МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

МАЗУРУК ОЛЕГ ВОЛОДИМИРОВИЧ

Допускається до захисту:
завідувач кафедри
інформаційних технологій,
д. т. н., доцент
_____ Т. В. Нескородєва
«_____» _____ 2022р.

Дослідження автоматизованої системи агрегації та побудови туристичних маршрутів

Спеціальність 122 «Комп'ютерні науки»

Кваліфікаційна (магістерська) робота

Науковий керівник:
Федоров Є.Є., професор
кафедри інформаційних технологій
д.т.н., професор

_____
(підпис)

Оцінка: _____/_____/_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2022

АНОТАЦІЯ

Мазурук О.В. Дослідження автоматизованої системи агрегації та побудови туристичних маршрутів.

У магістерські роботі здійснено аналіз предметної області та додатків що дозволяють виконати автоматичну побудову туристичного маршруту.

Розроблено математичну модель пошуку оптимального туристичного маршруту.

Розроблено автоматизовану систему для агрегації та побудови туристичних маршрутів.

Ключові слова: Інформаційно-аналітична система, мурашиний алгоритм, Java, Spring, SQL, ReactJS, Docker, Grafana.

Табл. 2. Рис. 41. Бібліограф.: 77 найм.

ABSTRACT

Mazuruk O.V. Research of the automated system of aggregation and construction of tourist routes.

The master's thesis analyzes of the subject area and applications that allow automatic construction of a tourist route was carried out.

The mathematical model of finding the optimal tourist route has been developed.

The automated system for aggregation and construction of tourist routes has been developed.

Keywords: Information-analytical system, ant colony optimization, Java, Spring, SQL, ReactJS, Docker, Grafana.

Tabl. 2. Fig. 41. Bibliography: 77 items.

ЗМІСТ

ВСТУП

Актуальність роботи: «Туристичний маршрут - це попередньо спланована туристами або суб'єктом туристичної діяльності подорож, що може охоплювати один чи декілька туристських шляхів і характеризується визначеним порядком пересування туристів через певні географічні пункти» [1]. Аби вміло планувати туристичний маршрут необхідно врахувати безліч факторів, щоб майбутня подорож виявилась достатньо цікавою.

Цікавий та передбачуваний маршрут – те, чого прагне кожний турист, щоб комфортно почувати себе під час подорожі. Один із найдоступніших способів планування маршрутів – туристичні фірми, проте зазвичай це додаткові витрати і заангажованість працівників спонсорськими домовленостями.

Також в даний час розроблена досить велика кількість додатків, які дозволяють самостійно спланувати майбутню подорож. При здійсненні вибору додатку, варто враховувати можливості планування – вільний час протягом дня, запланована кількість днів для подорожей, географічні пункти які є у пріоритеті відвідування, пріоритетний засіб пересування.

Мета дослідження: Розробити автоматизовану систему агрегації та побудови маршрутів.

Завдання дослідження:

- Здійснити аналіз предметної області;
- Здійснити аналіз переваг та недоліків існуючих додатків для побудови туристичних маршрутів;
- На основі отриманих результатів розробити автоматизовану систему агрегації та побудови маршрутів.

Об’єкт дослідження: процес агрегації та побудови туристичних маршрутів.

Предмет дослідження: Застосування Web-технологій для створення автоматизованої системи агрегації та побудови туристичних маршрутів.

Практичне значення одержаних результатів: Можливість функціонування системи в якості незалежного Web-додатку, взаємодія із іншими додатками використовуючи різні протоколи передачі даних.

Зв'язок роботи з науковими програмами, планами, темами. Наведені у кваліфікаційні роботі дослідження пов'язані з фундаментальною науково-дослідною роботою «Дослідження та комп'ютерно-математичне моделювання складних систем та процесів у науці, освіті та інформаційно-комунікаційній діяльності підприємств» (№ держреєстрації 0116U002394, 2018-2022 рр.).

Апробація результатів дослідження: Публікація тез на тему «Прикладне використання мурашиного алгоритму на прикладі побудови маршруту використовуючи мову програмування Java».

Структура кваліфікаційної роботи: Робота складається із вступу, трьох розділів та висновків до них, списку використаних джерел та одного додатку.

У першому розділі проведено огляд предметної області та додатків що дозволяють вирішити поставлену задачу.

У другому розділі проаналізовано перспективність даної системи, також наведено модель автоматизованої системи агрегації та побудови туристичних маршрутів.

У третьому розділі наведено опис користувацького інтерфейсу, а також особливості налаштування технологій.

Кваліфікаційна робота включає в себе 77 сторінок, 41 рисунок і список літератури із 49 джерел.

РОЗДІЛ 1

АНАЛІЗ ОСОБЛИВОСТЕЙ АВТОМАТИЗОВАНОЇ СИСТЕМИ АГРЕГАЦІЇ ТА ПОБУДОВИ ТУРИСТИЧНИХ МАРШРУТІВ

1.1 Аналіз предметної області

Туризм – це тимчасова діяльність людей по проведенню часу поза домом для відпочинку або ділових чи інших цілей[1]. Туризм має доволі довгу історію розвитку, беручи свій початок із класичної античності.

Туризм відрізняється від екскурсій тим, що туристи йдуть «протоптаною дорогою», отримуючи переваги від створених систем обслуговування та залишаються ізольовані від труднощів та небезпек під час проходження маршруту. Наслідком цього є додаткові надходження коштів до бюджету країни.

Сучасний туризм — це комерційно організований вид діяльності, який орієнтується на бізнес. За рахунок тривалої історії розвитку існує доволі велика кількість класифікацій туризму. Розглянемо класифікації згідно із ст. 4 Закону України «Про туризм»[2]:

- Дитячий туризм – вид туризму, який має на меті забезпечити дозвілля дитині, проте чіткі вікові межі відсутні. До цього типу відносяться екологічні табори, різні освітні центри та різновиди активного туризму.
- Молодіжний туризм – вид туризму, який розподіляється за віковою ознакою і згідно із міжнародними категоріями сюди відноситься молодь віком від 18 до 35 років.
- Сімейний туризм – вид туризму, який передбачає відпочинок сім’єю(з дітьми та без них). Головно метою є створення умов відпочинку для сприяння налагодження стосунків та благополуччя родини.
- Туризм для людей похилого віку – вид туризму, який здійснюється за віковим розподілом і в переважні більшості включає в себе ряд рекреаційних заходів.

- Культурно-пізнавальний туризм – вид туризму, що включає в себе обов'язкові відвідування історичних та культурних пам'яток. Головною метою є ознайомлення із культурним надбанням нації чи етносу.
- Лікувально-оздоровчий туризм – вид туризму, який є одним із найдавніших та найпопулярніших у світі. Включає в себе цілий перелік лікувальних заходів, які спрямовані на поліпшення здоров'я.
- Спортивний туризм – вид туризму, який відноситься до неолімпійських та включає в себе подолання певних маршрутів які можуть містити ряд перешкод(річкові пороги, мости, греблі, шлюзи, гірські перевали та ін.).
- Релігійний туризм – вид туризму, що передбачає собою мандрування вірян до святих місць по заданому маршруту.
- Екологічний туризм – вид туризму, який включає в себе подорожі до недоторканих природних територій або територій на які людство здійснило мінімальний вплив.
- Сільський туризм - проживання у сільській місцевості за рахунок оренди житла. Також може включати в себе поєднання роботи на сільськогосподарських угіддях та відпочинком.
- Підводний туризм – вид туризму, який передбачає плавання під водою із використанням спеціального спорядження.
- Гірський туризм – вид туризму, який базується на пересуванні людини або групи людей заданим маршрутом у гірській місцевості
- Пригодницький туризм – вид туризму, який має на меті відвідування екзотичних місць, при цьому використовуючи відмінні від популярних маршрути. Також до цього виду ми можемо включати пересування нетрадиційними транспортними засобами.
- Мисливський туризм – вид туризму, що включає в себе полювання на тварин, при цьому пересування відбувається по підготовленим і зарання випробуваним шляхам.
- Автомобільний туризм – вид туризму, основу якого складає пересування автомобілем місцевістю, яка відрізняється від постійного місця

проживання, при цьому автомобіль може бути приватною власністю або ж орендованим.

- Самодіяльний туризм – вид туризму, який передбачає самостійне планування всієї подорожі та маршруту без залучення послуг туроператорів.
- Рекреаційний туризм – вид туризму, який забезпечує відновлення екзистенційних потреб людини. Також має певні схожості із лікувально-оздоровчим туризмом, проте включає більший спектр послуг.
- Внутрішній туризм – вид туризму, що передбачає подорож громадян певної країни, не залишаючи її межі. Є надмножиною більшості видів туризму.
- Зовнішній туризм – вид туризму, при якому громадяни певної країни перетинають державний кордон. Як і внутрішній туризм, даний тип туризму є надмножиною більшості видів туризму, перерахованих у попередніх пунктах.
- Соціальний туризм – вид туризму, який передбачає виділення коштів із державного бюджету для фінансування подорожі. Сама мета подорожі може варіюватись в залежності від кожного конкретного випадку.
- Комерційний туризм – вид туризму, який базується на отриманні економічної вигоди туристичними агенціями.
- Сезонний туризм – вид туризму, основу якого складають відвідування певних місцевостей або проходження маршруту у певну пору року.

Переглянувши основні типи маршрутів, ми можемо прийти до висновку що це досить поширена та розвинена галузь у світі. Базуючись на вищевказаних видах туризму буде розроблено відповідно автоматизовану систему агрегації та побудови маршрутів. Дана система буде передбачати максимально можливе охоплення усіх особливостей кожного виду туризму.

1.2 Поняття автоматизованої системи побудови та агрегації

Автоматизована система – це програмне або апаратне забезпечення, яке передбачає стабільне функціонування без ручного втручання[3]. Усі бізнес-процеси повністю автоматизовані за допомогою апаратної частини або програмного коду.

Кожна автоматизована система складається із операцій. Операції – це сукупність апаратного та програмного забезпечення яке дозволяє функціонувати коректно без надання вхідних даних чи інструкцій.

Переваги автоматизованих систем[4]:

- Зменшує ризик користувацьких помилок;
- Забезпечує високу продуктивність системи;
- Гарантує коректне виконання стандартизованих операцій.

Недоліки автоматизованих систем[4]:

- Обмежена функціональність для користувача;
- Початкова вартість системи;
- Відсутність можливості коригування інструкцій чи вхідних даних.

Враховуючи усі переваги та недоліки, можна стверджувати що автоматизована система дозволяє здійснити економію трудовитрат та трудових ресурсів, підвищити точність виконання однотипних задач, що в свою чергу забезпечить надійність і прогнозованість у наданні певних послуг.

Якщо ми розглядаємо автоматизовану систему у контексті побудови та агрегації маршрутів, то можемо виокремити певні операції:

- Автоматична побудова маршрутів за певними категоріальними ознаками;
- Автоматична побудова маршрутів із урахуванням часу, який турист може витратити протягом дня;
- Агрегація відгуків користувачів на побудовані маршрути, для подальшого аналізу і оптимізації системи;

- Покрокова агрегація усього процесу побудови маршруту, для подальшого аналізу та оптимізації часу роботи системи;
- Агрегація маршрутів які були запропоновані користувачами;
- Перегляд та стандартизація усіх агрегованих даних;
- Визначення та заповнення пустих полів.

За рахунок усіх попередніх операцій ми можемо гарантувати створення єдиного уніфікованого набору даних. Даний набір даних ми можемо використати для подальшого аналізу і розвитку системи. Зокрема, на основі цих даних ми можемо покращити алгоритм побудови, час роботи алгоритму або ж час відгуку системи.

1.3.1 Огляд та аналіз веб-додатку Google Поїздки

Google поїздки – це web-служба яка дозволяє планувати подорожі, розроблена компанією Google(рис. 1.1.). Окрім web-версії, є доступні додатки для операційних систем Android та IOS. Також, за допомогою даного ресурсу ми можемо отримати дані про можливе бронювання авіаквитків, готелів та ресторанів, оренду житлового приміщення чи автомобіля.

