernate. Це найпопулярніша реалізація специфікації JPA, яка призначена для вирішення задач об’єктно-реляційного відображення (ORM).

Java Persistence API (JPA) – це специфікація API Java EE, яка надає можливість зберігати в зручному виглядів Java-об’єкти у базі даних.

ORM – це технологія програмування, яка зв’язує бази даних з концепціями об’єктно орієнтованих мов програмування, що створює «віртуальну об’єктну базу даних».

Ціллю Hibernate є звільнення розробника від значного об’єму порівняно низькорівневого програмування при роботу в об’єктно-орієнтованих засобах в реляційній базах даних, такій як MySQL, що використовується у роботі. Можна використовувати Hibernate як в процесі проектування системи класів і таблиць, так і для роботи з базою даних, що уже створена. Бібліотека вирішує не тільки задачу зв’язку класів Java з таблицями бази даних, а і надає засоби для автоматичної генерації і оновлення таблиць, побудову запитів і обробки отриманих даних, що значно зменшує час розробки [28]. Також Hibernate забезпечує прозору підтримку цілісності даних для стандартниз Java об’єктів (POJO).

Проектування Java-класів на таблиці бази даних може відбувається з допомогою конфігураційних XML-файлів або Java-анотацій. У роботі використовуються Java-анотації. Також забезпечуються можливості по організації відношень між класами «один-до-багатьох» і «багато-до-багатьох». У додаток до керування зв’язками між об’єктами Hibernate також може керувати рефлексивними відношеннями, де один об’єкт має зв’язок «один-до-багатьох» з іншими екземплярами свого типу даних.

Hibernate підтримує відображення користувацьких типів значень, що дозволяє перевизначати тип по замовчуванню SQL, проектувати тип ENUM на поле бази даних, так ніби вони є звичайними властивостями.

Колекції об’єктів даних зберігаються у вигляді колекцій Java-об’єктів, таких як набір чи список. Hibernate може бути налаштований на «ліниві» завантаження колекцій, а зв’язані об’єкти можуть бути налаштовані на каскадні операції. Це також дозволяє зменшити час розробки і забезпечує цілісність [29].

У роботі використано дві компоненти фреймворку, а саме Hibernate ORM, ядро Hibernate і власний API, і Hibernate Annotations, відображення за допомогою стандартних анотацій JPA.

Приклад зв’язку Java об’єкту User з базою даних за допомогою анотацій Hibernate представлено на рисунку 2.6.

```
@Entity
@Table(name = "user")
public class User implements Serializable {
    @Id
    private String id;

    @Column(length = 100, nullable = false)
    private String email;

    @Column(name = "first_name", length = 100)
    private String firstName;

    @Column(name = "last_name", length = 100)
    private String lastName;

    @Column(length = 100, nullable = false)
    private String password;

    @Column
    private Role role;

    @OneToMany(mappedBy = "user")
    @JsonIgnore
    private List<Rating> ratings = new ArrayList<>();

    @OneToMany(mappedBy = "user")
    @JsonIgnore
    private List<Review> reviews = new ArrayList<>();
```

Рис. 2.6 Приклад налаштування ORM

2.7 Створення і заповнення таблиць

Для створення і заповнення таблиць даними було використано безплатний засіб для міграцій баз даних FlyWay. Його основною перевагою є простота, а також те, що він дозволяє зменшити кількість операцій. FlyWay підтримує базу

даних MySQL, а також легко інтегрується з фреймворком розробки системи Spring.

Засіб FlyWay перед кожним запуском системи перевіряє папку db/migration і виконує неактивні MySQL скрипти з неї. Усі виконані операції також зберігаються у базі даних у спеціально створеній таблиці. На рис. 2.7 зображено список скриптів, які необхідні для коректної роботи системи.

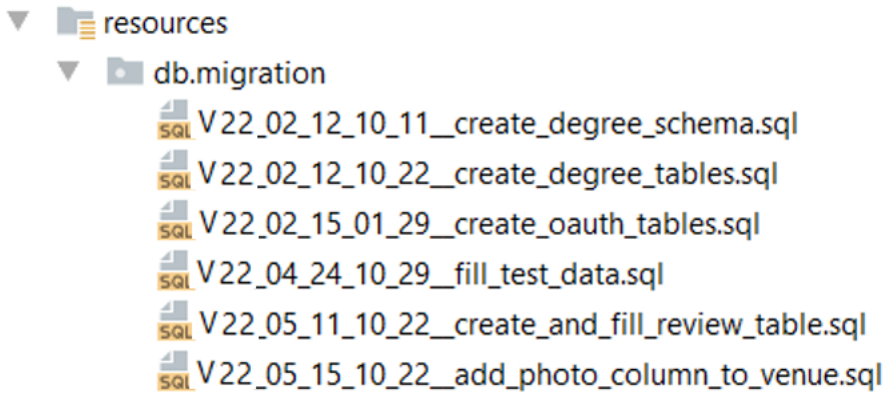

Рис. 2.7 Структура скриптів для міграцій

2.8 Висновки

У розділі було спроектовано систему у вигляді діаграм класів, використання, а також послідовностей побудови оптимального маршруту та створення місць відпочинку.

Також вибрано конкретну СУБД, яка найкраще підходить до поставленої задачі і спроектовано та створено базу даних системи.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ПОБУДОВИ МАРШРУТУ

3.1 Середовище та технологія розробки

З огляду на поставлені завдання, при розробці серверної складової програмного продукту було використано фреймворк Spring і середовище розробки Intellij IDEA. Для інтерактивної системи пошуку місць громадського харчування даний інструмент ідеально підходить, оскільки поєднує просте написання серверної складової на мові Java, створення скриптів на JavaScript і верстку сторінок сайту на HTML і CSS.

Spring Framework – це універсальний фреймворк з відкритим вихідним кодом для Java-платформи. Всі вихідні файли фреймворку доступні на GitHub. Його було обрано, тому що Spring забезпечує вирішення багатьох задач, з якими зіштовхується розробник програмного забезпечення. Цей фреймфорк пропонує послідовну модель і робить її застосування можливим для більшості типів застосунків [30].

Завдяки модульності фреймворка всі необхідні компоненти веб-застосунки можуть завантажуватися як окремі модулі через Maven, фреймворк для автоматизації збірки проектів на основі опису їх структури у файлі. Maven забезпечує декларативну збірку проекту, тобто у файлах міститься його специфікація, а не окремі команди виконання. Такий підхід спрощує компонування проекту і підтягування необхідних модулів.

Центральною складовою Spring є контейнер інверсії управління, який надає засоби конфігурації і керування об’єктами Java за допомогою рефлексії. Цей контейнер відповідає за керування життєвим циклом об’єкту, тобто за створення об’єктів, виклик методів ініціалізації і конфігурування об’єктів шляхом зв’язування їх між собою. Крім даного контейнера, для роботи було використано декілька модулів Spring Framework:

а) модуль доступу до даних – надає свій шар доступу до бази даних і підтримує усі популярні системи керування базами даних. Цей модуль забезпечує керування ресурсами, тобто автоматичне отримання і звільнення ресурсів бази даних. Також він дає можливість переведення виключень при доступі до даних у виключення Spring-а, забезпечує прозорість транзакцій в операціях з даними і отримання об'єктів бази даних з пулу з'єднань[31];

б) модуль MVC – каркас, який у основі має HTTP і сервлети і який надає багато можливостей для розширення і налаштування. У ньому визначені стратегічні інтерфейси для усіх функцій сучасної запито-орієнтованої системи. Кожен інтерфейс є простим і ясним, так щоб користувачам було просто його заново реалізовувати;

в) модуль аутентифікації і авторизації – інструментарій процесів аутентифікації і авторизації, який можна легко конфігурувати і який підтримує багато популярних протоколів і інструментів [32].

г) модуль тестування – каркас, який підтримує класи для написання модульних та інтеграційний тестів.

Для пришвидшення і спрощення розгортання застосунку було використано Spring Boot. Він дозволяє створювати самостійні застосунки, які зразу готові до використання, має вбудований web-сервер Tomcat, що позбавляє необхідності деплоїти WAR файли. Також Spring Boot автоматично налаштовує Spring там, де це можливо і не потребує XML конфігураціє, що спрощує роботу з ним.

Інтегроване середовище розробки Intellij IDEA дуже добре підходить до розробки web-застосунків, так як воно підтримує усі популярні фреймворки і платформи, такі як Java EE, Spring Framework, AngularJS [33]. Також у ньому наявне розумне автодоповнення, інструменти для аналізу якості коду, зручна навігація, розширені рефакторинги і форматування для Java, HTML, CSS, що значно пришвидшує і спрощує розробку програмного забезпечення. У Intellij IDEA є інтеграція з web-серверами, зокрема з сервером Tomcat, а також є

інструменти для роботи з базами даних і інструменти для запуску тестів та аналізу покриття коду.

При розробці в Intellij IDEA проекти застосунків мають вбудовану підтримку з такими популярними інструментами, як Bower, Grunt, Gulp, які дозволяють керувати скриптами JavaScript і стилями CSS, автоматизувати і оптимізувати процес веб-розробки.

Бібліотекою для журналювання Java програми було обрано Log4J, яка є частиною загального проекту «Apache Logging Project» [34]. Ця бібліотека дозволяє робити логи на кількох рівнях, залежно від їх строгості:

– рівень «OFF» – максимально можливий ранг, який призначений для виключення функції логування;

– рівень «FATAL» – призначений для тяжких помилок, які зупиняються роботу програми;

– рівень «ERROR» – призначений для усіх інших помилок виконання і неочікуваних умов;

– рівень «WARN» – призначений для відображення попереджень у випадках використання застарілих API, неправильне використання API, ситуації під час виконання програми, які є небажаними, але необов’язково неправильними;

– рівень «INFO» – призначений для відображення різної корисної інформації під час виконання програми.

– рівень «DEBUG» – призначений для відображення деталізованої інформації про потік виконання.

– рівень «TRACE» – призначений для відображення найбільш деталізованої інформації

Також Log4J дозволяє створювати свої рівні логування і налаштовувати їх, а також потік виводу логів у файлі конфігурацій.

Як фасад для протоколювання обрано Slf4J. Це бібліотека для протоколювання, яка ставить за мету надати максимально простий, але в той же час потужний фасад для різних систем протоколювання на Java. Slf4J надає

простий загальний інтерфейс для систем протоколювання, який не залежить від конкретної реалізації, зокрема Slf4J прозоро інтегрується з обраною реалізацією Log4J. Крім того вона надає можливість інтегрувати компоненти, що залежать від системи протоколювання Log4J шляхом підставлення реалізації, яка направляє логи цих систем в Slf4J.

Системою керування версіями було обрано Git, а веб-сервісом для хостингу проектів та їх спільної розробки обрано GitHub.

3.2 Опис архітектури системи

Для підвищення продуктивності системи і спрощення архітектури було реалізовано REST, архітектурний стиль взаємодії компонент розподіленого застосунку в мережі. REST являє собою узгоджений набір обмежень, які враховуються при проектуванні розподіленої системи. Для даної системи такий підхід забезпечує простоту уніфікованого інтерфейсу, відкритість компонент до можливих змін для задоволення потреб, що змінюються, прозорість зв’язків між компонентами системи для сервісних служб, надійність, яка виражається у стійкості системи до відмов на рівні системи у окремих компонентах, з’єднаннях чи даних.

Проте реалізація такого підходу накладає на розробника певні обмеження. По-перше, архітектура системи повинна відповідати моделі «клієнт-сервер». Розділення потреби інтерфейсу клієнта від потреб сервера, що зберігає дані, підвищує переносимість коду клієнтського інтерфейсу на інші платформи, а спрощення серверної складової покращує масштабованість. Також це розподілення дозволяє частинам розвиватись незалежно один від одного.

По-друге, необхідно щоб в період між запитами клієнта ніякої інформації про стан клієнта на сервері не зберігалось, тобто всі запроси клієнта повинні бути побудовані так, що щоб сервер отримав всю необхідну інформацію для виконання запиту. Стан сесії при цьому зберігається на стороні клієнта.