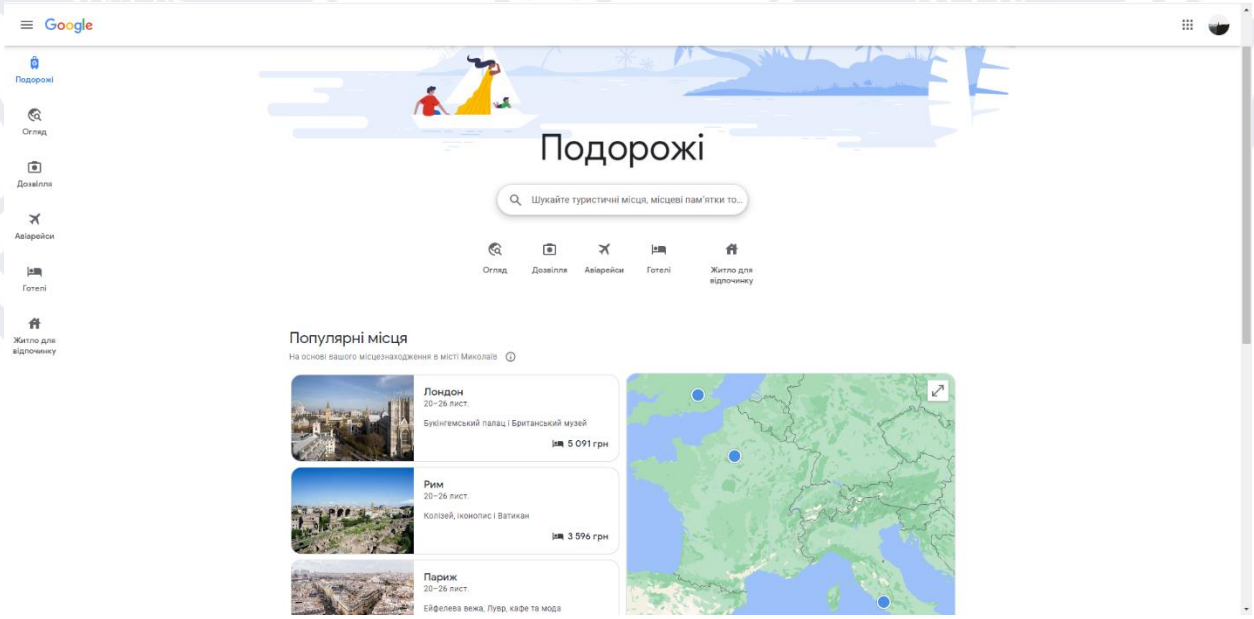


Рисунок 1.1 – Домашня сторінка веб-додатку Google поїздки[5]

Дана система здійснює агрегацію даних про попередньо згенеровані рекомендації для поїздки та відгуки користувачів залишені на кожний запропонований варіант(рис. 1.2). На основі цих даних відбувається подальша побудова майбутньої поїздки, базуючись на вподобаннях користувача. Також кожний користувач має змогу переглянути усі дані, що були агреговані системою.

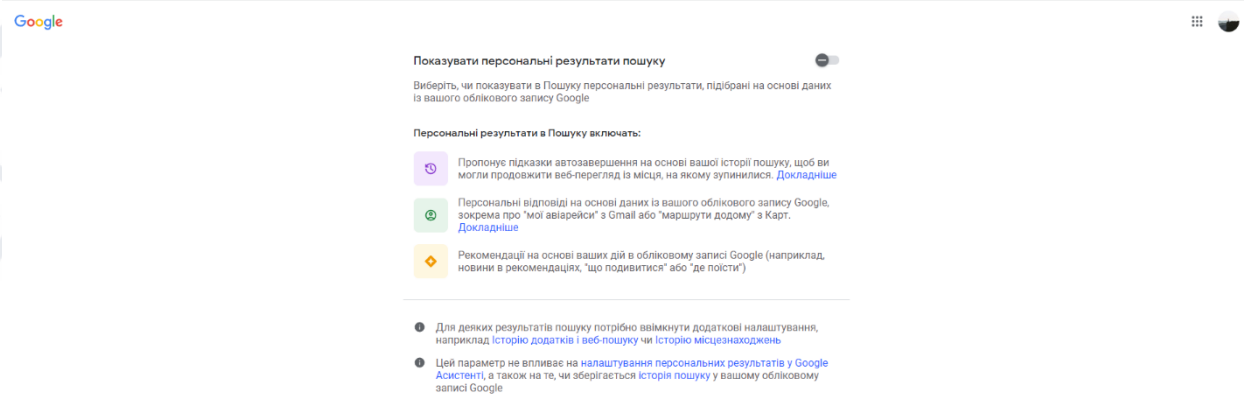


Рисунок 1.2 – Персоналізація пошуку для систем Google[5]

Варто зазначити, що важливою складовою даної системи є її інтеграція із пошуковою системою компанії Google(рис. 1.3). Даний підхід дозволяє використовувати один користувацький аккаунт задля агрегації даних про можливі вподобання користувача(використовуючи історію пошукових запитів).

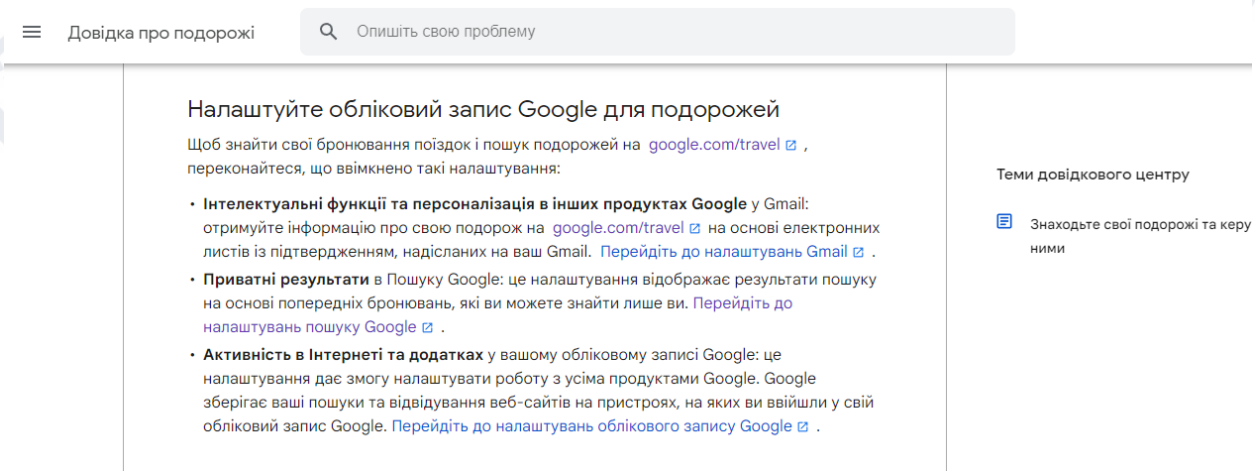


Рисунок 1.3 – налаштування персоналізації даних у Google аккаунті[5]

Важливою особливістю є автоматичне бронювання(рис. 1.4), що дозволяє за рахунок однієї системи повністю спланувати майбутню подорож і при цьому не використовувати інші системи бронювання. Також є можливість відредагувати свою подорож або ж скасувати бронювання.

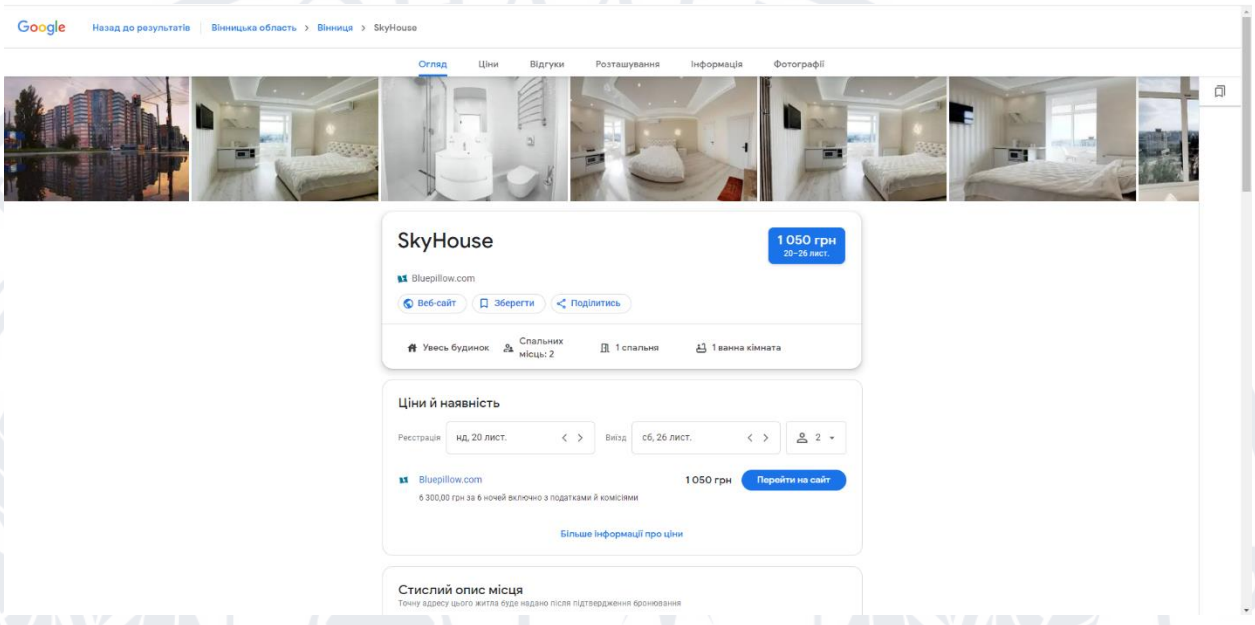


Рисунок 1.4 – бронювання готелю у додатку Google поїздки[5]

На основі вищеперерахованих особливостей ми можемо виділити наступні переваги:

- Зручний та продуманий користувацький інтерфейс;
- Можливість бронювання квитків, авто чи місця проживання;
- Інтеграція із пошуковою системою Google;
- Інтеграція із картами Google;
- Можливість коригувати маршрут;
- Можливість редагування або скасування бронювання;
- Можливість додати власні примітки до подорожі;
- Можливість залишити відгук по згенерованому маршруту;
- Можливість поширити власну подорож.

До недоліків ми можемо віднести наступне:

- Відсутня можливість планування подорожі враховуючи доступний вільний час;

- Відсутня можливість коригування маршруту в залежності від типу транспорту, відмінного від запропонованих(катер і т. д.);
- Не враховується сезонність маршруту;
- Не враховується привабливість маршруту;
- Відсутній перелік маршрутів які були запропоновані іншими користувачами;
- Збір користувацьких пошукових даних.

1.3.2 Огляд та аналіз веб-додатку Аналіз веб-додатку Wishtrip

Wishtrip – це web-служба, головна задача якої є надання користувачам даних про можливі маршрути. Власне, компанія вказує своєю головною метою забезпечення користувачів коректними та передбачуваними маршрутами для подорожей.

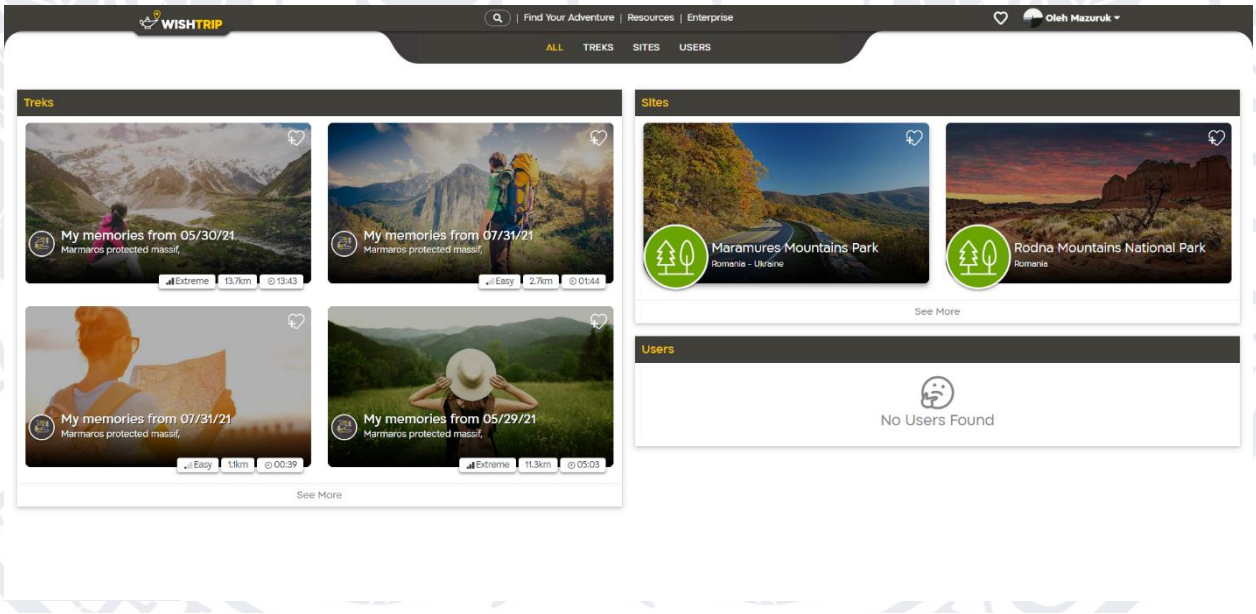


Рисунок 1.5 – Домашня сторінка веб-додатку Wishtirp[6]

Даний сервіс дозволяє доволі просто знаходити готовий маршрут, який був створений іншим користувачем(рис. 1.5), переглядати місцезнаходження культурних та історичних пам’яток в певному регіоні, орієнтуватись на незнайомій території, а також створювати власні маршрути які дозволять поділитись досвідом із іншими користувачами(рис. 1.6). Під власним

маршрутом мається на увазі покрокова інструкція із підкріпленими фото та відео, щоб інші користувачі доволі просто могли це використати у власній подорожі.