Інформація про стан сесії може бути передана сервером якомусь іншому сервісу для підтримки стійкого стану системи.

Фундаментальною умовою дизайну RESTful сервісів є уніфікованість інтерфейсу. Це дозволяє кожному з сервісів розвиватись незалежно. При такому відходу клієнти можуть змінювати стан системи тільки через дії, які динамічно визначені на сервері [35] Також всі ресурси повинні ідентифікуватись в запитах і бути концептуально відділеними від відображень, які повертаються клієнту.

Також RESTful системи повинні підтримувати наявність проміжних серверів, при цьому для клієнта не має бути різниці між проміжним вузлом і власне сервером. Використання проміжних серверів дозволяє підвищити масштабованість за рахунок балансування навантаження і розподіленого кешування.

Додатковою вимогою є можливість клієнтів і проміжних серверів виконувати кешування відповідей з серверу. При цьому самі відповіді повинні мати явне чи неявне позначення даних, які необхідно закешувати, щоб запобігти отриманню клієнтами застарілих чи неправильних даних. Правильне використовування кешування дозволяє повністю чи частково позбутись деяких взаємодій між клієнтом і сервером, що ще більше підвищує продуктивність системи.

Для керування інформацією сервісу, її передачею найкраще підходить протокол HTTP. Це протокол прикладного рівня, який орієнтований якраз на технологію «клієнт-сервер». Протокол HTTP підтримує багато методів, послідовностей з символів, які вказують на основну операцію над ресурсом, але у даній роботі необхідні тільки методи GET, POST і DELETE. Метод GET дозволяє отримати будь-які дані, ідентифіковані за допомогою URL в запиті ресурсу. Також за допомогою цього методу можна почати певний процес. Саме через цей метод будуються запити про повернення системою певних даних, таких як повернення списку з закладів чи подій.

Метод DELETE створений для передачі на сервер запиту про видалення вказаного ресурсу. У системі він використовується для передачі запитів на видалення закладу чи події.

Метод POST розроблений для передачі на сервер такої інформації, як особисті дані для реєстрації чи входу в систему, інформації про новий заклад чи подію для подальшого їх створення, тобто для передачі великого обсягу інформації. Також метод POST використовується для побудови нового маршруту, для цього у тілі запиту вказується широта і довгота локації, її радіус і кількість закладів, які користувач хоче відвідати.

Дані на сервер передаються через HTML-форми. У них можна вводити текст або вибирати підходящі варіанти зі списку. Дані, записані у форму, відправляються для обробки на сервері. Залежно від введених користувачем даних ця програма може формувати різні web-сторінки, відправляти запити до бази даних.

3.3 Опис основних функцій серверної сторони

Класи і функції, які відповідають за створення користувача і взаємодію з ним зображені у таблиці 3.1.

Таблиця 3.1

Функції керування класом «User»

Клас	Функція	Опис
UserController	createCustomer	Приймає запит з клієнту на створення нового користувача і повертає створеного
UserController	updateToManager	Приймає запит з клієнту про зміни ролі «Користувач» на роль «Менеджер»
UserController	getUser	Приймає запит з клієнту на дані про користувача з вказаним ідентифікаційним номером і повертає ці дані

Продовження таблиці 3.1

UserController	getLoggedInUser	Приймає запит з клієнту на дані про користувача поточної сесії і повертає ці дані
UserController	deleteUser	Приймає запит з клієнту на видалення користувача
UserService	save	Зберігає користувача
UserService	updateToManager	Змінює роль користувача на роль «Менеджер»
UserService	getUser	Повертає користувача по ідентифікаційному номері
UserService	getUserByEmail	Повертає користувача по адресі електронної пошти
UserService	deleteUser	Видаляє користувача

У таблиці 3.2 описано класи і функції, які відповідають за модель «Venue» на серверній стороні.

Таблиця 3.2

Функції керування класом «Venue»

Клас	Функція	Опис
VenueController	createVenue	Приймає запит з клієнту на створення нової рекреаційної зони (закладу) з вказаною інформацією і повертає створену рекреаційну зону (заклад)
VenueController	getVenue	Приймає запит з клієнту на дані про заклад з вказаним ідентифікаційним номером і повертає ці дані
VenueController	getVenuesList	Приймає запит з клієнту про список рекреаційних зон (закладів) з вказаним номером і розміром сторінки і повертає ці дані
VenueController	getVenuesInRadiusList	Приймає запит з клієнту на список рекреаційних зон (закладів) у заданій локації і повертає ці дані

Продовження таблиці 3.2

VenueController	buildPath	Приймає запит з клієнту на побудову оптимального маршруту і повертає побудований маршрут
VenueController	deleteVenue	Приймає запит з клієнту на видалення рекреаційної зони (закладу)
VenueService	save	Зберігає рекреаційну зону (заклад)
VenueService	getVenue	Повертає інформацію про рекреаційну зону (заклад) по ідентифікаційному номері
VenueService	getVenueByName	Повертає інформацію про рекреаційну зону (заклад) по імені
VenueService	getVenues	Повертає список рекреаційних зон (закладів)
VenueService	deleteVenue	Видаляє рекреаційну зону (заклад)
VenueService	getVenuesInRadius	Список рекреаційних зон (закладів) у вказаній локації

Класи і функції, що відповідають за модель «Event» на серверній стороні описано у таблиці 3.3.

Таблиця 3.3

Функції керування класом «Event»

Клас	Функція	Опис
EventController	createEvent	Приймає запит з клієнту на створення нової події з вказаною інформацією і повертає створену подію
EventController	getEvent	Приймає запит з клієнту на дані про подію з вказаним ідентифікаційним номером
EventController	getEvents	Приймає запит з клієнту на дані про події, які відбудуться після поточної дати у вказаній рекреаційні зоні (закладі) і повертає їх

Продовження таблиці 3.3

EventController	deleteEvent	Приймає запит з клієнту на видалення події
EventService	save	Зберігає подію
EventService	getEvent	Повертає інформацію про подію
EventService	getSortedActiveEvents InVenue	Повертає інформацію про активні події у вказаній рекреаційній зоні (закладі)
EventService	deleteEvent	Видаляє подію

У таблиці 3.4 описано класи і функції, які відповідають за модель «Rating» на серверній стороні.

Таблиця 3.4

Функції керування класом «Rating»

Клас	Функція	Опис
RatingController	createRating	Приймає запит з клієнту на виставлення конкретним користувачем оцінки вказаній рекреаційній зоні (закладу)
RatingController	getRating	Приймає запит з клієнту на отримання оцінки конкретного користувача по вказаній рекреаційній зоні (закладу)
RatingService	save	Зберігає оцінку користувача
RatingService	changeRating	Змінює оцінку користувача

Класи і функції, що відповідають за модель «Review» на серверній стороні описано у таблиці 3.5.

Таблиця 3.5

Функції керування класом «Review»

Клас	Функція	Опис
ReviewController	createReview	Приймає запит з клієнту на добавлення конкретним користувачем відгуку)

Продовження таблиці 3.5

ReviewController	getReview	Приймає запит з клієнту на отримання відгуку конкретного користувача вказаній рекреаційній зоні (закладу)
ReviewController	getVenueReviews	Приймає запит з клієнту на отримання усіх відгуків вказаного закладу
ReviewService	save	Зберігає відгук користувача
ReviewService	getReview	Повертає відгук користувача
ReviewService	getVenueReviews	Повертає всі відгуки про вказаний заклад

3.4 Розробка структури роботи

Весь функціонал реалізується з використанням наступної структури, яка зображена на рисунку 3.1.

Resources – це папка, яка використовується для зберігання статичних файлів проекту: зображень, скриптів JavaScript, файлів CSS і інших необхідних файлів.

Config – це пакет з усіма конфігураціями проекту, тобто з основною конфігурацією програми, налаштуваннями доступу до ресурсів і аутентифікації.

Controller – це пакет, у якому зберігаються усі контролери застосунку.

DTO – це пакет з усіма DTO. Data transfer object – це один з шаблонів проектування для передачі даних між підсистемами застосунку.

Exception – пакет з виключеннями, які використовуються у застосунку.

Model – пакет з класами моделі застосунку.

Repository – пакет з усіма репозиторіями, які реалізовують доступ до бази даних.

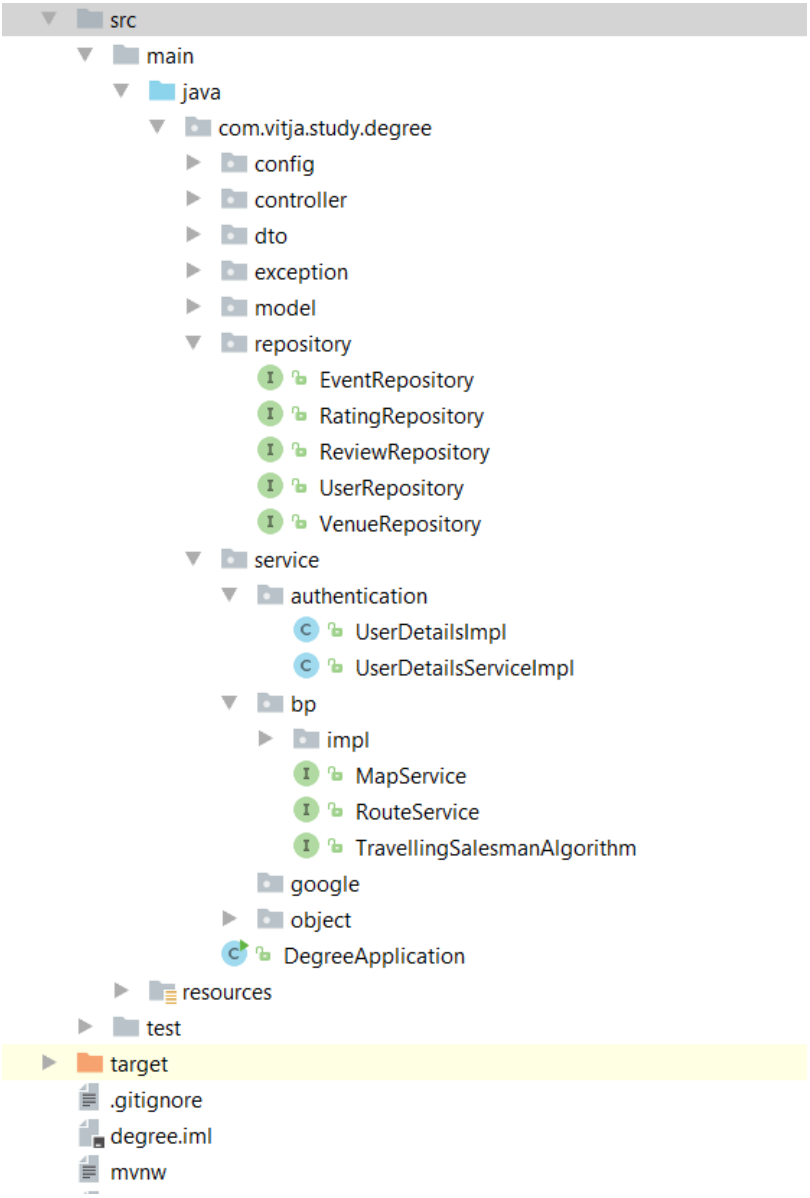

Рис. 3.1 Структура проекту

Service – це набір класів, які реалізовують основний функціонал застосунку, такі як сервіс авторизації, сервіси доступу до бази даних, а також бізнес-логіка.

3.5 Розробка системи аутентифікації та авторизації

Для інтерактивної системи побудови оптимального маршруту по розважальних заходах використано фреймворк Spring Security, який є стандартом для застосунків, які основані на Spring. Завдяки реалізованій системі

авторизації користувач може створити обліковий запис на сайті, пройти аутентифікацію, адміністратор системи може керувати обліковими записами. Протоколом авторизації було обрано Oauth2.0.

Аутентифікація це процедура перевірки автентичності. У випадку з сайтами мається на увазі перевірка справжності користувача за допомогою перевірки введеного ним логіна і пароля з тими, що уже зберігаються в базі даних.

Процедура аутентифікації використовується для обміну інформацією між двома джерелами даних, дані передаються по протоколу HTTPS, який є розширенням протоколу HTTP і який підтримує шифрування, що збільшує безпеку передачі. Завдяки їм можна захистити лінію зв'язку від крадіжки даних або підміни одного з учасників передачі інформації. Дані у протоколі передаються поверх криптографічних протоколів SSL і TLS [36].

Авторизація є наданням одному або групі користувачів певних прав для здійснення конкретних дій у системі. Також відбувається перевірка цих самих прав під час здійснення певних дій. Аутентифікація відрізняється від авторизації тим, що перевіряє легальність користувача шляхом порівняння його даних облікового запису з тими, що зберігаються в базі даних. Якщо перевірка пройдена, то тоді в силу вступає авторизація, яка перевіряє вже легальних користувачів системи на предмет дозволу виконання конкретних дій або відкриття доступу до певного контенту.

У створеній системі використовується класичне управління доступом на основі ролей. Ролі дають можливість визначити правила розмежування доступу для користувачів. Як протокол авторизації використано відкритий протокол Oauth2.0, а саме потік з представленням клієнту паролю. Тобто власник ресурсу надає клієнту логін і пароль, він передає їх серверу, а у відповідь отримує токен доступу до ресурсу. Після цього сторона клієнта передає цей токен при кожному запиті до системи, і на основі цього токену сервер проводить авторизацію користувача і визначає його ролі.

Фреймворк Spring Security підтримує використання протоколу авторизації Oauth2.0 і для його використання потрібно налаштувати сервер ресурсу, який надає користувачу ресурси, відповідно до запиту, і сервер авторизації, який перевіряє відповідність користувача до ресурсів, які він хоче отримати. Також в сервері авторизації вказується куди будуть зберігатись токени доступу до ресурсів і інформація про клієнта [37]. Також протокол Oauth2.0 вимагає реалізації інтерфейсу UserDetailsService, а саме вказання репозиторію і пошук у ньому користувача з відповідним логіном і повернення інформації про цього користувача. У поверненому об'єкті повинні бути username користувача, його пароль, інформація про те чи акаунт не заблокований чи не відключений, а також дозволи, які є у конкретного користувача.

Лістинг конфігурації серверу авторизації зображений на рис. 3.2. На ньому зображено клас AuthorizationServerConfiguration, який розширює клас AuthorizationServerConfigurerAdapter. Spring IoC підтягує уже створені об'єкти класів UserDetailsService, DataSource і AuthenticationManager. Для коректної роботи перевизначаються методи класу AuthorizationServerConfigurerAdapter.

У методі configure(AuthorizationServerSecurityConfigurer oauthServer) визначається доступ користувачів до токенів, метод, який перевіряє валідність токену, а також визначається спосіб шифрування паролів.

Метод configure(ClientDetailsServiceConfigurer clients) відповідає за налаштування бази даних, де буде зберігатись інформація про користувачів. У ньому також потрібно вказати як будуть шифруватись паролі перед збереженням у базі даних, а також вказується власне DataSource.

Метод configure(AuthorizationServerEndpointsConfigurer endpoints) відповідає за налаштування бази даних, де буде зберігатись інформація про токени доступу. У цьому методі також вказується менеджер аутентифікації і сервіс, який відповідає за пошук користувача і визначення його ролей.

У пакеті з контролерами проекту є контролер UserController, який відповідає за створення користувачів в інтерактивній системі, їх модифікації та видалення. Усередині пакету model знаходиться модель User.java, яка

представляє користувача і містить дані про нього. Крім цього, у папці db/migrations описані усі міграції бази даних, її створення і заповнення початковими даними.

```
public class AuthorizationServerConfiguration extends AuthorizationServerConfigurerAdapter {

    @Autowired
    @Qualifier("userDetailsService")
    private UserDetailsService userDetailsService;

    @Autowired
    private DataSource dataSource;

    @Autowired
    @Qualifier("authenticationManagerBean")
    private AuthenticationManager authenticationManager;


    @Override
    public void configure(AuthorizationServerSecurityConfigurer oauthServer) throws Exception {
        oauthServer.passwordEncoder(new BCryptPasswordEncoder())
                .allowFormAuthenticationForClients()
                .tokenKeyAccess("permitAll()")
                .checkTokenAccess("isAuthenticated()")
        ;
    }

    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
        clients.jdbc(dataSource).passwordEncoder(new BCryptPasswordEncoder());
    }

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
        endpoints.tokenStore(tokenStore())
                .authenticationManager(authenticationManager);
        endpoints.userDetailsService(userDetailsService);
    }

    @Bean
    public TokenStore tokenStore() { return new JdbcTokenStore(dataSource); }
```

Рис. 3.2 Лістинг конфігурації серверу авторизації

3.6 Опис роботи алгоритму пошуку

Для побудови оптимального маршруту по рекреаційних (відпочинкових) зонах необхідно знайти шлях, який включатиме у себе бажану кількість місць відпочинку але при цьому буде найкоротшим з усіх можливих шляхів.

Задача пошуку оптимального шляху по місцях відпочинку у певній області дуже подібна на одну з самих відомих задач комбінаторної оптимізації – задачу комівояжера. Вона полягає у пошуку найвигіднішого маршруту, який проходить через вказані міста по одному разу з поверненням у початкове місто. В умовах

задачі вказуються критерій вигідності маршруту (найкоротший, найдешевший і тому подібне) і відповідні матриці відстані, вартості тощо. Алгоритм, реалізований в системі, відрізняється від задачі комівояжера тим, що у ньому непотрібно повертатись до початкової точки, також граф переходів між точками є направленим, а вага шляхів між двома точками відрізняється від напрямку, так як при обчисленні ваги враховується рейтинг місць відпочинку. Також у ньому вказується максимальна кількість місць відпочинку, які користувач бажає відвідати, тобто немає необхідності будувати повний маршрут.

Оптимізаційна постановка як задачі комівояжера, так і задачі, яка поставлена при розробці системи, відноситься до класу NP-важких задач.

Якщо розв'язок задачі можна представити алгоритмом з поліноміальною складністю в залежності від розміру входу, то задачу відносять до класу P, у противному випадку відносять до NP класу [38]. Задачі, які мають поліноміальні по часу алгоритми рішення, можна вирішувати за допомогою комп'ютера значно швидше, ніж шляхом прямого перебору, час якого є експоненціальним.

Проте цю задачу можна вирішити не прямим перебором, а з використанням методу динамічного програмування. Динамічне програмування – це метод вирішення складних задач шляхом їх розбиття на простіші підзадачі. Використання цього методу значно збільшує швидкість виконання алгоритму, при великих кількостях місць відпочинку.

На рис. 3.3 зображено лістинг основної функції класу TravellingSalesmanAlgorithmImpl, в якому знаходиться реалізація модифікованого алгоритму комівояжера [39]. Функція getBestPath(double[][] distanceMatrix, int maxNodes) приймає матрицю відстаней між місцями відпочинку і максимальну кількість місць відпочинку, яку користувач хоче відвідати. У ній відбувається перевірка на валідність отриманих даних, пошук найкоротших шляхів при можливих початкових точках, вибір з них найоптимальнішого і повернення цього шляху.

Пошук найкоротшого шляху при заданому початковому закладі виконується у функції getBestPathForStartNode(int start). Він відбувається у три етапи.

```
@Override
public List<Integer> getBestPath(double[][] distanceMatrix, int maxNodes) {
    possibleRoutes.clear();
    distance = distanceMatrix;
    N = distance.length;
    maxNodesNumber = N < maxNodes ? N : maxNodes;

    if (maxNodesNumber <= 0) throw new IllegalStateException("N cannot be <= 0");
    if (N != distance[0].length) throw new IllegalStateException("Matrix must be square (n x n)");

    if(N == 1 || maxNodesNumber == 1){
        return Collections.singletonList( o: 0);
    }

    if(N == 2 && maxNodesNumber == 2){
        return Arrays.asList(0, 1);
    }

    for(int i = 0; i < N; i++){
        TemporaryPath bestPathForStartNode = getBestPathForStartNode(i);
        possibleRoutes.add(bestPathForStartNode);
    }
    Optional<TemporaryPath> result = possibleRoutes.stream()
            .min(Comparator.comparing(TemporaryPath::getDistance));
    return result.isPresent() ? result.get().getPath() : new ArrayList<>();
}
```

Рис. 3.3 Лістинг основної функції класу TravellingSalesmanAlgorithmImpl

Спочатку ініціалізується матриця memo, в яку будуть записуватись усі знайдені комбінації відстаней між місцями відпочинку. Потім ця матриця заповнюється значеннями відстаней між початковим місцем відпочинку і усіма іншими. Далі програма циклічно виконує пошук найкоротших шляхів між n місцями відпочинку на основі уже знайдених відстаней між $(n - 1)$ кількістю закладів. На кожному кроці циклу інформація про обчислену найкоротшу відстань зберігаються у матрицю memo для того, щоб у наступному кроці можна було на її основі знайти найкоротший шлях для більшої кількості місць відпочинку.

Код реалізації першого етапу зображений на рис. 3.4 з лістингом частини функції getBestPathForStartNode(int start).

Після цього програма проходить по усім значенням шляхів з заданою кількістю місць відпочинку у них і знаходить найменше значення ваги

маршруту, а список місць відпочинку, через які проходить цей маршрут зберігається у змінній tempEndState.

```
List<Integer> possibleEndStates = combinations(maxNodesNumber, N);
int tempEndState = possibleEndStates.get(0);
Double[][] memo = new Double[N][1 << N];

// Add all outgoing edges from the starting node to memo table.
for (int end = 0; end < N; end++) {
    if (end == start) continue;
    memo[end][(1 << start) | (1 << end)] = distance[start][end];
}

for (int r = 3; r <= maxNodesNumber; r++) {
    for (int subset : combinations(r, N)) {
        if (notIn(start, subset)) continue;
        for (int next = 0; next < N; next++) {
            if (next == start || notIn(next, subset)) continue;
            int subsetWithoutNext = subset ^ (1 << next);
            double minDist = Double.POSITIVE_INFINITY;
            for (int end = 0; end < N; end++) {
                if (end == start || end == next || notIn(end, subset)) continue;
                double newDistance = memo[end][subsetWithoutNext] + distance[end][next];
                if (newDistance < minDist) {
                    minDist = newDistance;
                }
            }
            memo[next][subset] = minDist;
        }
    }
}
```

Рис. 3.4 Лістинг реалізації першого етапу алгоритму

Реалізація даного етапу зображена на рис. 3.5.

Останнім етапом є відновлення знайденого найкоротшого шляху. Для цього починаючи з вибраного місця відпочинку переглядаються місця відпочинку (рекреаційні зони, заклади відпочинку, що в них розташовані) зі змінної tempEndState і відновлюється мінімальний маршрут, уже зі збереженням індексів місць відпочинку у змінній списку під назвою tour.

Після цього знайдений шлях повертається у головну функцію класу. Реалізація третього етапу представлена на рис. 3.6.

```
// Connect tour back to starting node and minimize cost.
for (int i = 0; i < N; i++) {
    if (i == start) continue;
    for (int possibleEndState : possibleEndStates)
        try {
            double tourCost = memo[i][possibleEndState];
            if (tourCost < minTourCost) {
                minTourCost = tourCost;
                tempEndState = possibleEndState;
            }
        } catch (Exception ignored) {
        }

}
```

Рис. 3.5 Лістинг реалізації другого етапу алгоритму

```
int lastIndex = start;
int state = tempEndState;

// Reconstruct TSP path from memo table.
for (int i = 1; i < maxNodesNumber; i++) {

    int index = -1;
    for (int j = 0; j < N; j++) {
        if (j == start || notIn(j, state)) continue;
        if (index == -1) index = j;
        double prevDist = lastIndex == start ? memo[index][state] : memo[index][state] + distance[index][lastIndex];
        double newDist  = lastIndex == start ? memo[j][state] : memo[j][state] + distance[j][lastIndex];
        if (newDist < prevDist) {
            index = j;
        }
    }

    tour.add(index);
    state = state ^ (1 << index);
    lastIndex = index;
}
tour.add(start);
Collections.reverse(tour);
return new TemporaryPath(minTourCost, tour);
```

Рис. 3.6 Лістинг реалізації третього етапу алгоритму

Описана функція, що реалізовує алгоритм комівояжера, викликається у функції buildPath(int radius, double lat, double lon, int venuesAmount) класу RouteServiceImpl. Ця функція приймає значення радіусу пошуку, координати центру кола, в якому буде проводитись пошук і максимальну кількість місць відпочинку, які користувач бажає відвідати. Лістинг функції зображено на фрагменті коду на рис. 3.7.