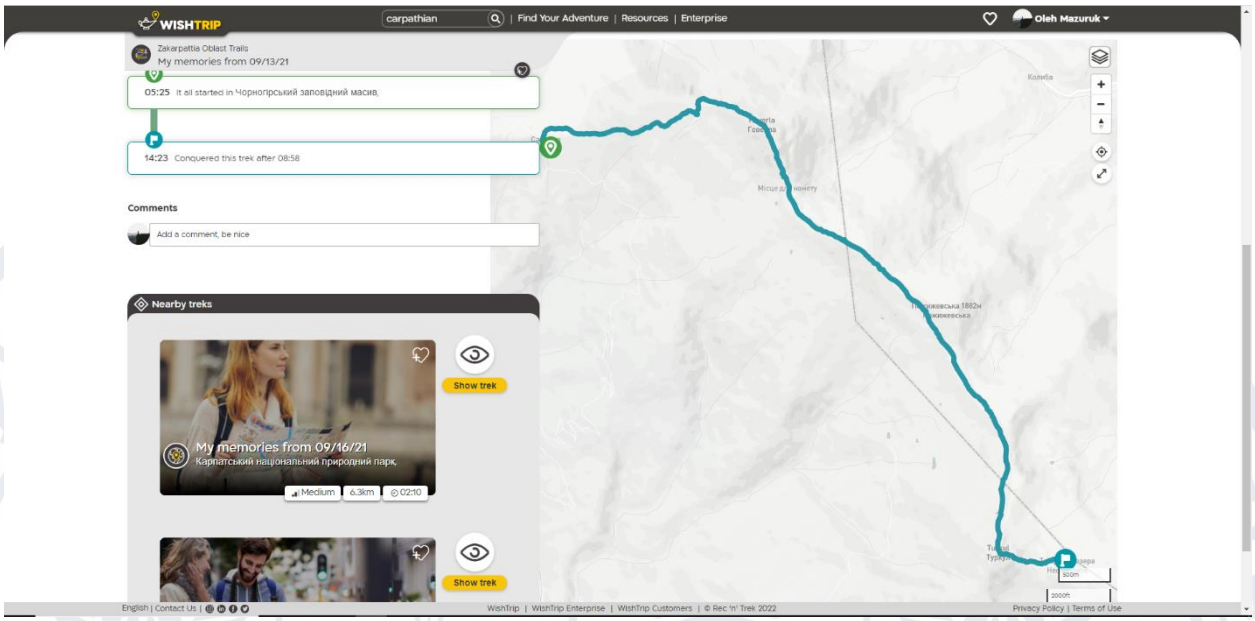


Рисунок 1.6 – Готовий маршрут[6]

Також, платформа Wishtrip надає доволі цікаву особливість – конструктор ігор. В описі даного додатку зазначено, що дана особливість реалізується зо допомогою модуля Game Builder. При створенні маршруту, користувач може створити квест, який буде запропоновано іншим користувачам. Знайти таке запрошення у гру доволі просто – біля маршруту буде відмітка із зображенням гри. Дана функція є доволі цікавою для популяризації відвідування певних місць на локації.

Окрім вищеперерахованих функцій, варто згадати про сувенірні альбоми. Дана функція дозволяє користувача генерувати електронну версію альбому, яка буде по завершенню проходження маршруту - агрегувати дані. На виході користувач отримує поєднання фото та відео із даними по маршруту(висота, погода, час і т.д.).

Якщо ж ми розглядаємо реалізовану функціональність даної платформи для бронювання квитків, авто чи місця проживання – можемо стверджувати, що дана функція не набула високої популярності та не була відлагоджена у повній мірі.

На основі вищеперерахованих особливостей ми можемо виділити наступні переваги:

- Зручний користувацький інтерфейс;
- Агрегування маршрутів запропонованих користувачами;
- Можливість додати власні примітки до подорожі;
- Можливість залишити відгук по згенерованому маршруту;
- Генерація електронного альбому;
- Конструктор квестових ігор;
- Можливість поширити власну подорож.

До недоліків ми можемо віднести наступне:

- Відсутня можливість планування подорожі враховуючи доступний вільний час;
- Відсутня можливість коригування маршруту в залежності від типу транспорту, відмінного від запропонованих(катер і т. д.);
- Не враховується сезонність маршруту;
- Не враховується привабливість маршруту;
- Відсутність можливість бронювання;
- Відсутність автоматична генерація маршруту;
- Відсутність інтеграції із іншими системами.

1.3.3 Огляд та аналіз веб-додатку Triplt

Tripit – це web-служба, яка дозволяє користувачам створювати власні маршрути, що базуються на отриманому досвіді. Також дана система дозволяє агрегувати та поширювати усі створені маршрути.

На відміну від попередніх систем, дана система надає дуже незначний функціонал для користувачів. В даній системі присутня лише функціональність для створення маршрутів користувачами та їх агрегація(рис. 1.7). Єдиною відмінністю є власний блог, що ведеться даною компанією і містить статті що носять рекомендаційний характер для туристів.

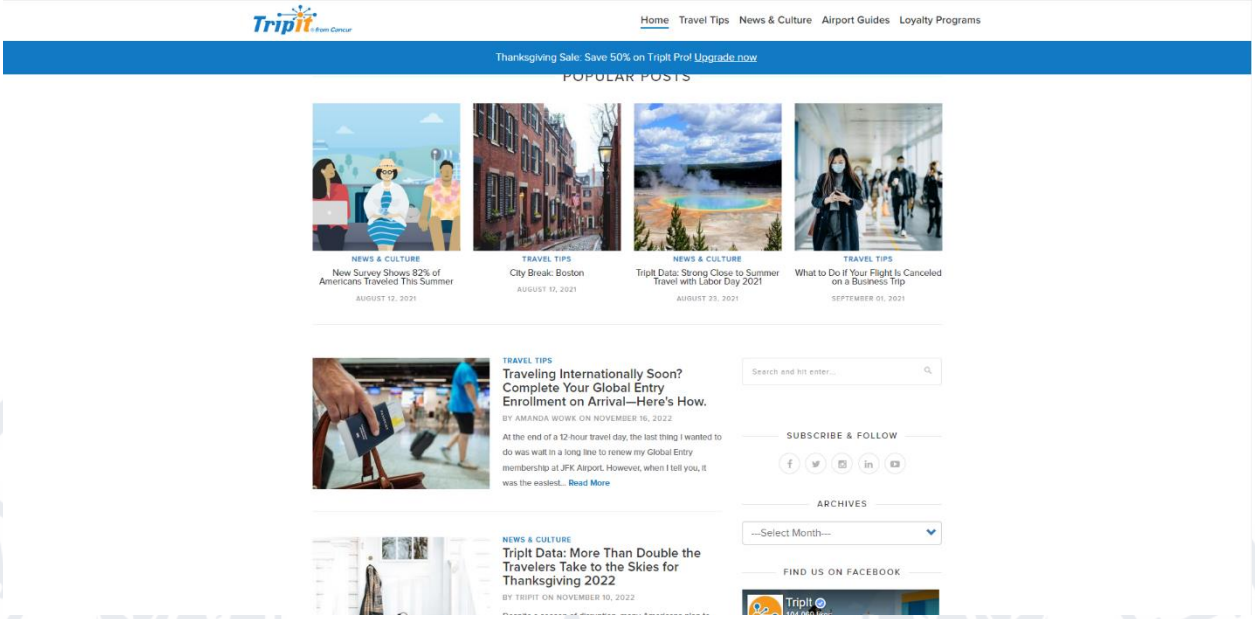


Рисунок 1.7 – Домашня сторінка веб-додаткуTriplt[7]

На основі вищеперерахованих особливостей ми можемо виділити наступні переваги:

- Агрегування маршрутів запропонованих користувачами;
- Можливість додати власні примітки до подорожі;
- Власний блог;
- Можливість поширити власну подорож.

До недоліків ми можемо віднести наступне:

- Відсутня можливість планування подорожі враховуючи доступний вільний час;
- Відсутня можливість коригування маршруту в залежності від типу транспорту, відмінного від запропонованих(катер і т. д.);
- Не враховується сезонність маршруту;
- Не враховується привабливість маршруту;
- Відсутність можливість бронювання;
- Відсутність автоматична генерація маршруту;
- Відсутність інтеграції із іншими системами;
- Незручний користувацький інтерфейс;
- Повільна робота web-додатку;
- Відсутність можливості залишити відгук на маршрут.

1.3.4 Огляд та аналіз веб-додатку Sygic Travel Trip Planner

Sygic Travel Trip Planner – це web-служба, що надає можливість користувачам переглядати визначні історичні та культурні місця, варіанти місць що хотів би відвідати користувач(парки, кафе, ресторани). Також дана система дозволяє агрегувати та поширювати усі створені маршрути(рис. 1.8).

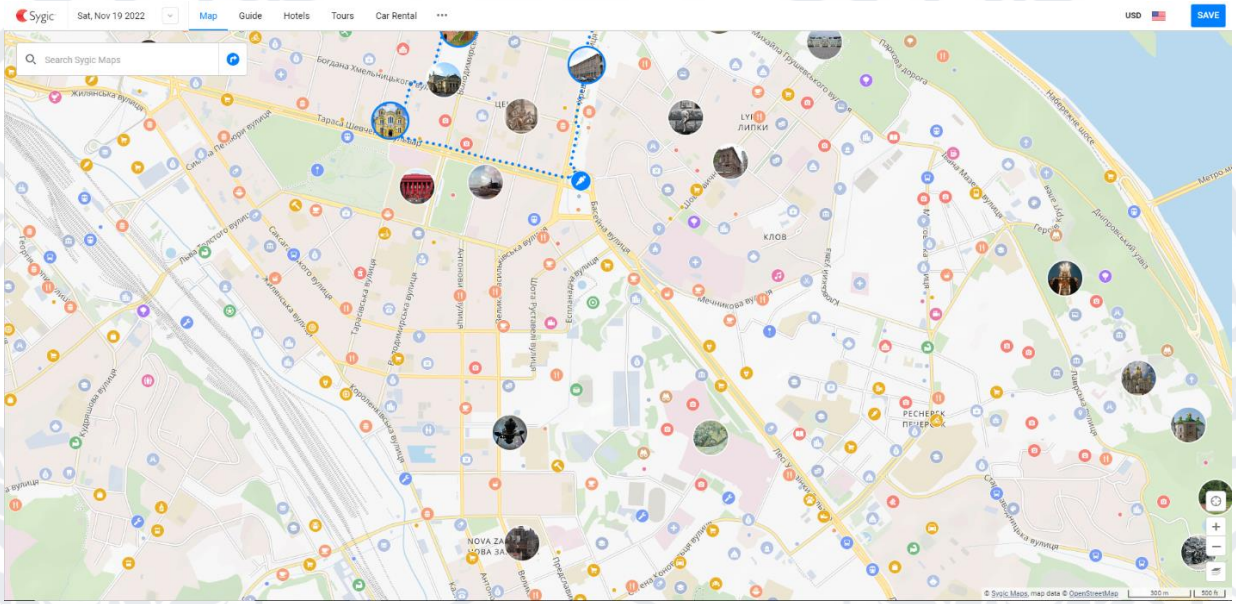


Рисунок 1.8 – Домашня сторінка веб-додатку Sygic Travel Trip Planner[8]

Дана система має функціональність автоматичної побудови маршруту із урахуванням орієнтованого часу подорожі(рис. 1.9). Проте, базуючись на відгуках користувачів, ми можемо стверджувати що дана система не завжди коректно складає маршрут із урахуванням часу.