Функція buildPath спочатку звертається до бази даних і отримує список місць відпочинку у заданій локації. Далі ініціалізується матриця довжин маршрутів між вказаними місцями. Кожному місцю відпочинку присвоюється

порядковий номер, який при виконанні алгоритму комівояжера буде асоціюватись з відповідним місцем відпочинку. Після цього для кожного елементу матриці знаходиться місця відпочинку з відповідними порядковими номерами, знаходиться відстань між ними, рейтинг місць відпочинку і на основі цих даних обчислюється вага ребра між двома місцями відпочинку.

Метод getBestPath() повертає список з цілих чисел, де кожне число відповідає певному місцю відпочинку. Після отримання цього списку на основі отриманих індексів відновлюється маршрут уже з вибраних місць відпочинку.

```
@Override
public List<Venue> buildPath(int radius, double lat, double lon, int venuesAmount) {
    List<Venue> venuesInCircle = mapService.getVenuesInRadiusList((double) radius, lat, lon);
    if(venuesInCircle.isEmpty()){
        return new ArrayList<>();
    }
    double[][] distanceMatrix = new double[venuesInCircleSize][venuesInCircleSize];
    for(int i = 0; i < venuesInCircleSize; i++){
        Venue iVenue = venuesInCircle.get(i);
        for (int j = 0; j < venuesInCircleSize; j++){
            if(i == j) {
                distanceMatrix[i][j] = 0;
                continue;
            }
            Venue jVenue = venuesInCircle.get(j);
            double distance = mapService.getDistance(iVenue.getLat(), iVenue.getLon(), jVenue.getLat(), jVenue.getLon());
            double jvenueRating = jVenue.getRating();
            distanceMatrix[i][j] =  jvenueRating < 3 ? distance * 1.67 : distance * (5.0 / jVenue.getRating());
        }
    }
    List<Integer> route = travellingSalesmanAlgorithm.getBestPath(distanceMatrix, venuesAmount);
    List<Venue> resultVenues = new ArrayList<>();
    for(Integer i : route){
        resultVenues.add(venuesInCircle.get(i));
    }
    return resultVenues;
}
```

Рис. 3.7 Лістинг функції buildPath()

Також варто звернути увагу на клас MapServiceImpl у якому реалізований функціонал, який зв’язаний з мапою і операціями з нею. Зокрема у цьому класу реалізована функція getVenuesInRadiusList(Double radius, Double lat, Double lon), що приймає радіус і координати локації, потім звертається до бази даних, отримує список місць відпочинку, перевіряє чи місце відпочинку знаходиться у вказаній локації і потім повертає список бажаних місць відпочинку.

Обчислення відстані між центром кругу і конкретним місцем відпочинку відбувається у методі getDistance(Double startLat, Double startLon, Double endLat, Double endLon), який приймає координати початкової і кінцевої точки, потім

переводить значення довготи і широти в радіани і обчислює відстань між ними. Лістинг описаних функцій зображено на рис. 3.8.

```
@Override
public List<Venue> getVenuesInRadiusList(Double radius, Double lat, Double lon) {
    return venueService
            .getVenuesInRadius(lat, lon, radius: radius / 50)
            .stream().filter(venue ->
                    this.getDistance(lat, lon, venue.getLat(), venue.getLon()) < radius
            )
            .collect(Collectors.toList());
}

@Override
public Double getDistance(Double startLat, Double startLon, Double endLat, Double endLon) {
    double startLatRad = toRadians(startLat);
    double startLonRad = toRadians(startLon);

    double endLatRad = toRadians(endLat);
    double endLonRad = toRadians(endLon);

    double result = acos((sin(startLatRad) * sin(endLatRad) +
            cos(startLatRad) * cos(endLatRad) * cos(startLonRad - endLonRad))) * EARTH_RADIUS;
    return result;
}
```

Рис. 3.8 Лістинг функцій класу MapService

3.7 Опис технологій використаних для розробки інтерфейсу системи

Розмітка веб-сторінок у системі виконана на стандартизованій мові розмітки HTML. Мова HTML інтерпретується браузером, а отриманий у результаті текст, зображення чи відео відображається на екрані монітору. Кожен документ, який створено використання HTML, являє собою набір елементів, де кожен з обох сторін має позначки, іменовані тегами.

Документ, створений з використанням HTML містить визначені стандартом елементи і HTML надає можливість створити тільки ескіз сторінки. Для того, щоб зробити сторінку зручнішою і приємнішою для користувача було використано CSS. CSS – це формальна мова опису зовнішнього вигляду документу, який написано за допомогою мови розмітки, у нашому випадку мови HTML. У проекті за допомогою CSS задавались кольори інтерфейсу та шрифтів, власне шрифти, положення окремих блоків на сторінці, а також відображення інших аспектів представлення зовнішнього вигляду сторінок. Використання CSS

дозволяє розділити опис логічної структури веб-сторінок від опису зовнішнього вигляду цих сторінок. Таке розділення збільшує доступність документу, надає гнучкість розробки і можливість керування його зовнішнім виглядом, а також зменшує складність і повторюваність в структурному змісті. Крім того, за допомогою CSS сторінку можна зробити адаптивною, тобто відображати її по різному, для пристроїв з різним розширенням екрану, що робить зручнішим використання застосунку.

Для вирішення задач, які зв'язані з динамічним завантаженням сторінок і даних на них, було використано скриптову мову програмування JavaScript. Вона дозволяє добавити сторінкам інтерактивності.

Для зручності і пришвидшення розробки програмного забезпечення було використано бібліотеку JQuery. Вона орієнтована на взаємодію мови JavaScript і HTML. Основною її перевагою є те, що вона допомагає легко отримати доступ до будь-якого елементу DOM [40], об'єктної моделі документу, а також дозволяє звертатись до атрибутів і змісту отриманих елементів. Однією з основних переваг бібліотеки є те, що вона надає зручний програмний інтерфейс для роботи з AJAX.

Для покращення досвіду користувача у роботі з системою було реалізовано підхід до побудови користувацьких інтерфейсів веб-застосунків AJAX. Він полягає у тому, що дані передаються з браузера до веб-серверу у «фоновому» режимі. При такому підході при необхідності завантажити певні дані скрипт визначає, яка конкретно інформація потрібна і робить відповідний запит на сервер, сервер у відповідь повертає тільки ту частину документу, на яку прийшов запит, а скрипт вносить зміни, відповідно до отриманої інформації. У результаті при оновленні даних веб-сторінка перезавантажується частково, що пришвидшує веб-застосунок і робить його зручнішим. Також такий підхід дозволяє значно зменшити трафік, так як з серверу передається тільки необхідна інформація, яку потрібно змінити, відповідно зменшуються навантаження на сервер і пришвидшується реакція інтерфейсу. Технологія є асинхронною, що дає можливість за допомогою браузера відіслати запит на сервер, а потім робити що

завгодно на веб-сторінці очікуючи відповідь якщо передбачається, що вона буде [41].

Технологія AJAX має такі властивості:

– для отримання даних і стилізації інформації використовується HTML і CSS;

– для того щоб забезпечити динамічне відображення і взаємодію з інтерфейсом DOM-моделлю використовується JavaScript і описана вище бібліотека JQuery;

– XMLHttpRequest для виеористовується для асинхронного обміну даними з веб-сервером;

– дані між клієнтом і сервером передаються у форматі JSON.

Класичний застосунок з використанням AJAX складається з двох частин. Перша виконується в браузері і пишеться на JavaScript, друга на сервері і написана на Java. Між даними частинами відбувається обмін даними через асинхронний XMLHttpRequest.

3.8 Створення адаптивності

Із системою побудови оптимального шляху по рекреаційних зонах можна працювати з різних платформ. Запустити її можна за допомогою браузера на стаціонарному комп'ютері, ноутбуці, смартфоні або планшеті. Зовнішній вигляд сайту розроблявся адаптивно, щоб зручно відображатися на будь-якому розширенні екрану.

Для створення адаптивної інтерактивної системи використовувався фреймворк Bootstrap. Це набір інструментів для створення сайтів і веб-застосунків, який поширюється у вільному доступі [42]. Бібліотека підключається шляхом збереження мінімалізованих файлів зі стилями і JavaScript у папку ресурсів, після чого спеціалізовані класи Bootstrap можна задіяти в верстці HTML для відображення сторінок.

Основна перевага фреймворку – це якісно реалізована grid-сітка для масштабування веб-сторінки і створення адаптивного дизайну. Класична grid-сітка є спробою прискорити розробку сайтів завдяки використанню форматів, які розраховані на ширину в 960 пікселів. Найбільш поширеним варіантом вважається сітка на 12 колонок.

До основних інструментів Bootstrap відносяться:

– сітка – заздалегідь задані розміри колонок, які можна відразу використовувати, наприклад за допомогою класу .col-md-2;

– шаблони – фіксований або гумовий шаблон документа;

– форми – класи для створення адаптивний форм;

– типографіка – опис для шрифтів, наявність певних класів для роботи зі шрифтами на кшталт коду і цитат;

– медіа –дає певні можливості для управління зображеннями і відео;

– навігація – спеціальні класи для оформлення вкладок, меню, панелі інструментів і інших варіантів навігації;

– алерт – стилізація підказок, діалогових і спливаючих вікон [33].

Система побудови оптимального шляху по розважальних заходах виглядає на різних пристроях наступним чином – вигляд сайту на екрані монітору ноутбука чи комп'ютера з високим розширенням екрану зображено на рисунках 3.9 та 3.10.

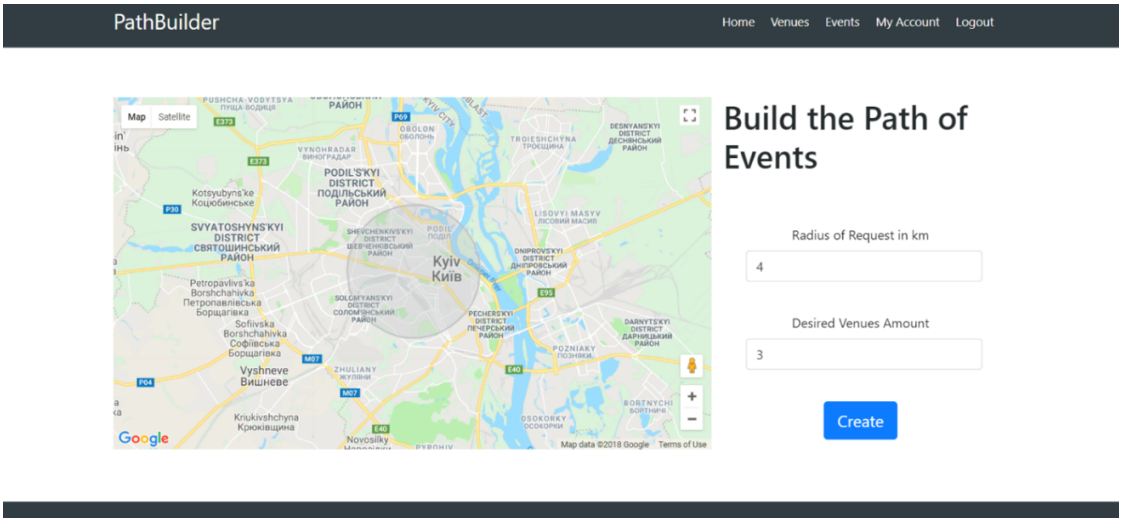

Рис. 3.9 Вигляд головного вікна додатку при перегляді на ПК

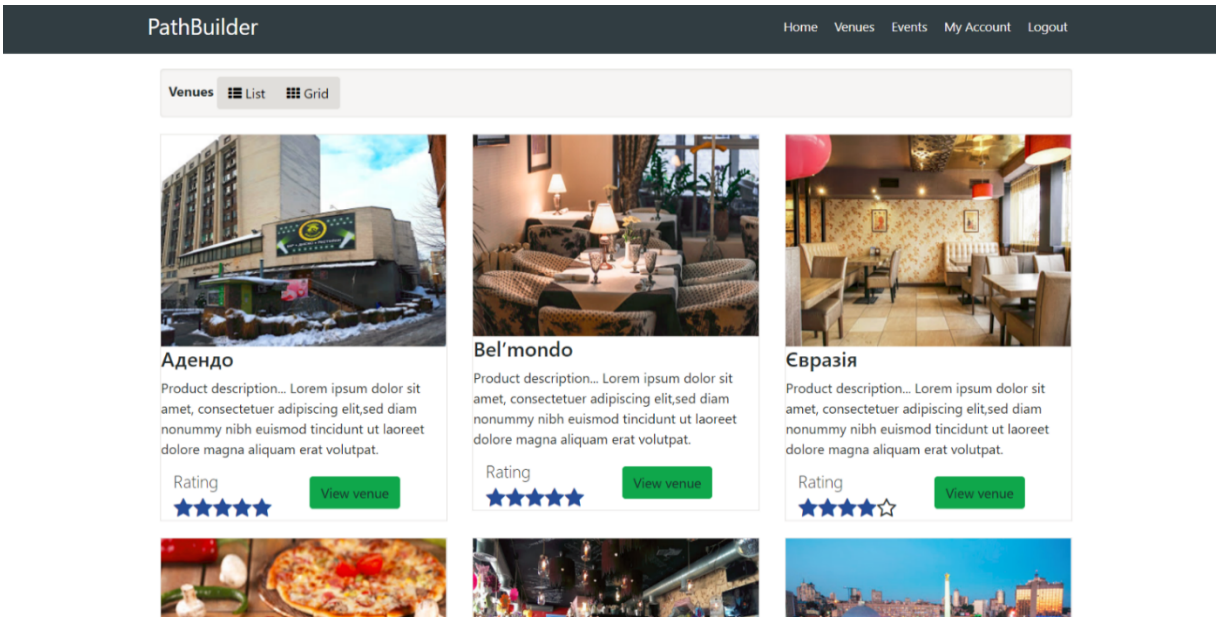

Рис. 3.10 Вигляд вікна закладів при перегляді на ПК

Вигляд сайту система на екрані смартфону з малим розширенням екрану зображено на рисунках 3.11 та 3.12

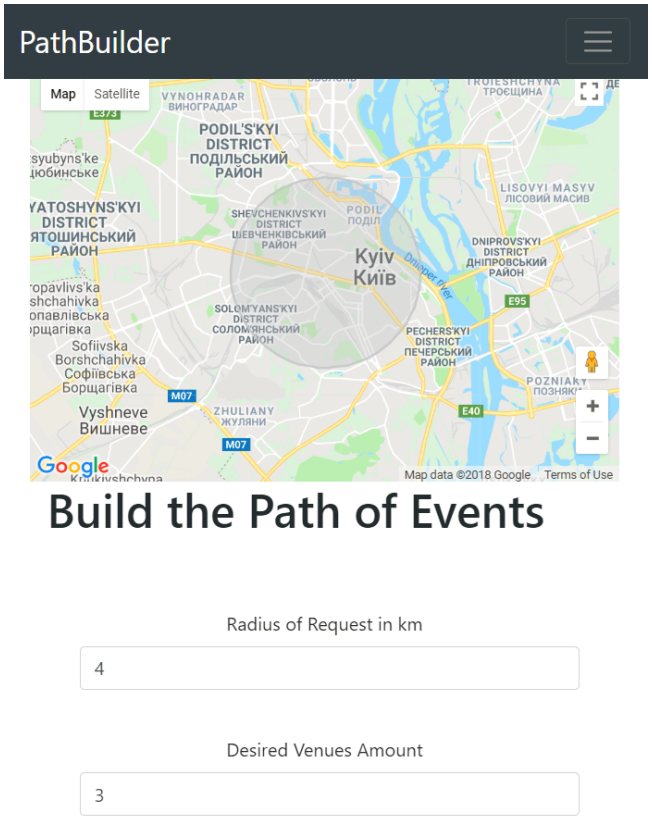

Рис. 3.11 Вигляд головного вікна додатку при перегляді на смартфоні

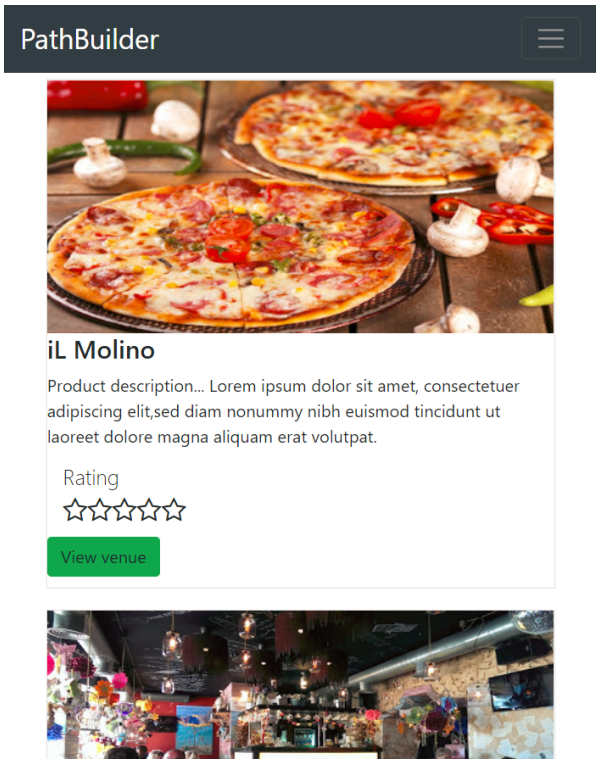

Рис. 3.12 Вигляд вікна закладів при перегляді на смартфоні

Вигляд сайту на екрані планшета з середнім розширенням екрану зображено на рисунках 3.13 та 3.14.

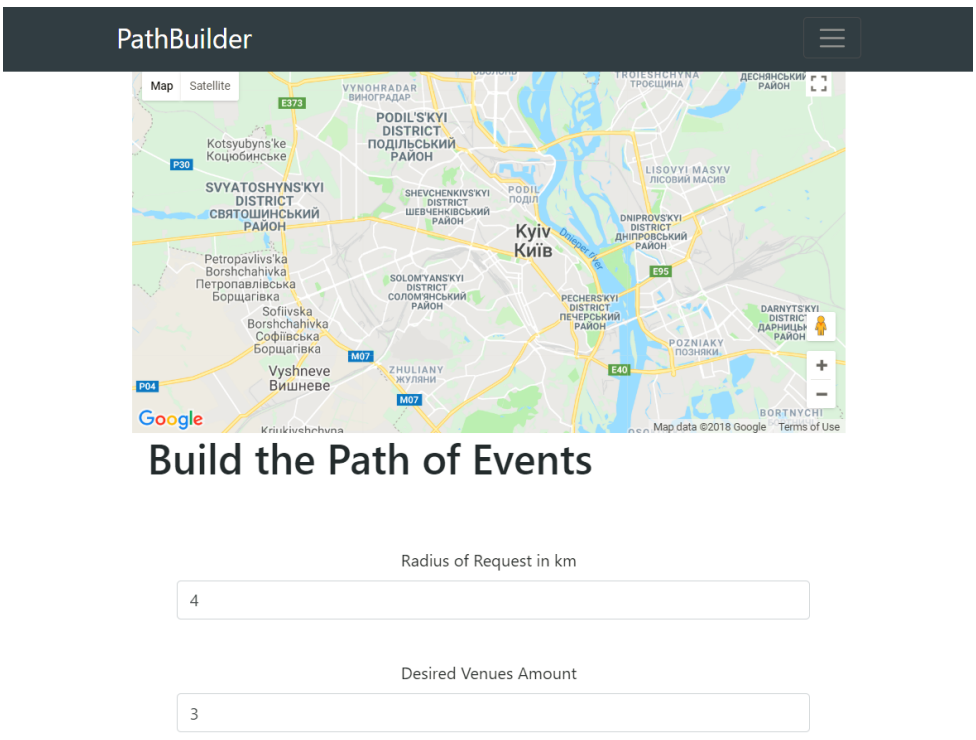

Рис. 3.13 Вигляд головного вікна додатку при перегляді на планшеті

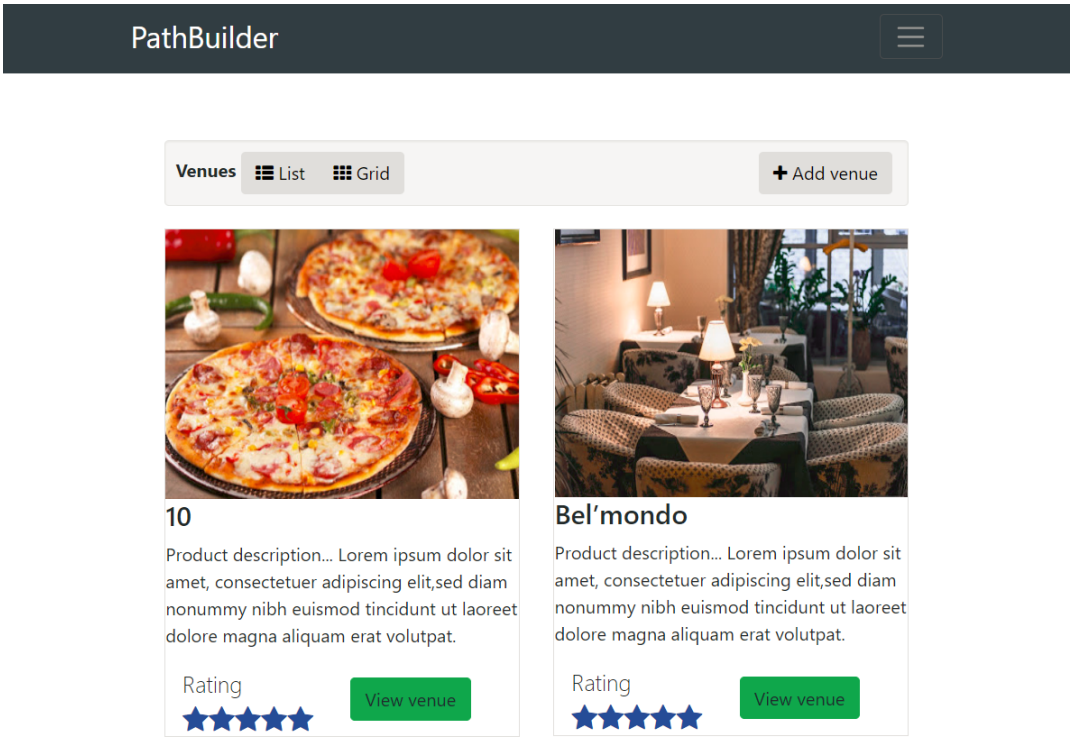

Рис. 3.14 – Вигляд вікна закладів при перегляді на планшеті

3.9 Тестування програмного продукту

Було виділено основні функції, які web-застосування має виконувати, та перевірено їх у процесі тестування. У таблиці 3.6 описано випробування функції входу в систему.

Таблиця 3.6

Авторизація користувача

Мета тесту:	Перевірка функції «Вхід у систему»
Початковий стан web-застосування	Web-застосування запущене в браузері користувача
Вхідні дані:	Логін та пароль користувача
Схема проведення тесту:	Ввести логін та пароль у відповідні поля, натиснути кнопку «Sign In»
Очікуваний результат:	Відкрита головна сторінка web-застосування
Стан web-застосування після проведення випробувань:	Відкрита головна сторінка web-застосування

У таблиці 3.7 описано випробування функції додавання рекреаційної зони.