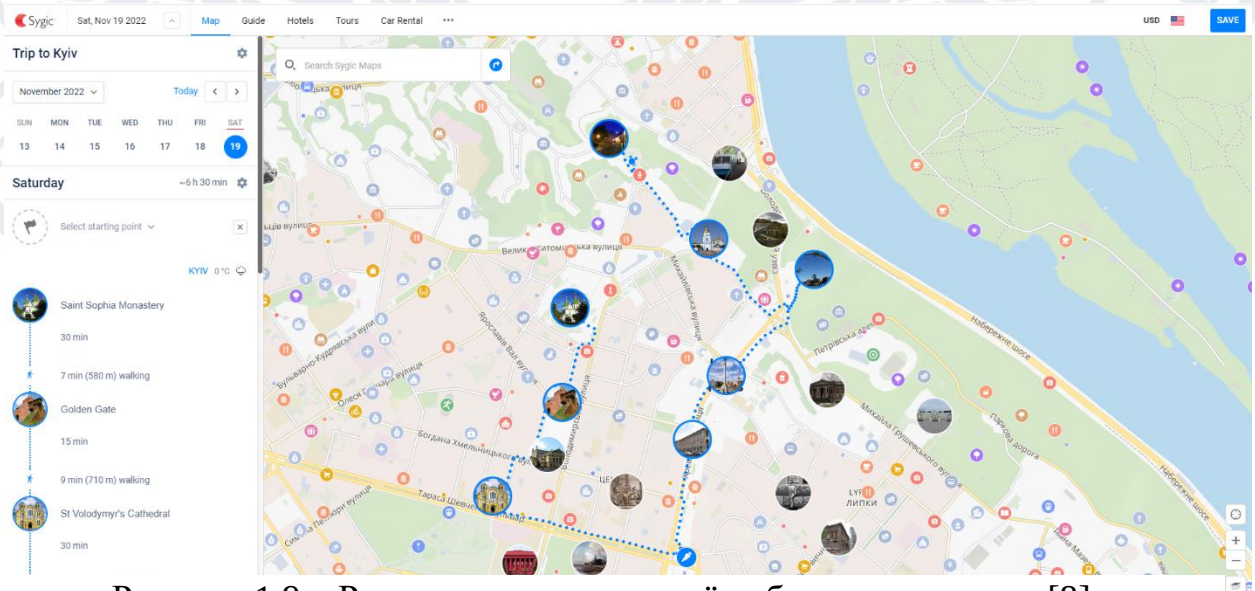


Рисунок 1.9 – Результат автоматичної побудови маршруту[8]

Також, дана система має функціонал для детального огляду деяких найпопулярніших туристичних напрямків у форматі 360°. Ще варто зазначити можливість використання додатку у офлайн режимі, можливість бронювання авіаквитків, авто та місць проживання використовуючи вбудований функціонал та можливість корегування маршруту.

На основі вищеперерахованих особливостей ми можемо виділити наступні переваги:

- Агрегування маршрутів запропонованих користувачами;
- Урахування орієнтованого часу подорожі при побудові маршруту;
- Можливість використання офлайн;
- Можливість додати власні примітки до подорожі;
- Власний блог;
- Можливість поширити власну подорож.

До недоліків ми можемо віднести наступне:

- Відсутня можливість коригування маршруту в залежності від типу транспорту, відмінного від запропонованих(катер і т. д.);
- Не враховується сезонність маршруту;
- Не враховується привабливість маршруту;
- Відсутній блог;
- Проблематичне управління орієнтованого часу подорожі.

Отже, ми можемо звести усі порівняльні критерії у одну загальну таблицю:

Таблиця 1.1 – порівняльна таблиця аналогів за визначеними критеріями.

Критерії	Google Trips	WishTrip	Triplt	Sygic
Зручний дизайн	✅	✅	❌	✅
Автоматична побудова маршруту	✅	❌	❌	✅

Продовження таблиці 1.1

Агрегація маршрутів	✗	✓	✓	✓
Відгуки користувачів	✓	✗	✗	✓
Урахування орієнтованого часу подорожі	✗	✗	✗	✓
Інтеграція з іншими системами	✓	✗	✗	✓
Урахування категоріальних ознак маршруту	✗	✗	✗	✗
Можливість бронювання	✓	✗	✗	✓
Власний блог	✗	✗	✓	✗

1.4 Висновки до розділу 1

У даному розділі було здійснено аналіз предметної області, розглянуто поняття автоматизованої системи побудови та агрегації, а також проведено аналіз існуючих аналогів. Наступний розділ буде присвячений моделюванню автоматизованої системи агрегації та побудови маршрутів, що включає в себе формування загальних вимог до архітектури системи, функціональних вимог, бази даних, а також буде обґрунтувано обрані технології.

РОЗДІЛ 2
МОДЕЛЮВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ АГРЕГАЦІЇ ТА ПОБУДОВИ ТУРИСТИЧНИХ МАРШРУТІВ

2.1 Перспективність автоматизованої системи агрегації та побудови туристичних маршрутів

Задля більш детального огляду перспективності даної системи, варто здійснити аналіз туризму в світі та Україні. Оскільки туризм є важливою складовою економічного сектору – обґрунтування перспективності системи також будуть включати даний фактор.

Більшість статистичних даних про загальні тенденції туризму частіше базуються на даних із банківських транзакцій, рідше на даних із авіакомпаній або опитувань населення. Саме тому, на даний момент можна достатньо просто знайти коректні статистичні дані і провести максимально достовірний у своїх висновках аналіз даної предметної області.

Також, даний аналіз допоможе визначитись із необхідною функціональністю системи та дозволить оптимізувати певні маркетингові питання. В результаті це дозволить визначити на чому слід зосередити основні зусилля при розробці системи.

Варто зазначити, що аналіз туризму буде розглядатись у періоді від 1950 року до 2019 року. Не будуть розглядати дані за 2020-2022 роки включно, оскільки в 2020 році розпочалась пандемія Covid-19, що негативно вплинула на туристичний сектор, у 2021 році відбувались стабілізаційні заходи щодо пандемії, а у 2022 році, на додачу до пандемії, відбулось повномасштабне вторгнення росії в Україну, що призвело до дестабілізації світового порядку.

В першу чергу буде розглянуто світовий туризм. Згідно із даними ourworldindata, які у свою чергу посилаються на дані всесвітньої туристичної організації ООН(ЮНВТО), у 1950 році кількість міжнародних туристів

становила близько 25 мільйонів. Через 69 років кількість міжнародних туристів зросла до 1,4 мільярдів.

Можна констатувати, що кількість туристів зросла у 56 разів. Це спричинено швидкою глобалізацією світу, яка зумовила потребу в зростаючій кількості міжнародних контактів.

Аналізуючи графік на рисунку 2.1, можемо спостерігати зростаючу тенденцію відвідування країн Азії та Африки. На противагу цьому популярність країн Америки та Європи зменшується, хоч динаміка цих змін не є швидкою, однак чітко прослідковується.

Якщо ми розглядаємо динаміку змін кількості туристів протягом календарного року, можемо спостерігати що відбувається постійний приріст(рис. 2.1). Тому, у світовому туризмі розробка даної системи може виявитись досить перспективною і затребуваною.

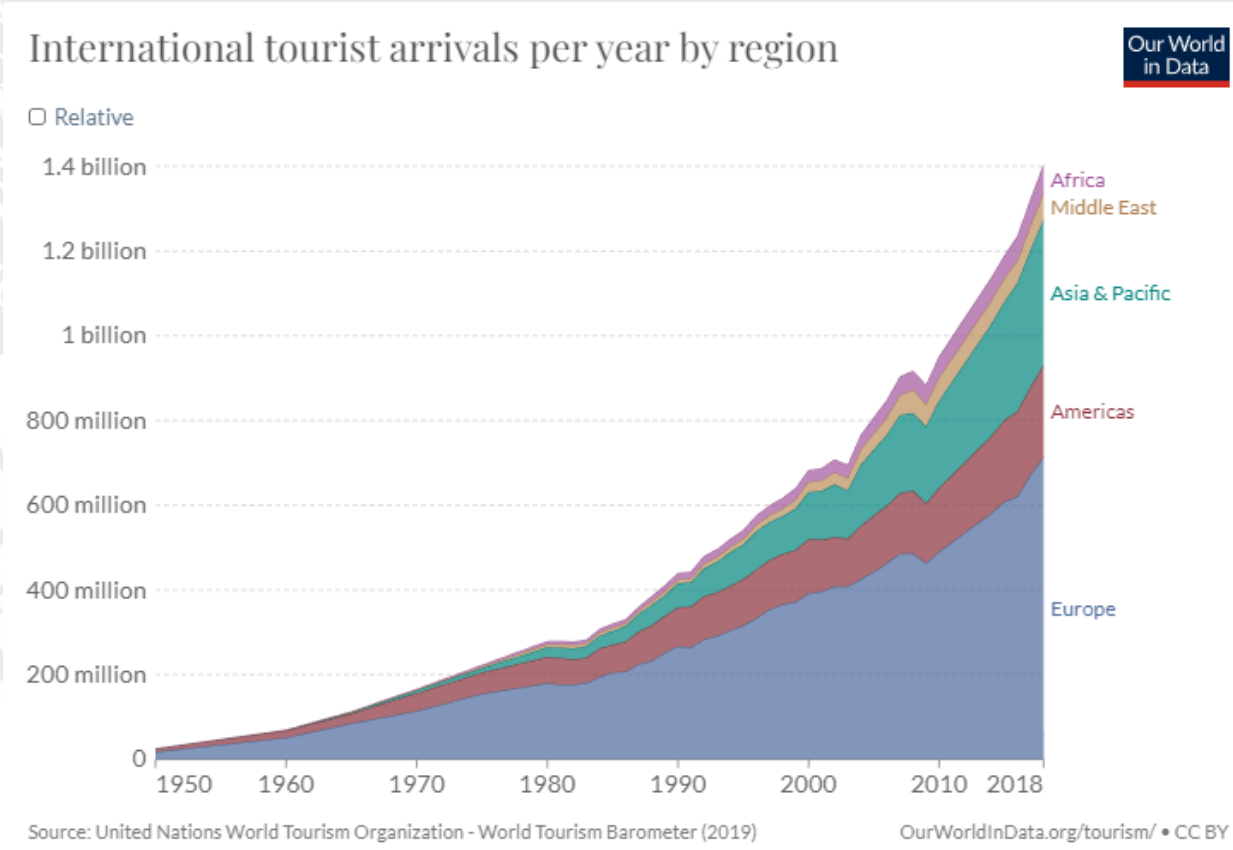


Рисунок 2.1 – Графік кількості туристів у світі, в період із 1950 р. по 2019 р.[9]

Розглянемо діаграму що відображає кількість туристів які прибули до України в період із 1995 року по 2019 рік. В контексті України, ми не можемо

проаналізувати дані в період із 1950 року по 1991 рік, оскільки на той час країна перебувала у окупації радянським союзом. А в період із 1991 року по 1995 рік дані були відсутні або не достовірні.

Згідно із даними ourworldindata, станом на 1995 рік, кількість туристів складала приблизно 6 мільйонів. Через 14 років кількість міжнародних туристів зросла до 13 мільйонів[10]. Звичайно, ми можемо спостерігати доволі незначний приріст, у порівнянні із іншими країнами та світовими тенденціями, проте цьому посприяли ряд факторів, таки як узурпація влади злочинним угрупованням у 2010 році, а також доволі значний спад у 2013-2014 роках, спричинений окупаційними діями росії у відповідь на Революцію Гідності. Дані події спричинили дестабілізацію політичних процесів, що в свою чергу посилили спад туризму в Україні.

Проте, якщо ми розглядаємо популяризацію туризму у період від 2000 року до 2009 року – вона цілковито збігається із світовою. Саме тому ми можемо спрогнозувати кратне зростання кількості туристів в Україні, після її перемоги у війні.

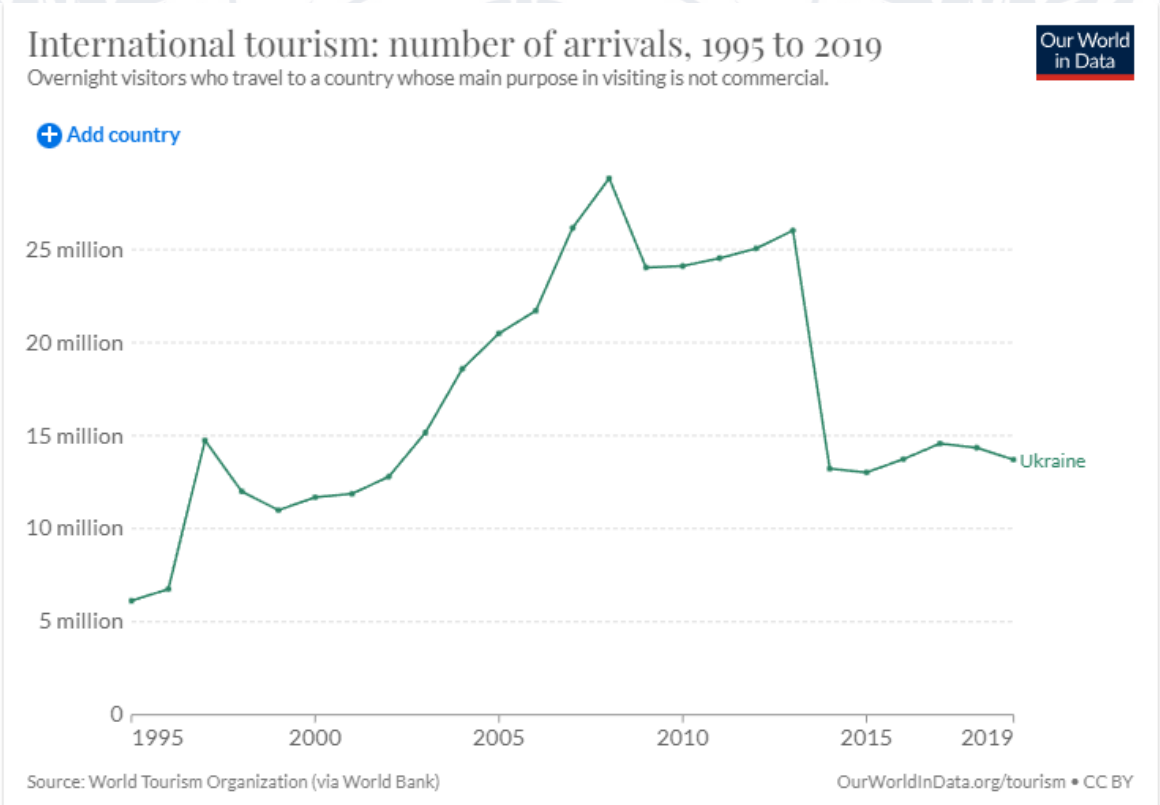


Рисунок 2.2 – Графік кількості туристів в Україні, у період із 1995 р. по 2019 р.[9]

Тому розробка даної системи може виявитись досить корисною для туристів, які матимуть бажання відвідати Україну.

Ще одним важливим фактором, який варто розглянути – це країни, які приймають найбільшу кількість туристів. Даний фактор є досить важливим, в контексті визначення найбільш перспективних напрямків розвитку системи. Під розвитком системи мається на увазі наповнення початкових даних про країну, таких як: історичні, визначні та найбільш відвідувані місця, користувацькі маршрути і додавання критеріїв(сезонність, тип пересування та ін.) до місць та маршрутів.

З огляду на рисунок 2.3, ми можемо виокремити Канаду, США, Китай та країни-члени Європейського Союзу(ЄС). Саме тому початковий набір даних для системи варто сформувати враховуючи найбільш-популярні для відвідування країни.

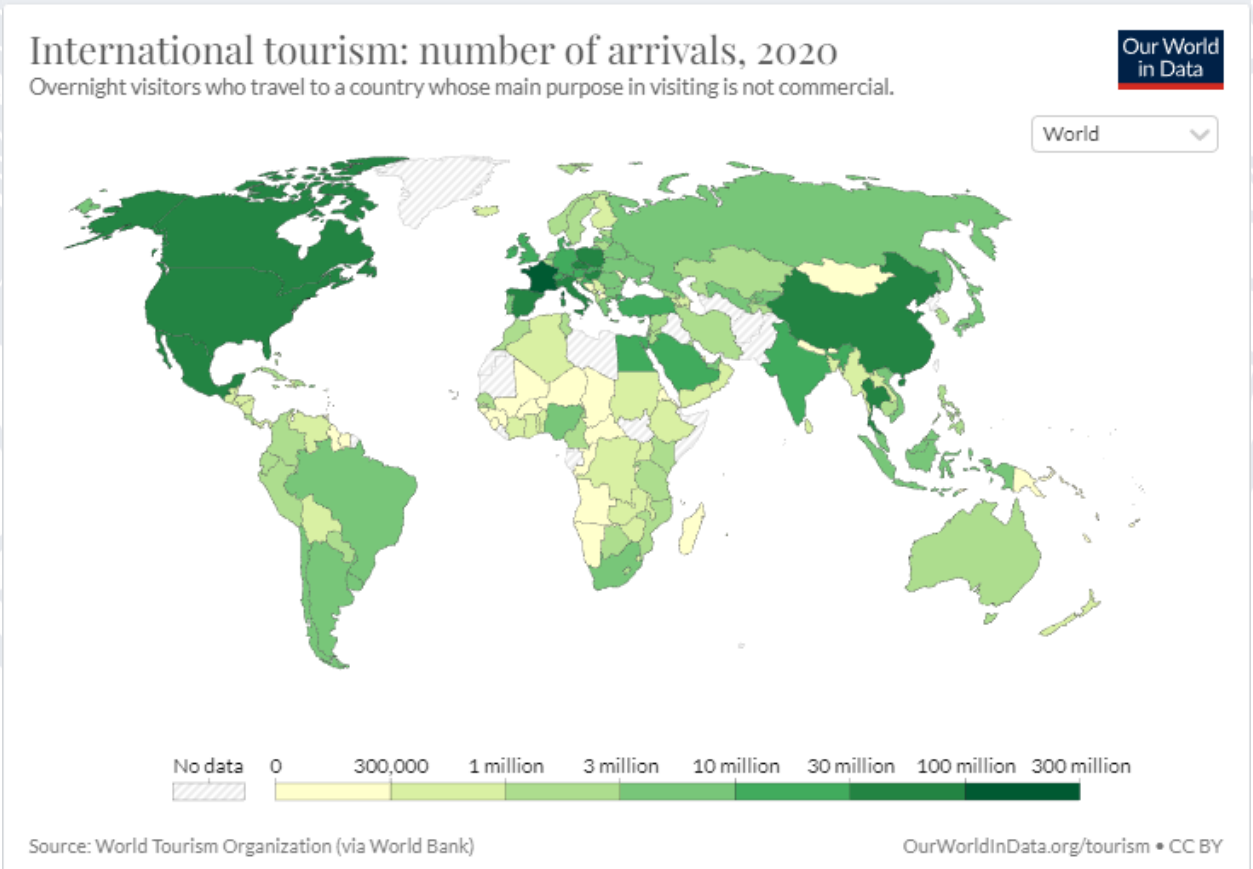


Рисунок 2.3 – Карта розподілу кількості туристів за країнами(2020 рік)[9]

2.2 Визначення функціональних вимог

Необхідно розробити систему для автоматизованої системи агрегації і побудови туристичних маршрутів, яка відповідатиме наступним функціональним вимогам:

- Здійснення авторизації та автентифікація користувача;
- Можливість відновлення аккаунту, якщо пароль був втрачений;
- Наявність функції «Запам’ятати мене», яка забезпечує довший час роботи у системі без повторної автентифікації;
- Можливість прив’язати аккаунт користувача до електронної пошти або номеру телефону, з метою можливого відновлення аккаунту у майбутньому;
- Можливість зміни електронної пошти чи номеру телефону, що були прив’язані до аккаунту;
- Можливість заміни паролю;
- Наявність перевірки надійності паролю при реєстрації аккаунту та при заміні;
- Наявність декількох мов у системі і можливість їх перемикати(обов’язковими є українська та англійська мови);
- Наявність анкети користувача із можливістю редагування, що містить дані про користувача, такі як: прізвище, ім’я, по-батькові, країна проживання;
- Наявність інструменту моніторингу споживання ресурсів системою, який здійснюватимуть моніторинг стану системи(кількість виділених потоків, споживання оперативної та периферійної пам’яті, кількість створених сутностей, навантаження на процесор, дані про кількість http процесів із розподілом по кодам http відповідей);
- Наявність інструменту моніторингу системних логів із можливістю очистки та збереження у певний файл, а також можливість перемикати рівень моніторингу(згідно стандарту RFC 5424);

- Наявність інструменту що дозволить здійснювати управління користувачами, а саме: видалити, створити, відновити або призупини дію аккаунту, скинути пароль, переглянути останні дії користувача, призначити певну системну роль;
- Можливість створення туристичних маршрутів користувачами, які міститимуть усі необхідні дані, такі як: довжина, складність, тип пересування, тип сезону, привабливість, додаткові зображення та фото, детальну мапу подорожі, перелік можливих історичних та культурних місць;
- Можливість редагування створених маршрутів автором;
- Наявність автоматичної генерації туристичного маршруту;
- Генерація туристичного маршруту може включати початкові та кінцеві точки, тип пересування, тип сезону, враховувати орієнтований час подорожі, а також містити перелік місць, які користувач бажає відвідати;
- Якщо при генерації вказано лише країну чи місто, за замовчуванням використовувати наступні дані:
    - Тип сезону – поточний;
    - Перелік місць, які користувач бажає відвідати – найбільш популярні місця;
    - Тип пересування - ходьба;
    - Початкові та кінцеві точки маршруту – найбільш популярні місця;
    - Орієнтований час подорожі – 8 годин.
- Наявність автоматичної генерації для документації програмного коду та програмного інтерфейсу системи(API);
- Можливість ведення блогу для користувачів;
- Можливість поширювати маршрути та дописи у блогах у інших соціальних мережах;
- Можливість видалення аккаунту користувача;
- Можливість виходу із системи.

2.3 Визначення нефункціональних вимог

Необхідно розробити систему для автоматизованої системи агрегації і побудови туристичних маршрутів, яка відповідатиме наступним нефункціональним вимогам:

- Мінімальний обсяг оперативної пам’яті для запуску складає 6 ГБ;
- Мінімальний обсяг периферійної пам’яті для запуску складає 10 ГБ;
- Мінімальна кількість ядер процесора повинна становити 4, із тактовою частотою 2,2 МГц;
- Розгортання системи здійснити за допомогою контейнеризації, використовуючи хостингові сервіси;
- В якості основної операційної системи(ОС) на сервері використовувати Ubuntu версії 22.04;
- Доступ до серверу здійснювати виключно через особистий кабінет, який буде створений на хостинговому сервісі, а також використовуючи ssh протокол;
- Підключені до серверу по ssh здійснюється виключно за протоколом SSH-2;
- Авторизацію та автентифікацію системи здійснювати використовуючи JSON Web Token(JWT) та відповідно до стандарту RFC 7519;
- Для підпису токену використовувати приватний ключ, який згенерований алгоритмом RS256;
- Взаємодію між сервером та клієнтом здійснювати за допомогою протоколу HTTPS;
- Відкрити на сервері для прослуховування порт 443, а також встановити автоматичний редірект із 80 порту на 443;
- Використовуючи можливості контейнеризації, встановити автоматичне перезавантаження системи у випадку системної помилки, що призвела до зупинки роботи;

- Здійснити налаштування резервного копіювання даних в особистому кабінеті користувача на хостинговому сервісі, якщо дана функція не доступна – слід використати готові інструменти і налаштувати процесу автоматичного створення копій важливих файлів;
- Забезпечити моніторинг використання ресурсів серверу;
- Для контролю помилок у системі необхідно створити єдиний механізм обробки(рис. 2.4), який у свою чергу буде зберігати детальний опис помилок у певний файл, а для клієнту буде повернуто лише певний http-статус із помилкою виконання(наприклад 500-599) без опису помилки;

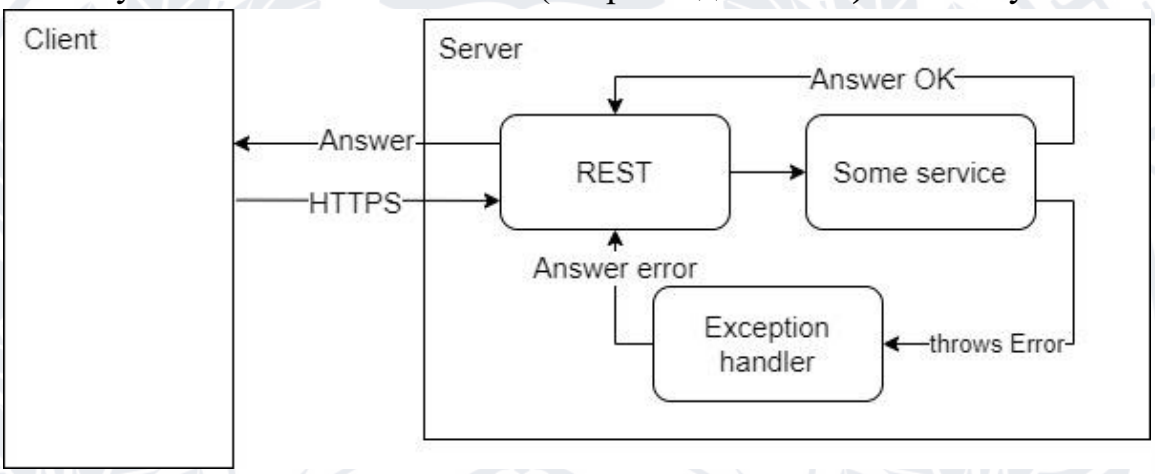


Рисунок 2.4 – Архітектура механізму обробки помилок

- При інтеграції інших систем, слід використовувати верифікацію користувача на основі JWT-токену. Також, як один із етапів верифікації, варто згенерувати публічний ключ та передати його системі що буде інтегрованою, за допомогою файлового носія або ж використовуючи REST-API;
- В якості інструменту контейнеризації варто обрати продукт, який буде відповідати наступним вимогам:
    - можливість створення приватних мереж;
    - можливість управління ресурсами для кожного контейнеру;
    - підтримка системи безперервної доставки та розгортання;
    - підтримка кластеризації системи.
- Для перевірки надійності системи та контролю додавання нових функцій – здійснити автоматизацію тестування.