Таблиця 3.7

Додавання рекреаційної зони

Мета тесту:	Перевірка функції «Додавання рекреаційної зони»
Початковий стан web-застосування	Web-застосування запущене на пристрої користувача. Користувач авторизований в системі.
Вхідні дані:	Назва місця відпочинку, розташування місця відпочинку (закладу).
Схема проведення тесту:	Авторизуватися як менеджер чи адміністратор. Перейти на сторінку усіх місць відпочинку. Натиснути на клавішу створення нового місця відпочинку. Створити місце відпочинку.
Очікуваний результат:	Відкрита сторінка списку місць відпочинку з новим місцем відпочинку на ній.
Стан web-застосування після проведення випробувань:	Відкрита сторінка списку місць відпочинку з новим місцем відпочинку на ній.

У таблиці 3.8 описано випробування функції додавання події.

Таблиця 3.8

Додавання події

Мета тесту:	Перевірка функції «Додавання події»
Початковий стан web-застосування	Web-застосування запущене на пристрої користувача. Користувач авторизований в системі.
Вхідні дані:	Назва події, назва місця відпочинку (закладу), у якому подія створюється, дата початку і дата кінця події.
Схема проведення тесту:	Авторизуватися як менеджер чи адміністратор. Перейти на сторінку усіх подій. Натиснути на клавішу створення події. Створити подію.
Очікуваний результат:	Відкрита сторінка списку подій з новою подією на ній.
Стан web-застосування після проведення випробувань:	Відкрита сторінка списку подій з новою подією на ній.

У таблиці 3.9 описано випробування функції генерації маршруту.

Таблиця 3.9

Генерація маршруту

Мета тесту:	Перевірка функції «Додавання новини»
Початковий стан web-застосування	Web-застосування запущене на пристрої користувача. Користувач авторизований в системі.
Вхідні дані:	Цент бажаної локації, її радіус і максимальна кількість закладів.
Схема проведення тесту:	Перейти на головну сторінку. Заповнити всі дані і натиснути на кнопку «Create».
Очікуваний результат:	На карті побудований маршрут, а знизу на головній сторінці відображено список закладів з маршруту.
Стан web-застосування після проведення випробувань:	На карті побудований маршрут, а знизу на головній сторінці відображено список закладів з маршруту.

3.10 Демонстрація і опис роботи програми

Доцільно буде змоделювати реальну ситуацію в якій користувач заходить на сайт вперше і користується його функціями.

Для початку користувачу необхідно зареєструватись у системі. Для цього необхідно заповнити усі поля на сторінці реєстрації (рис. 3.15), а саме адресу електронної пошти, ім’я, прізвище і пароль від акаунту.

Після цього користувача автоматично перенаправить на домашню сторінку застосунку, яка зображена на рис. 3.17. Наступного разу коли користувач буде використовувати систему йому потрібно буде зайти у систему через уже створений профіль. Для цього потрібно заповнити усі поля на сторінці логіну (рис. 3.16) і натиснути на «SIGN IN».

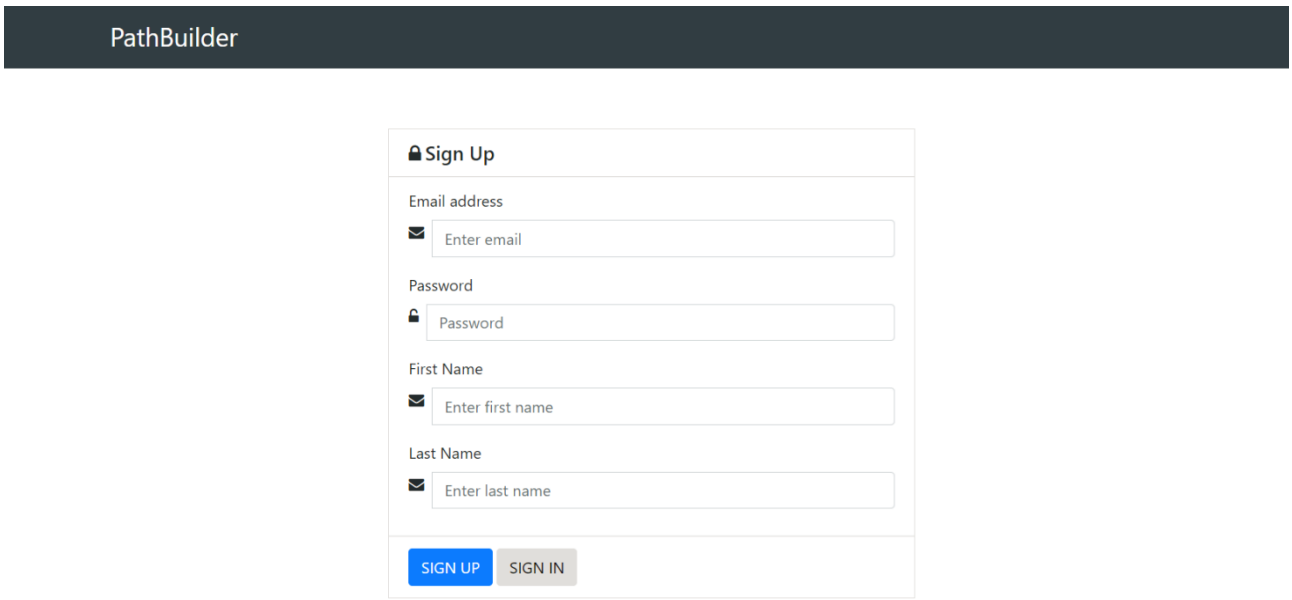

Рис. 3.15 Сторінка реєстрації користувача

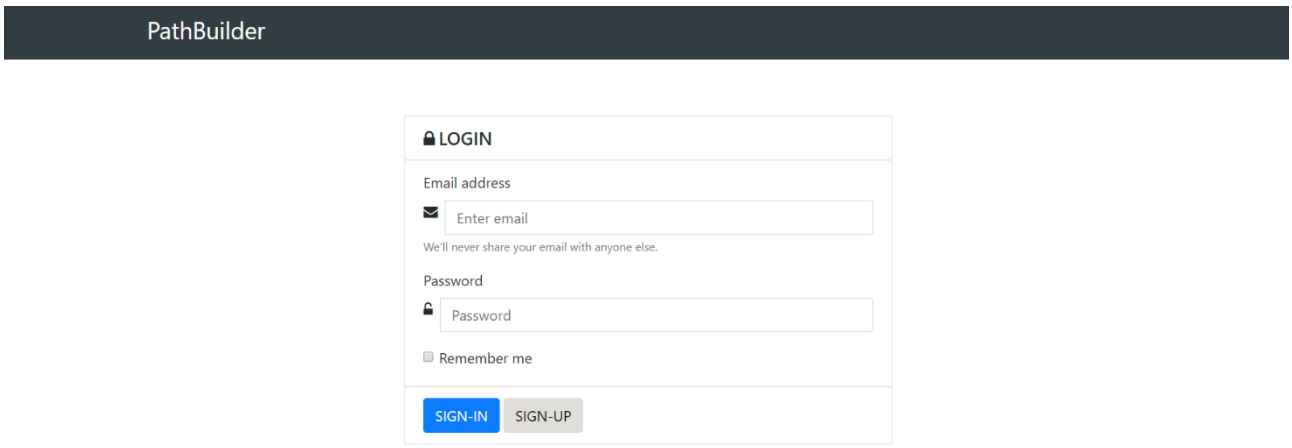

Рис. 3.16 – Сторінка входу

З головної сторінки користувач може перейти на інші сторінки системи, а також побудувати маршрут із місць відпочинку. Розглянемо ці можливості детальніше. Основною функцією веб-застосунку є генерація шляху з рекомендованих до відвідання рекреаційних зон. Для цього користувачу необхідно вибрати локацію, в якій він бажає відпочивати, її радіус і кількість місць відпочинку, які він хоче відвідати. Після цього йому необхідно натиснути на кнопку «Create» і на карті згенерується рекомендований маршрут (рис. 3.18). Також знизу під картою відображається список місць відпочинку у маршруті з

короткою інформацією про них. Натиснувши на кнопку «View venue» можна перейти на сторінку відображення детальної інформації про місце відпочинку (рис. 3.19).

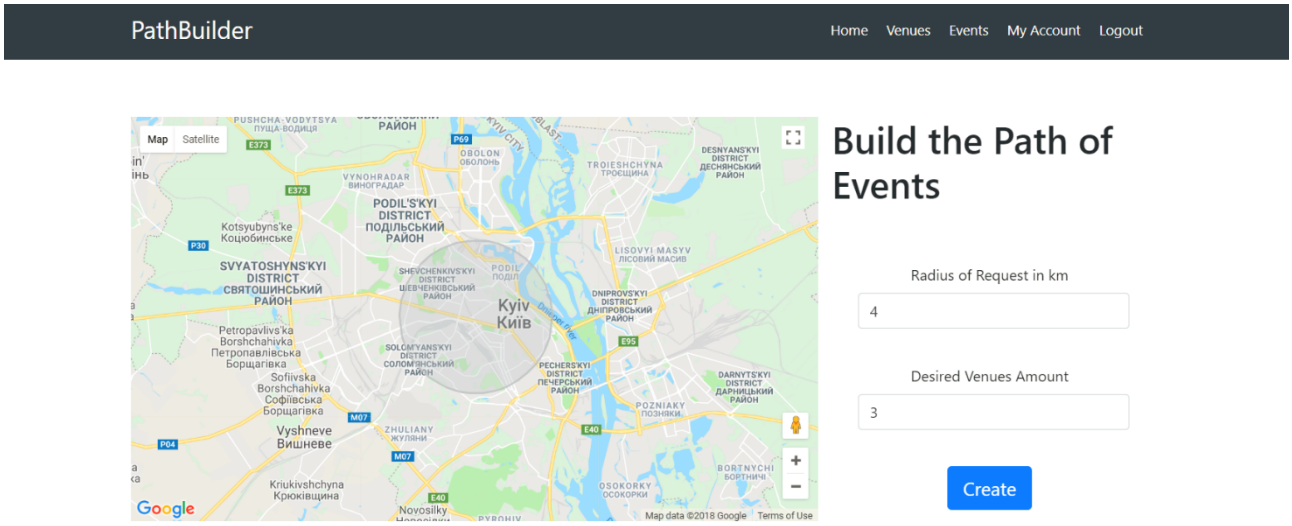

Рис. 3.17 Домашня сторінка застосунку

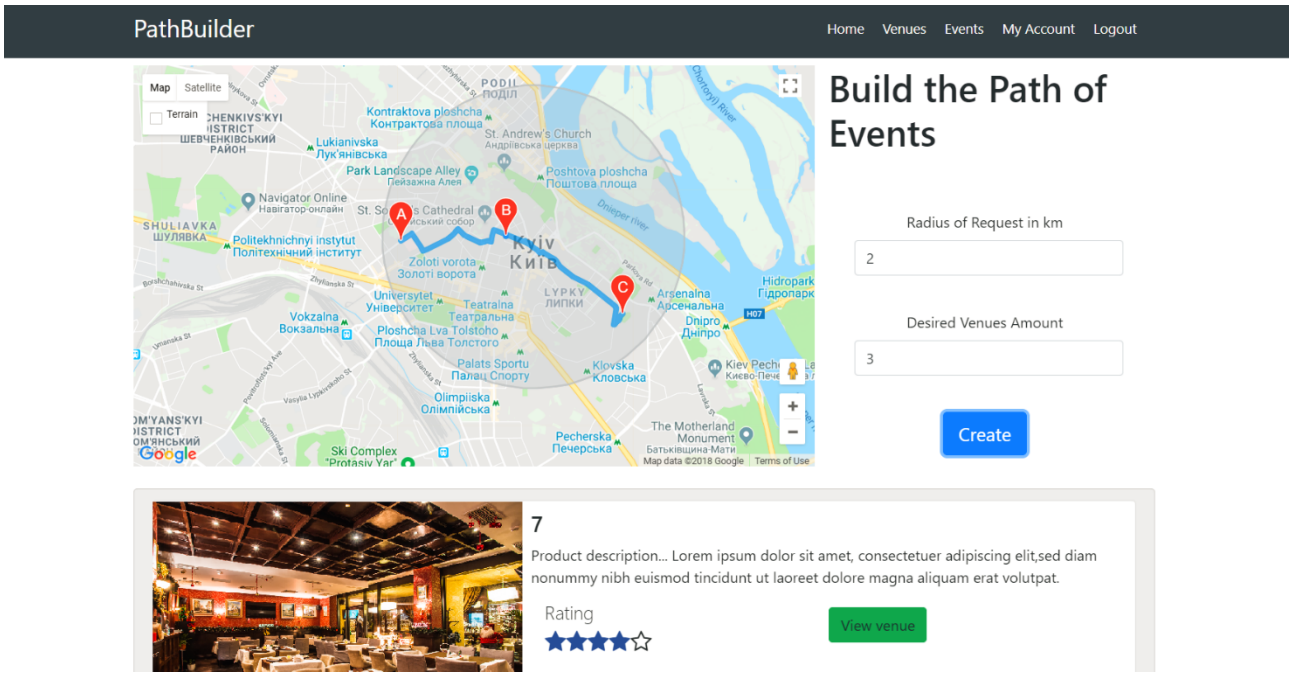

Рис. 3.18 Результат побудови маршруту

На сторінці місця відпочинку відображено назву місця відпочинку, його детальний опис, середній рейтинг, список подій і список користувацьких відгуків про дане місце відпочинку.