2.4 Розподіл груп доступу та визначення типів користувачів

У системі автоматизованої агрегації і побудови туристичних маршрутів передбачаються різні дозволи та ролі для користувачів. Буде створено 5 основних ролей для користувачів. Також буде дозволений обмежений доступ для неавторизованих користувачів.

У таблиці нижче перераховані дозволи для кожної користувацької ролі:

Таблиця 2.1 – розподіл прав доступу користувачів згідно ролей.

Можливості	Guest	User	Moderator	Admin	Owner
Перегляд карти із створеними маршрутами	✅	✅	✅	✅	✅
Перегляд туристичного маршруту	✅	✅	✅	✅	✅
Генерація туристичних маршрутів	❌	✅	✅	✅	✅
Заповнення анкети	❌	✅	✅	✅	✅
Створення/редагування /видалення маршрутів	❌	✅	✅	✅	✅
Створення/редагування /видалення локацій	❌	✅	✅	✅	✅

Продовження таблиці 2.1

Створення/редагування /видалення коментарів	✗	✓	✓	✓	✓
Створення/редагування /видалення нових пропозицій	✗	✗	✓	✓	✓
Доступ до панелі адміністрування додатком	✗	✗	✗	✓	✓
Створення/редагування /видалення нових користувачів	✗	✗	✗	✓	✓
Можливість призначати інші ролі	✗	✗	✗	✗	✓
Додавання нових категоріальних ознак маршруту	✗	✗	✗	✗	✓

2.5 Моделювання бази даних

Моделювання бази даних – це процес створення віртуальної версії внутрішньої структури бази даних, або її частин для відображення зав'язків між структурами даних[12]. Головна мета даного процесу – це наочність даних, що дозволяє оптимізувати або ж реорганізувати структури даних.

Будь-який процес моделювання бази даних складається із декількох етапів і розпочинається створенням концептуальної моделі. Концептуальна модель має на меті відобразити початкову структуру та загальні вимоги до проекту.

Для системи автоматизованої агрегації і побудови туристичних маршрутів у концептуальні моделі важливим є визначення основних сутностей та їх зав’язків між собою. Основні сутності ми можемо виокремити здійснивши аналіз класифікації туризму, наведену в 1 розділі кваліфікаційної роботи.

В результаті аналізу, варто виокремити наступні сутності: Маршрут, Місце, Країна, Геодані, Тип сезону, Тип пересування, Відгук або коментар, Анкета користувача, Роль користувача, Прикріплені фото або відео. Графічно даний результат зображено на рисунку 2.5.

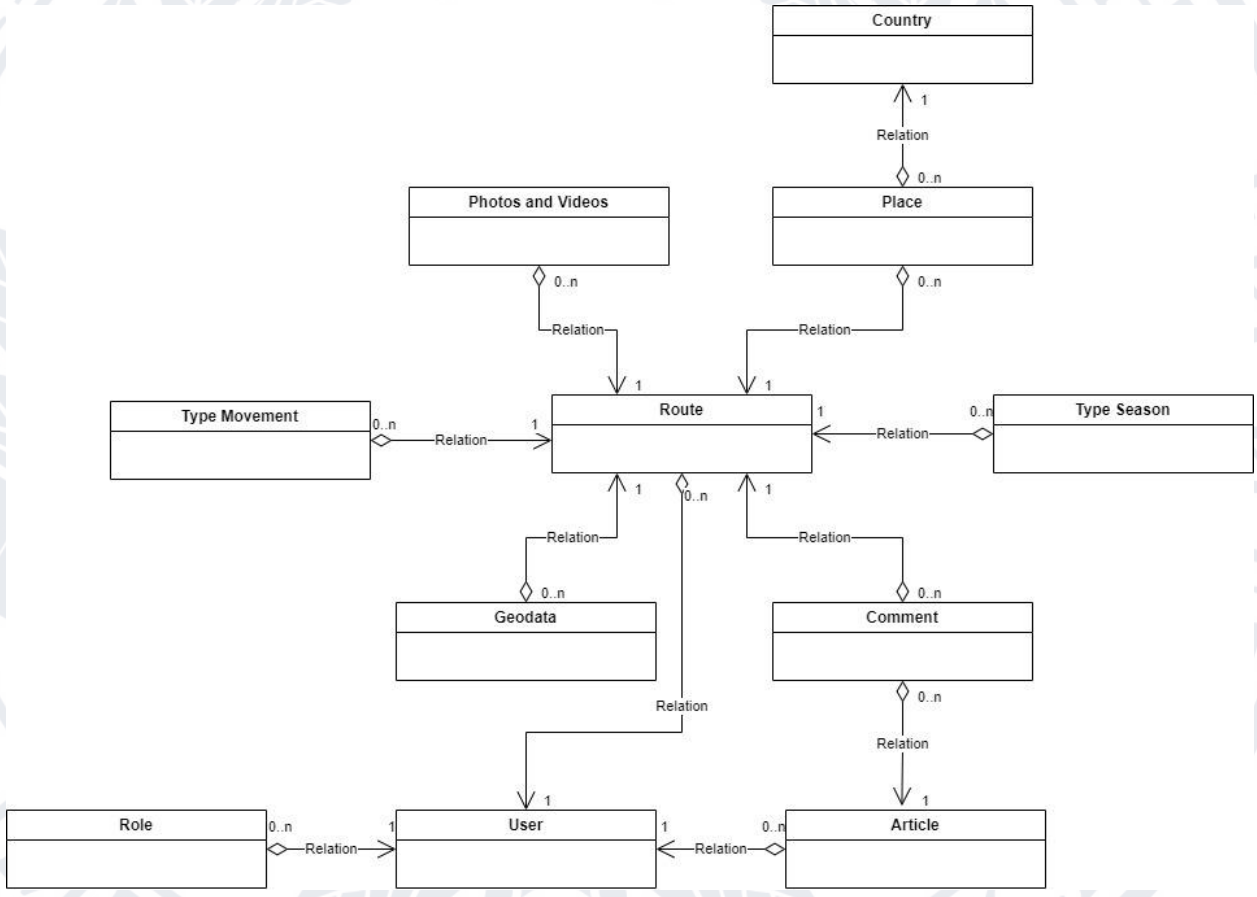


Рисунок 2.5 – Концептуальна модель сутностей

Наступним кроком буде побудова логічних та фізичних моделей даних. На мою думку, об’єднати дані кроки буде досить вдалим рішенням, для того щоб більш наочно продемонструвати готову модель бази даних. На даному етапі будуть додані атрибути даних до кожної сутності із вказаними типами даних та довжиною.

В результаті ми отримуємо схему, яку проілюстровано на Додатку А. Підведемо короткий підсумок по структурі бази даних:

- Структура бази даних розроблена відповідно до основних потреб туристів та з урахуванням класифікаційного розподілу;
- Здійснена нормалізація бази даних відповідно до першої нормальної форми, яка характеризується наявністю у кожній таблиці первинного ключа, уникненням неключових атрибутів та атомарністю даних;
- Здійснена нормалізація бази даних відповідно до другої нормально форми, яка відповідає вимогам першої нормальної форми та виокремленням в окрему таблицю даних що повторюються(наприклад – тип сезону);
- Здійснена нормалізація бази даних відповідно до третьої нормальної форми, яка відповідає вимогам другої нормальної форми та забезпечує винесення полів, що залежать від інших полів в окрему таблицю.

2.6 Архітектурне моделювання додатку

Архітектура – це набір зважених та раціональних рішень, які приймаються на ранніх етапах розробки проєкту, щодо виокремлення та структуризації компонентів системи[11]. Трудовитрати на майбутню розробку та підтримку системи буде визначати початкове архітектурне вирішення і чим більш вдалою буде отримана стартова реалізація – тим менше затрат буде у майбутньому. Головним критерієм вдалого архітектурного рішення буде виступати кількість трудозатрат, які необхідно витратити для додавання нового функціоналу до системи.

Оскільки автоматизована система агрегації та побудови туристичних маршрутів буде розроблено в якості web-додатку, тому в першу чергу буде використаний шаблон клієнт-сервер. Даний шаблон здійснює розподіл усіх компонентів взаємодії на клієнтські додатки та сервер[13]. Клієнтські додатки відображають користувацьку частину, а також здійснюють обмін даними із

сервером. Сервер здійснює обмін даних та може взаємодіяти із багатьма клієнтськими додатками.

Перевагою даного шаблону є те, що в якості клієнтського додатку ми можемо використовувати як web-додаток, так і мобільний або desktop-додаток, використовуючи єдиний API серверу, як зображено на рисунку 2.6.

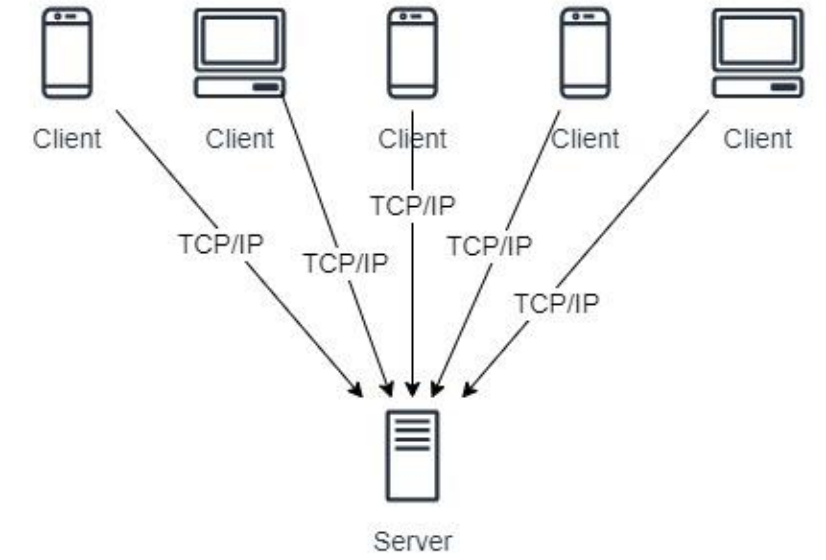


Рисунок 2.6 – Клієнт-серверна архітектура

Виконаємо проектування архітектури програми із урахуванням вимог архітектурного шаблону клієнт-сервер. В даному випадку, користувачеві буде надано інтерфейс, для взаємодії із серверною частиною. Оскільки система передбачає агрегацію даних, окрім обробки запитів, нам слід виокремлювати також сутності, що будуть відображати дані, які в подальшому будуть збережені до бази даних(БД). В даному випадку найбільш доцільним шаблоном є Model-View-ViewModel(MVVM).