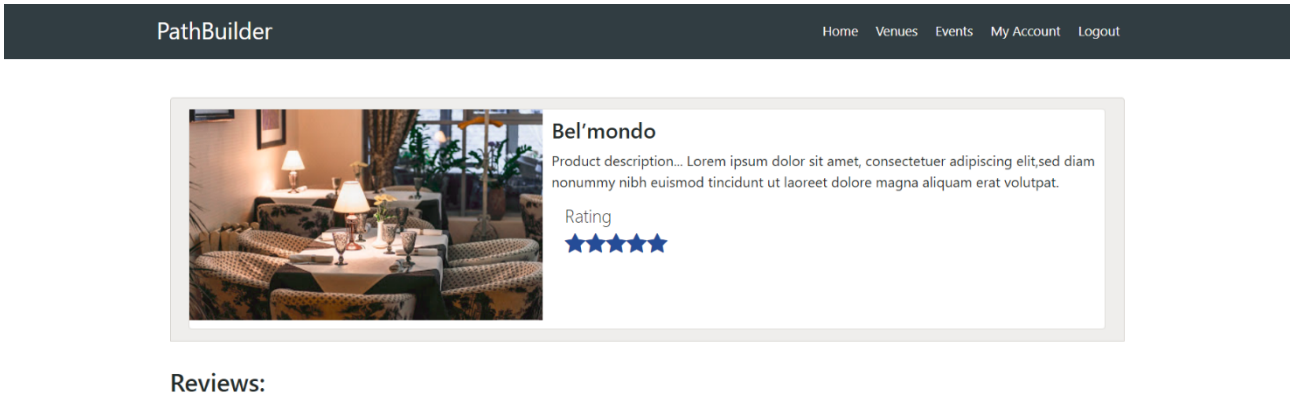

Рис. 3.19 Сторінка місця відпочинку

З будь якої сторінки через навігацію у верхній частині сторінки можна потрапити на сторінку місць відпочинку (рис. 3.20).

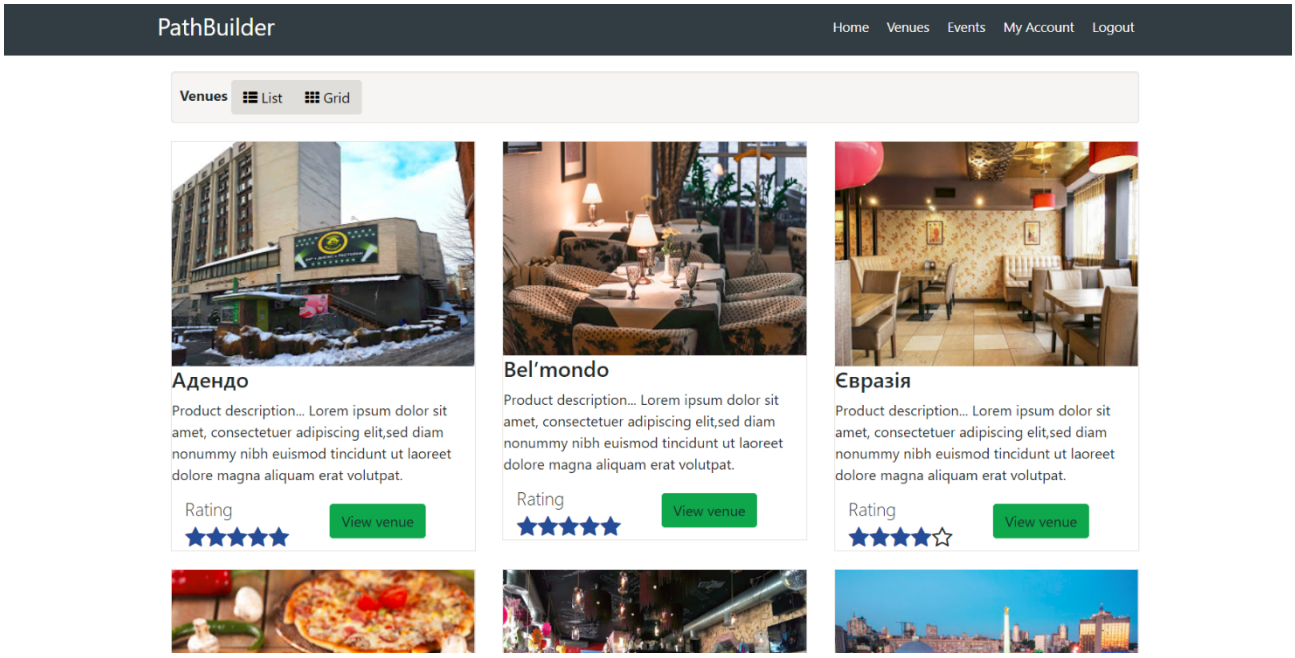

Рис. 3.20 Сторінка місць відпочинку

На сторінці місць відпочинку відображається список усіх місць відпочинку уведених в систему. Його можна відобразити у вигляді сітки чи у виді списку із елементів. Користувач може дати оцінку певному місцю відпочинку, натиснувши на відповідній зірці у блоці відображення рейтингу.

Також з навігаційної панелі користувач може перейти на власний профіль (рис. 3.21), де розміщена основна інформація про користувача.

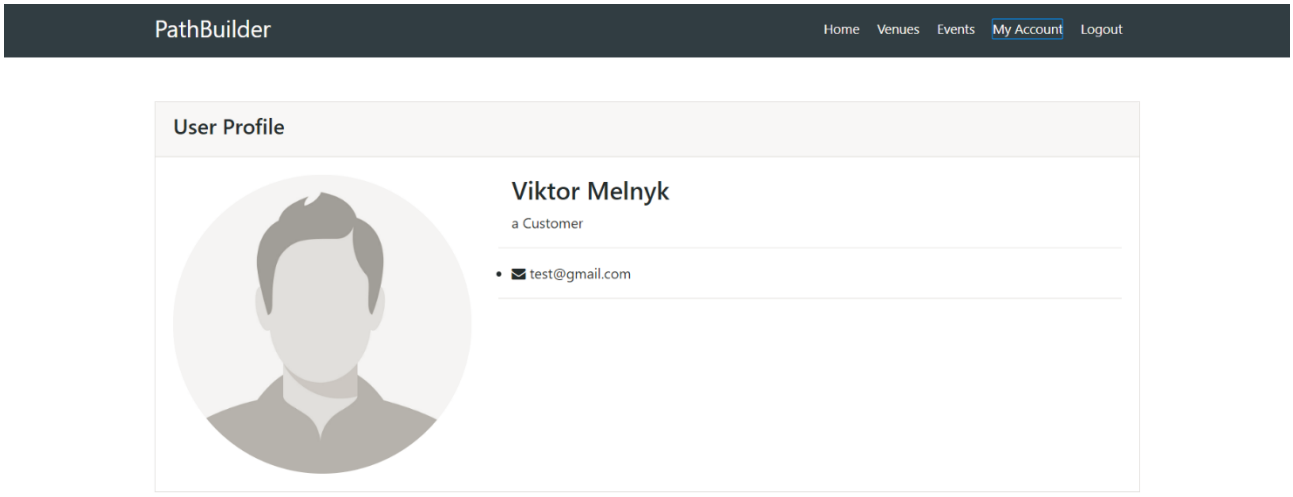

Рис. 3.21 Профіль користувача

Також користувач може переглянути список активних чи запланованих подій, які відбуваються у закладах. Для цього потрібно перейти на вкладку «Events» в панелі навігації.

Менеджери закладів мають у застосунку додатковий інтерфейс. У них є можливість створювати нові заклади і події у закладах. Для цього потрібно на сторінці закладів чи подій натиснути на кнопку зі знаком «плюс». Після цього відкриється вікно створення. На рисунку 3.22 зображено вікно створення події.

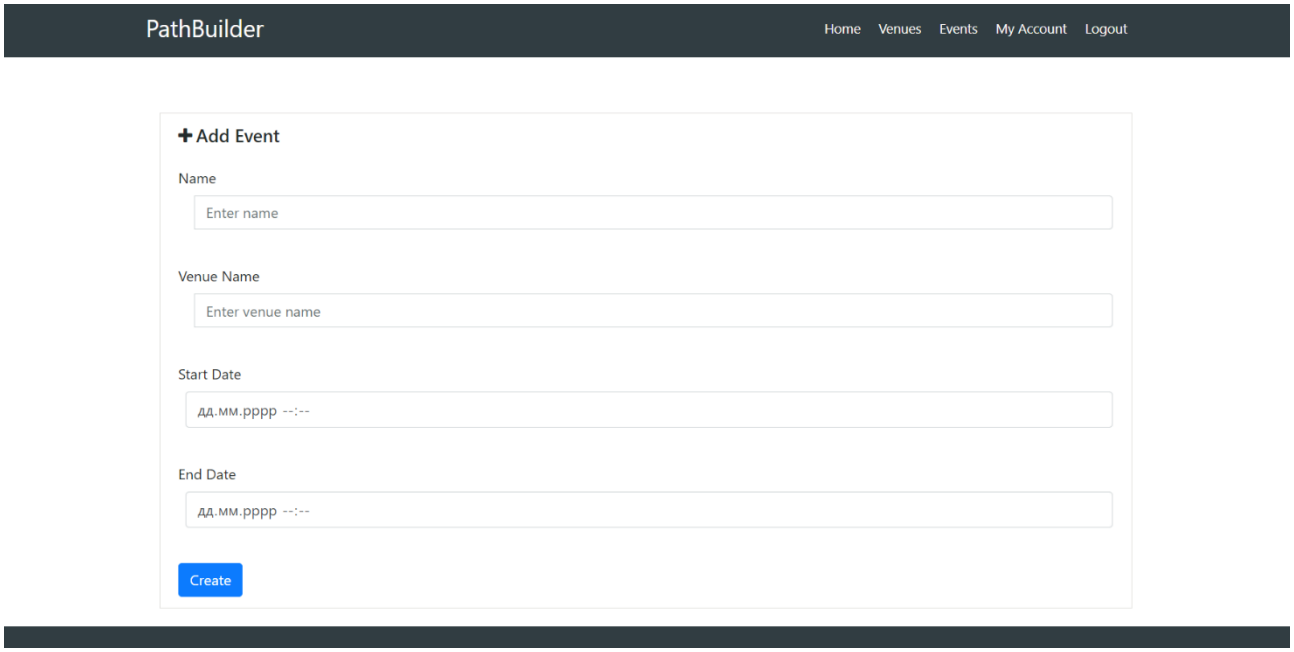

Рис. 3.22 Сторінка створення події

Для того щоб добавити нове місце відпочинку, менеджеру потрібно зайти на сторінку усіх місць відпочинку і натиснути на кнопку зі знаком «плюс» і підписом «Add Venue». Після цього відкриється вікно створення. На рисунку 3.23 зображено вікно створення нового місця відпочинку.

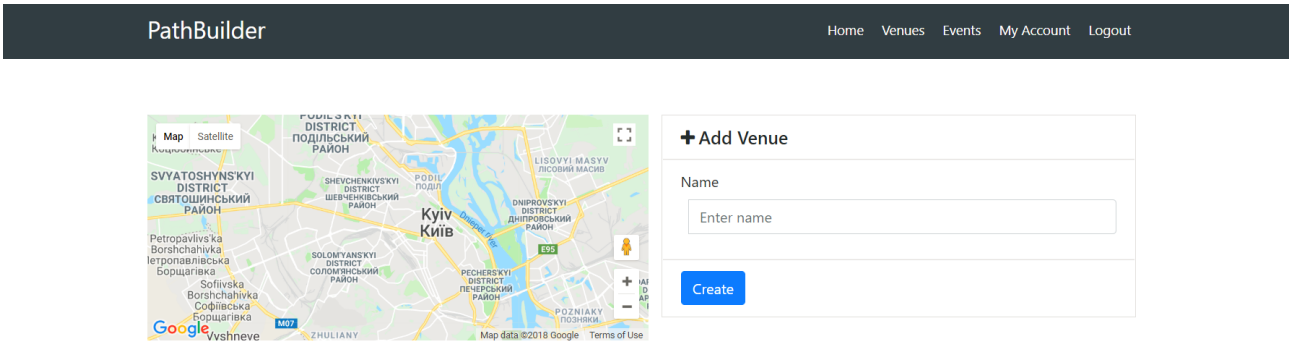

Рис. 3.23 Сторінка додавання нового місця відпочинку

Для того щоб залишити відгук про певне місце відпочинку користувачу необхідно зайти на сторінку конкретного місця відпочинку. На цій сторінці

потрібно натиснути на кнопку «Give Review». Після цього відкриється вікно створення відгуку, яке зображено на рисунку 3.24.

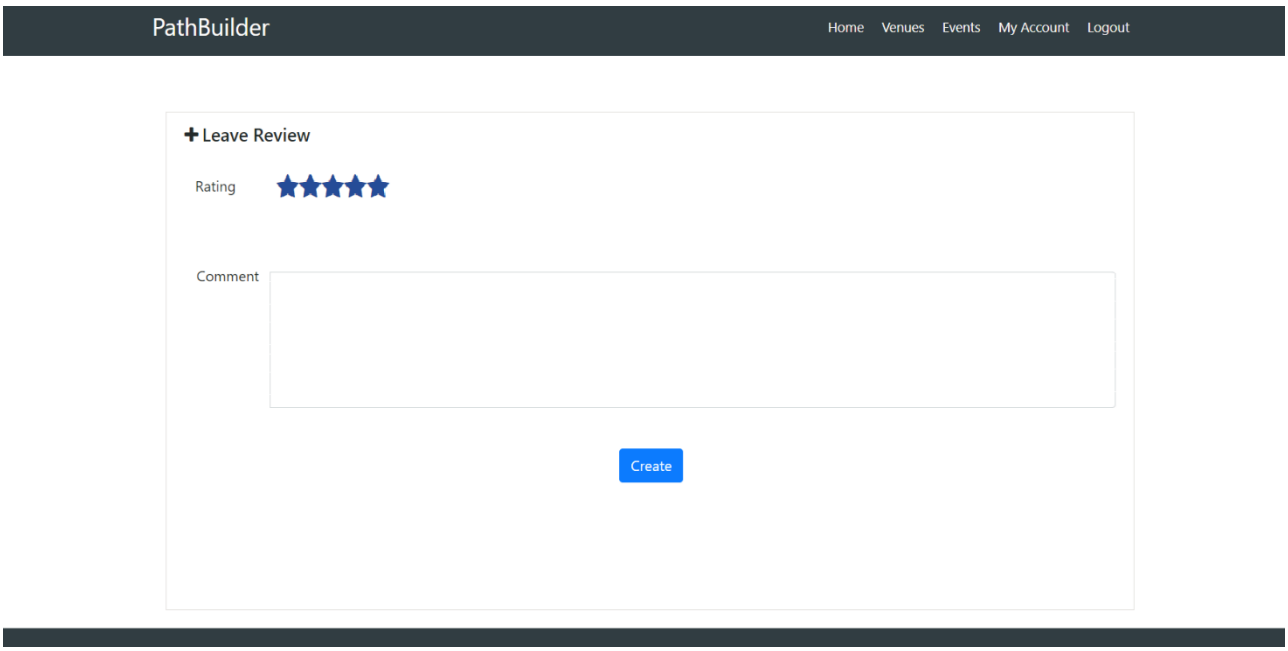

Рис. 3.24 Сторінка залишення відгуку

У вікні залишення відгуку користувач може оцінити місце відпочинку, натиснувши на одну з зірок, а також залишити відгук у полі «Comment». Після заповнення цих полів потрібно натиснути на кнопку «Create».

Для того щоб вийти з поточного акаунту потрібно натиснути на «Logout» у меню. Після цього у користувача перекине на сторінку входу в систему і користувач матиме змогу зайти з іншого акаунту.

3.11 Висновки

В даному розділі було аргументовано та описано вибір інструментів та технологій, що найбільше підходять для розробки системи. Також було описано архітектуру системи, а також основні функції, такі як аутентифікація, авторизація.

Розписано роботу алгоритму пошуку, а також підходи та принципи при створенні інтерфейсу системи.

ВИСНОВКИ

У ході виконання магістерської роботи була розроблена система побудови оптимального маршруту по рекреаційних зонах у заданій локації. Система являє собою веб-сайт з зручним інтерфейсом, який зроблений адаптивно для використання на комп'ютерах, планшетах і смартфонах під управлінням різних операційних систем.

Архітектурою було обрано REST, оскільки така модель є найбільш зручною для реалізації описаної системи. Щоб зробити інтерактивну систему адаптивної для всіх платформ використовувався фреймворк Bootstrap 3. Візуальна складова інтерактивної системи реалізовувалася з використанням HTML + CSS, а також функціоналом бібліотеки JQuery. Також використовувався AJAX для динамічної роботи сторінок сайту.

База даних для системи створена з системою керування базами даних MySQL. Також використовуються інструменти ORM Hibernate і Spring Data. У базі даних зберігаються дані зареєстрованих користувачів. Для аутентифікації у системі використовується Spring Security і Oauth2.0.

Завдяки описаним інструментам розробки було створено великий каталог місць відпочинку. Відвідувачі сайту можуть знайти нові місця відпочинку для відвідання а також побудувати маршрут по найкращих з них. Також користувач може отримати список усіх місць, які збережені у системі, і список подій, які у них відбудуться найближчим часом. Для об’єктивної оцінки якості місць відпочинку у користувачів є можливість оцінити місця відпочинку і залишити відгук про них.

Система може служити великим каталогом місць відпочинку з корисним функціоналом і наявністю докладної інформації для користувачів. Для власника сайту це відкриває можливість продавати рекламні місця. У магістерській роботі були вирішені всі поставлені завдання.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Писаревський І. М. Планування та організація туристських маршрутів: Підручник / І. М. Писаревський. М. В. Тонкошкурі Харк. нац. акад. міськ госпва. -Х. : ХНАМГ. 2012. - 304 с.
2. Бейдик О. О. Словник-довідник з географі туризму, рекреалогії та рекреаційної географії. - К.: Ін-т туризму, 1998
3. Geunes J. Operations Planning: Mixed Integer Optimization Models (Operations Research Series). CRC Press, 2014. P. 167–172.
4. WebGL public wiki [Електронний ресурс] – Режим доступу до ресурсу: https://www.khronos.org/webgl/wiki/Main_Page
5. Cook W. J. In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation. Princeton University Press, 2012. P. 19–39.
6. Applegate D.L., Bixby R.E., Chvátal V., Cook W.J. The Traveling Salesman Problem. Princeton University Press, 2007. P. 44–52
7. Ortega H. The Shortest Path Problem, 2007 p. – 310 c.
8. Rocca M. Advanced Algorithms and Data Structures, 2021 p. – 768 c.
9. Cormen T. Introduction to Algorithms. Fourth Edition, 2022 p. –. – 681 c
10. Dynamic Programming [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/dynamic-programming/
11. P vs NP problem [Електронний ресурс] – Режим доступу до ресурсу: https://www.claymath.org/wp-content/uploads/2022/06/pvsnp.pdf
12. Branch and Bound algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/branch-and-bound-algorithm/
13. Cutting plane algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/topics/engineering/cutting-plane
14. R. Trudeau Introduction to graph theory 1993 p. – 412 c.
15. Unveiling the Held-Karp Algorithm: Revolutionizing the Traveling Salesman Problem [Електронний ресурс] – Режим доступу до ресурсу:

https://medium.com/@data-overload/unveiling-the-held-karp-algorithm-revolutionizing-the-traveling-salesman-problem-9fb45b4cf58d

16. An Effective Heuristic Algorithm for the Traveling-Salesman Problem [Електронний ресурс] – Режим доступу до ресурсу: https://www.cs.princeton.edu/~bwk/btl.mirror/tsp.pdf

17. Glover F. Tabu Search, Springer, 1997. P. 401.

18. ACO Algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/topics/engineering/ant-colony-optimization

19. Ant colony optimization [Електронний ресурс] – Режим доступу до ресурсу: http://www.scholarpedia.org/article/Ant_colony_optimization

20. Сервіс побудови маршрутів для велосипедистів [Електронний ресурс] – Режим доступу до ресурсу: http://outdoorsgps.com/

21. Сервіс збереження маршрутів по закладах [Електронний ресурс] – Режим доступу: https://www.strava.com/

22. Сервіс для побудови маршрутів на карті [Електронний ресурс] – Режим доступу: https://www.plotaroute.com/routeplanner

23. Сервіс пошуку розважальних закладів на карті [Електронний ресурс] – Режим доступу: https://www.google.com/maps/search

24. Goodrich M., Roberto T. Algorithm Design and Applications, Wiley, 2015. P. 513–514.

25. Діаграма послідовності [Електронний ресурс] – Режим доступу до ресурсу: https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema13/tema13_3

26. M Paul DuBoisy SQL Cookbook: Solutions for Database Developers and Administrators 3rd Edition, 2014 p. – 869 c.

27. М. Фаулер UML. Короткий посібник по UML. 1997 – 169 с.

28. Christian Bauer and Gavin King Java Persistence Api і Hibernate, 2017 p. – 888 c.

29. Catalin Tudose Java Persistence with Spring Data and Hibernate, 2023 p. – 435 c.

30. Hemrajani A. Agile Java Development with Spring 2016 р. – 401 с.

31. Risberg T., Brisbin J. Spring Data for Web Development 2012 р.– 316 с.

32. Spring Data JPA [Електронний ресурс] – Режим доступу: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

33. Spring Security – Roles and Privileges [Електронний ресурс] – Режим доступу: http://www.baeldung.com/role-and-privilege-for-spring-security-registration

34. Log4J Overview [Електронний ресурс] – Режим доступу: https://logging.apache.org/log4j/2.x/manual/index.html

35. Richardson L. RESTful Web APIs: Services for a Changing World – 406 с.

36. Ричер Д. Oauth2.0 в дії 2017 р. – 400 с.

37. Understanding OAuth2 REST [Електронний ресурс] – Режим доступу: http://www.bubblecode.net/en/2016/01/22/understanding-oauth2/

38. Алгоритм комівояжера [Електронний ресурс] – Режим доступу: https://github.com/williamfiset/Algorithms/blob/master/com/williamfiset/algorithms/graphtheory/TspDynamicProgrammingIterative.java

39. IntelliJ IDEA – JetBrains [Електронний ресурс] – Режим доступу до ресурсу: https://jetbrains.com/products/idea/

40. Бенкен Е. AJAX: the definitive guide, 2012 р. – 289 с.

41. Holdener A. Bootstrap: Responsive Web Development, 2008 р. – 250 с.

42. Design responsive websites using Bootstrap [Електронний ресурс] – Режим доступу: https://helpx.adobe.com/in/dreamweaver/using/bootstrap.html