MVVM – це шаблон проектування, який визначає принципи виокремлення та поділу функціональних частин системи[14]. У нашому випадку система буде розділена на три частини(рис. 2.7):

- View – це користувацький інтерфейс;
- ViewModel – це серверна частина системи, що дозволяє здійснювати обмін даними між клієнтом та сервером;
- Model – це шар, що відповідає за опис даних та взаємодію між системою та сховищем даних.

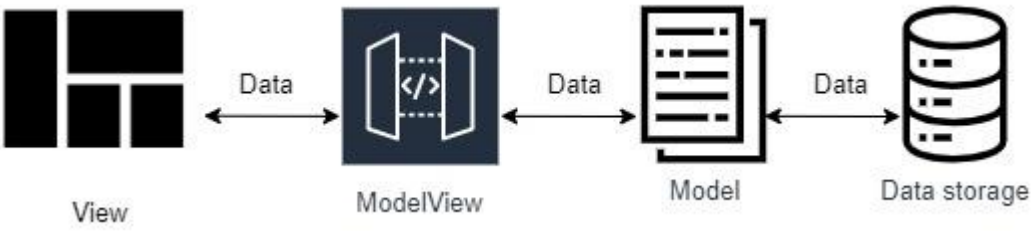


Рисунок 2.7 – MVVM архітектура[14]

Шар View буде відображати в окремий клієнтський додаток. Розглянемо детальніше ModelView та Model. На даній схемі два абстрактні шари, проте із чітко виокремленими обов'язками. Окрім того, що дані шари буде застосовано до серверної частини системи, до них буде застосовано декомпозицію. Шар ModelView буде розподілений на три додаткові шари:

- Controller – це шар що буде відповідати лише за обмін даними із клієнтським додатком та викликом шару Service.
- Service – це шар, в якому буде міститись уся бізнес-логіка додатку. Даний шар взаємодіє із шаром Controller та шаром Repository;
- Repository – це шар, в якому відбувається взаємодія із базою даних на основі опису даних, що сформовані у шарі Model. Даний шар взаємодіє із шаром Service.

В результаті отримуємо наступний розподіл зображений на рисунку 2.8.

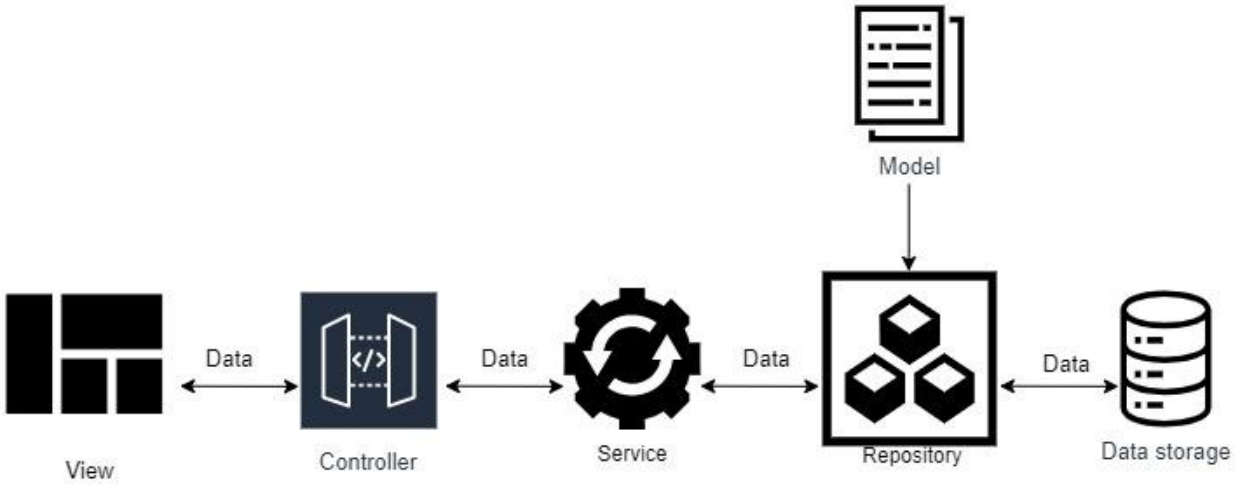


Рисунок 2.8 – Декомпозиція MVVM архітектури

Наступним кроком буде визначення парадигм, які будуть використані для побудови клієнтського та серверного додатків. Під словом «парадигма»

мається на увазі визначенням структури додатку, яка буде використана при розробці.

На сьогодні виокремлюють три основні парадигми програмування: структурне, функціональне та об’єктно-орієнтоване[15]. Якщо ми розглянемо модель бази даних, отриману у розділі 2.5, то можемо дійти до висновку, що під дану структуру доцільно використати об’єктно-орієнтоване програмування(ООП).

В свою чергу ООП включає ряд інструментів, які будуть використані при розробці додатку:

- Інкапсуляція – це можливість групування і приховування даних та функцій. В результаті інкапсуляції користувач має доступ тільки до наданого API, при цьому конкретна реалізація залишається прихованою;
- Спадкування – це можливість розширення чи перевизначення змінних та функцій у певній ділянці видимості(зазвичай у класах);
- Поліморфізм – це можливість використання єдиного інтерфейсу, при цьому маючи безліч реалізацій. Наслідком цього є система, що не залежить від низькорівневих реалізацій. Такий підхід ще називають «Інверсією залежностей»[16].

2.7 Постановка задачі автоматизованої побудови маршрутів у вигляді математичної моделі

Розглянемо загальну поставку для даної задачі: необхідно знайти найкоротший шлях між двома точками використовуючи задану мапу, в цьому випадку в якості місць які планує відвідати користувач виступатимуть вершини графа, а дорога що сполучає дані місця – ребра із заданими вагами. Таким чином, дана задача зводиться до пошуку найкоротшого та найоптимальнішого маршруту на графі.

Пошук найкоротшого маршруту базується на знаходженні шляху між заданими вершинами графу, щоб сума ваг ребер при цьому була мінімальною.

Математична модель буде виглядати наступним чином: у заданому зваженому графі, що містить набір вершин V, а також ребер K із наступною функцією ваги: $f: K \to R$, та заданими елементами v із V, визначити маршрут R із v до $v`$ такий, що:

$$\sum_{r \in R} f(r) \tag{2.1}$$

є найменшою з-поміж усіх наявних маршрутів, які є допустимими для сполучення вершин v та $v`$.

2.7.1 Навчання із підкріпленням

Навчання із підкріпленням(англ. Reinforcement learning, або RL) – це область машинного навчання, яка застосовуються для знаходження найкращого можливого шляху або поведінки[17]. Особливістю даного типу навчання є те, що штучний інтелект здійснює процес навчання базуючись на власних помилках.

Навчання із підкріплення ставить перед собою наступні цілі[18]:

1) Зменшити кількість помилок до мінімального значення. Це досягається за рахунок повторення ходів моделлю, при цьому на кожній наступній ітерації враховується досвід попередніх ітерацій;
2) Збільшити максимальну вигоду в результаті виконання певного завдання. Варто зауважити, що вигоду слід вказати завчасно. Під вигодою, в контексті побудови туристичних маршрутів, можна визначити наступні показники: мінімальний час проходження маршруту або проходження найкрасивіших місць певної локації.

Навчання із підкріпленням поділяється на два наступні типи[19]:

1) Позитивне підкріплення – це збільшення поведінкової сили моделі за рахунок збільшення певної змінної, щоб модель повторювала рішення, які є задовільними для вирішення поточної задачі;

2) Негативне підкріплення – це зменшення поведінкової сили моделі за рахунок зменшення певної змінної, щоб модель уникала повторів рішення, які не є задовільними для вирішення поточної задачі.

Головним аргументом застосування RL для вирішення даної задачі є відсутність даних для навчання нейромережі. Проте, RL має ряд недоліків, ключовим з яких є те, що певні параметри можуть негативно впливати на динаміку навчання. Це може спричинити доволі помітну і непрогнозовану затримку для користувачів. Тому, даний алгоритм не буде використаний для вирішення проблеми автоматизованої побудови туристичних маршрутів.

2.7.2 Мурашиний алгоритм

Мурашиний алгоритм – це один із методів вирішення проблеми оптимізації знаходження найкоротшого маршруту(або ж задачі комівояжера)[20]. В основі алгоритму знаходиться опосередковане спілкування мурах, яке відбувається за рахунок феремонних слідів. Це дозволяє мурахам самоорганізовуватися. Також, спілкування за рахунок феромонів відображає спілкування в колонії за рахунок зміни навколишнього середовища. Дана модель поведінки ще має назву stigmergy.

Процес знаходження рішення, відбувається за рахунок переміщення агентів(штучні мурахи) на зваженому графі. У процесі переміщення кожний агент поступово створює рішення, при цьому змінюючи значення вершин або ребер графа. Сам процес зміни значення вершини або ребер графу є моделлю феромонів.

Даний алгоритм складається із наступних кроків[21]:

1) Кожний агент знаходиться у вихідному положенні, на усіх маршрутах феромони відсутні;
2) Кожний агент розпочинає рух у пошуках їжі, при цьому ймовірність проходження певної точки(вершини графу) складає 0,5 кожна;

3) Коли агенти знаходять коротший шлях до кінцевої точки, ймовірність інших агентів натрапити на цей шлях вища. За рахунок цього збільшується ймовірність проходження певних точок, що були успішно пройдені раніше і приводять до певної кінцевої точки;
4) Більшість агентів використовує коротший шлях для повернення на вихідну точку, внаслідок чого збільшується концентрація феромонів у точках, які відповідають найкоротшому маршрутові, при цьому кількість феромонів зменшується на точках які входять до довших маршрутів. Тому, з певним часом, імовірність використання певних точок зростає і усі агенти будуть використовувати коротший шлях. Мета алгоритму досягнута.

Розглянемо детальніше пересування агентів по графові. Обрахування імовірності переходу k-го агенту із вершини i до вершини j на ітерації n, виглядає наступним чином:

$$p_{ij} = \begin{cases} \dfrac{(\tau_{ij}(n))^{\alpha}(\eta_{ij})^{\beta}}{\sum_{l \in V^{acept}} (\tau_{il}(n))^{\alpha}(\eta_{il})^{\beta}}, & j \in V^{acept} \\ 0, & j \notin V^{acept} \end{cases} \qquad (2.2)$$

де, $\tau_{ij}(n)$ - це умовна імовірність переходу із вершини i до вершини j, яка залежить від інтенсивності феромонів для відповідного ребра;

η_{ij} - це апріорна ефективність переходу між вершинами;

V^{acept} - визначає множину допустимих вершин.

Також, пересування агентів має перелік певних особливостей, а саме[22]:

- при обрахуванні імовірності переходу здійснюється балансування концентрації феромонів $\tau_{ij}(n)$, що є значенням яке відображає попередні пересування. Для управління цим балансом застосовуються коефіцієнти α та β. Коли коефіцієнт α = 0, агенти ігнорують попередні дані щодо

проходження даної вершини. Якщо $\beta = 0$, тоді вибір здійснюється виключно на основі феромонів;

- множина V^{acept} визначає допустимі вершини для поточного агента. Дана множина включає в себе перелік вершин, які агент може відвідати на поточній ітерації, а також вершини які ще не були відвідані. Перелік вершин де побував агент видаляються із V^{acept} множини, та додаються до множини V^{tabu}.

Також, варто зазначити, що правило зміни концентрації феромонів виглядає наступним чином:

$$\tau_{ij}(n) = (1-\rho)\tau_{ij}(n-1) + \Delta\tau_{ij}(n) \quad (2.3)$$

Враховуючи попередні визначення, алгоритм пошуку найкоротшого маршруту буде мати наступний вигляд[22]:

1. Ініціалізація вхідних даних, а саме: встановлення коефіцієнту α, що визначає вплив феромона; встановлення коефіцієнту β, що визначає інтенсивність евристики; встановлення коефіцієнту ρ, що визначає інтенсивність випаровування; задання максимального числа ітерацій N; задання розміру популяції K; задання множини вершин ($V = \{1,...,M\}$) та матриці ваг ребер($[d_{ij}]$, $i, j \in \overline{1,M}$.) вхідного графу; створення цільової функції;
2. Ініціалізація початкового рівня феромонів $\tau_{ij}(0)$ на кожному ребрі графу. Зазвичай присвоюється невелике позитивне число;
3. Ітерація $n = 1$;
4. Номер агенту $k = 1$;
5. Розміщуємо мурах в різні вершини випадковим способом, тобто $i = round(1 + (M-1)U(0,1))$; (2.4)
6. Множина вершин, які містять пройдені $V^{tabu} = \{i\}$, $x_{k1} = i$;
7. Здійснюємо обрахунок вірогідності переходу k-го агенту із поточної вершини в інші;

8. Здійснюємо вибір вершини, яка буде задовольняти наступну нерівність:

$$\sum_{j=1}^{j_c-1} p_{ij} < U(0,1) \le \sum_{j=1}^{j_c} p_{ij} \qquad (2.5)$$

9. Якщо $|V^{tabu}| < M$, то $i = j_c$, здійснюємо перехід до пункту 7;

10. Якщо $F(x_k) < F(x^*)$, тоді $x^* = x_k$;

11. Якщо $k < K$, то $k = k + 1$, тоді переходимо до пункту 5;

12. Правило зміни рівня феромонів представлено в наступному вигляді:

$\tau_{ij}(n) = \tau_{ji}(n) = (1-\rho)\tau_{ij}(n-1) + \gamma |Q_{ij}|$, $i \in \overline{1, M-1}$, $j \in \overline{i+1, M}$,

де $Q_{ij} \subset \{x_k\}$ - це множина вершин, які містять ребро (I;j) або ж (j, ib);

13. Якщо $n < N$, тоді $n = n + 1$ та перехід до пункту 3, інакше завершення виконання алгоритму і результатом виконання є x^*.

Даний алгоритм доволі оптимально підходить для вирішення даної задачі. Проте є певні набори даних, для яких алгоритм функціонувати коректно не буде, а саме:

- користувач надає однакові початкову і кінцеву точку. Для вирішення даної підзадачі необхідно додати попередню обробку даних і при наявності однакових початкової і кінцевої точки застосувати наступне: розрахувати кількість часу, який необхідно витратити до кожної із проміжних точок; визначити час який необхідний для проходження половини маршруту і проміжну точку яка відповідає даному критерію; після чого розділити маршрут навпіл і розрахувати інший маршрут повернення до вихідної точки. За рахунок даного підходу ми надаємо можливість туристові відвідати більшу кількість місць.
- користувач надає проміжні точки маршруту з-поміж яких є однакові. Для вирішення даної підзадачі необхідно додати попередню обробку даних, перед застосуванням алгоритму, а саме: здійснити ітерацію кожного елементу і перевірити на наявність повторів; якщо повтори

присутні: розділити маршрут на кількість частин, при цьому кожна частина не повинна містити однакових проміжних точок, після чого застосувати алгоритм до проміжних маршрутів. Після завершення знаходження проміжних маршрутів необхідно з’єднати їх у відповідності до вхідного маршруту.

Враховуючи певну оптимізацію вхідного набору даних, представлено схему побудови туристичного маршруту на основі мурашиного алгоритму на рисунку 2.9.

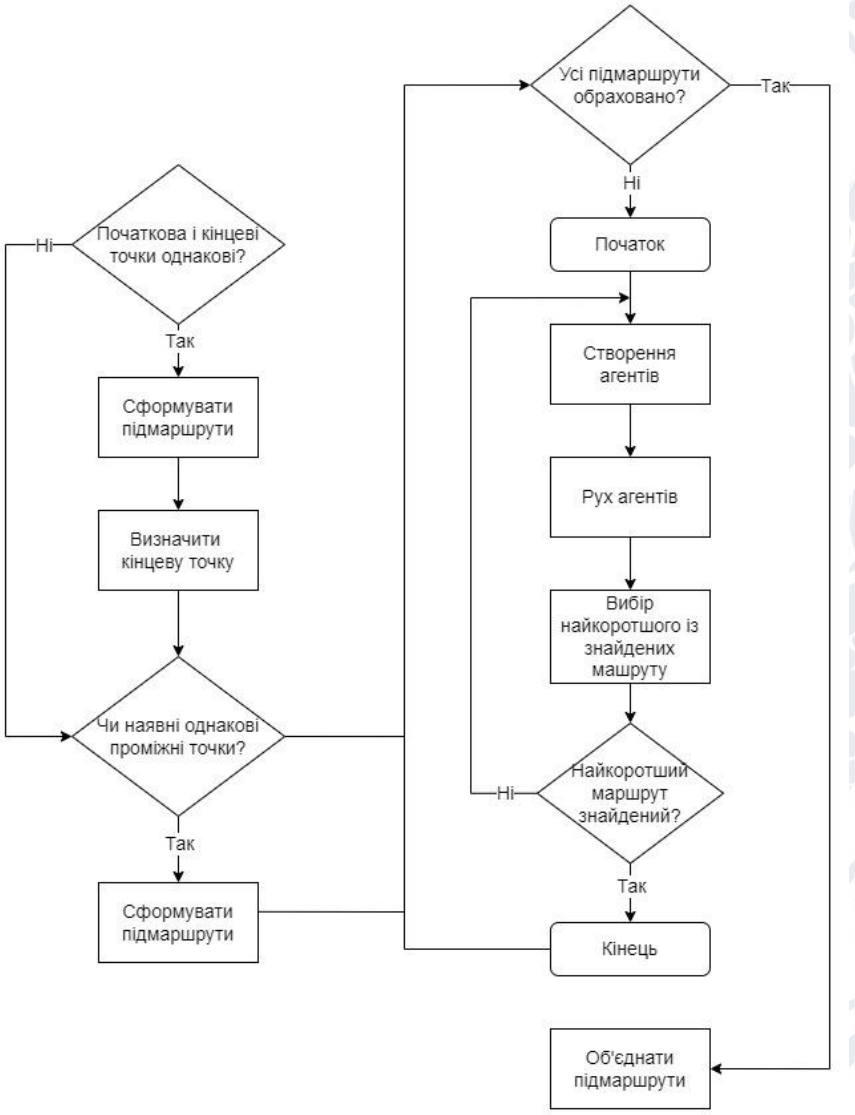


Рисунок 2.9 – блок схема мурашиного алгоритму із попередньою обробкою даних

За рахунок усунення вищеперерахованих недоліків для вхідних наборів даних, даний алгоритм можна застосувати для вирішення проблеми автоматичної побудови туристичних маршрутів.

2.8 Обґрунтування обраних технологій

Здійснивши аналіз наявних мов програмування, технологій та фреймворків, було сформовано наступний технологічний стек проекту: СУБД PostgreSql, засіб адміністрування СУДБ Pgadmin4, мова програмування Java для серверної частини, фреймворк Spring, мова програмування Javascript для клієнтської частини, фреймворк ReactJS, інструмент збору метрик Prometheus, інструмент для відображення графіків Grafana, інструмент потокової обробки в реальному часі Hazelcast, інструмент перевірки якості коду Sonar cube, інструмент реверс-проксі Traefik, в якості розгортання та оркестрації Docker та Docker-compose. Розглянемо більш детально кожен із компонентів.

2.8.1 СУБД PostgreSQL

PostgreSQL – це реляційна база даних, вихідний код якої є відкритим. Мова запитів розроблена на основі SQL, використовує і розширює її можливості[23]. PostgreSQL здійснює управління даними, які зберігаються в БД. Також, передбачається можливість працювати паралельно із декількома БД. Процес об’єднання і управління декількома БД називається кластеризацією. Кожний кластер містить певний перелік таблиць, що відповідають за зберігання метаінформації, яка описує внутрішню структуру поточної БД.

Також, важливою особливістю PostgreSQL є резервне копіювання даних. Для забезпечення даної функції було надано допоміжну програму pg_dump[24], яка постачається за замовчуванням із інсталятором СУДБ. Процес копіювання даних відбувається за рахунок запису вказаних даних у стандартний вивід(зазвичай у файл). Дана утиліта працює на хості, де було розгорнуто СУДБ, а також вимагає прав суперкористувача. Використання даного методу при резервному копіюванні надає можливість створювати копії які будуть сумісні із різними версіями PostgreSQL, в той час як резервні копії

які працюють за рахунок безпосереднього копіювання файлів є залежними від конкретної версії серверу.

При інсталяції даної СУДБ є ряд параметрів, які встановлюються за замовчуванням. Проте, якщо є необхідність, стандартні параметри можна перевизначити. Список параметрів які були перевизначені є наступними[25]:

- postgres_user – змінна, що визначає ім’я користувача, якому буде встановлено роль суперкористувача та створено БД із відповідною назвою. За замовчуванням – postgres, при розгортанні змінну було перевизначено на walker.
- postgres_password – змінна, що встановлює новий пароль для користувача postgres_user, за замовчуванням postgres.
- pgdata – змінна, яка встановлює шлях, який буде використовуватись для збереження усіх даних СУДБ.
- temp_buffers – змінна, що відповідає максимальному розміру буфера для тимчасових файлів, встановлене значення складає 64 мб;
- shared_preload_libraries – змінна, яка дозволяє завантажити одну або декілька бібліотек, що дозволить зекономити час при першому виклику бібліотеки, оскільки завантаження бібліотеки відбувається безпосередньо під час першого виклику. Варто зауважити, що дана функція не буде працювати, якщо СУДБ буде розгорнуто на операційні системі(ОС) Windows.
- vacuum_cost_delay – змінна, яка визначає час перебування процесу у режимі сну в разі перевищення часу виконання. За замовчуванням дорівнює 0, однак при розгортанні було встановлено 10 мілісекунд.
- vacuum_cost_page_hit – змінна, що визначає ціну очищення буферу, який розташований у спільному кеші. Ціна впливає на кількість, протягом якого будуть утримуватись ресурси в режимі блокування. За змінна становить 1, однак при розгортання було встановлено 5.

- vacuum_cost_page_miss – змінна, що визначає ціну очищення буферу, який розміщується на диску. За замовчуванням змінна становить 10, однак при розгортанні встановлено 15.
- effective_in_concurency – змінна, що встановлює максимальну кількість паралельних операцій читання та запису для кожного сеансу користувача. Якщо необхідно відключити паралельні запити, тоді варто присвоїти даній змінній 0, якщо ж передбачається асинхронність, тоді присвоюється значення від 1 до 1000. При розгортанні встановлено значення 100.

2.8.2 Pgadmin4

Pgadmin4 – це програма, що надає зручний користувацький інтерфейс для адміністрування СУДБ PostgreSQL[26]. Дана програма розгортається в якості незалежного сервісу, що дозволяє адмініструвати одразу декілька СУДБ.

Важливою особливістю Pgadmin4 є наявність двохфакторної автентифікації, яку варто налаштувати при розгортання на реальному хостові. Даний підхід дозволяє зменшити імовірність несанкціонованого доступу до системи. Pgadmin4 надає два типи двохфакторної автентифікації: використовуючи надсилання повідомлення із кодом підтвердження на пошту користувача, яку було вказано при реєстрації, а також використовуючи спеціалізовані додатки, такі як Google Authenticator.

При інсталяції Pgadmin4 було перевизначено ряд системних параметрів, таких як[27]:

- pgadmin_default_email – змінна, яка визначає адресу електронної пошти для користувача за замовчуванням, вона є обов’язковою при запуску. За замовчуванням дана змінна admin@admin.admin

- pgadmin_default_password – змінна, яка встановлює пароль для користувача за замовчуванням, проте не є обов'язковою при запуску. За замовчуванням пароль відсутній.
- pgadmin_disable_postfix – змінна, що визначає запуск утиліти postfix, яка відповідає за доставку електронних листів для відновлення або заміни пароля. За замовчуванням дана утиліта не запускається. Також, при її використанні варто визначити SMTP-сервер, який буде використано. Дану утиліту слід активувати, якщо передбачається використання двохфакторної автентифікації використовуючи надсилання повідомлень на електронну пошту користувача.
- pgadmin_enable_tls – змінна, що дозволяє встановити прослуховування додатком лише порту 443, оскільки за замовчуванням прослуховується 80 порт. У разі увімкнення tls прослуховування, варто надати сертифікат та ключ.
- pgadmin_listen_port – змінна, що дозволяє прослуховувати нестандартний порт. Оскільки за замовчуванням прослуховується 80, а при увімкненні tls – 443. Також, якщо увімкнено tls прослуховування, порт який ми встановимо у дану змінну – також використовувати лише tls.
- gunicorn_access_logfile – змінна, яка дозволяє зберігати логи Pgadmin4 у певний файл, оскільки за замовчуванням усі логи надсилаються у stdout, що не дозволяє відслідкувати помилки.

Важливою функціональністю, яку надає Pgadmin4, є наявність зручного користувацького інтерфейсу для резервного копіювання та відновлення даних. В офіційні документації вказано, що при використанні даних функції використовується утиліта pg_dump, яка є пріоритетним інструментом для резервного копіювання. Також, при виконанні даної операції є можливість вказати які саме дані, в якому форматі і в який файл записати.

Для виконання відкладених завдань, або завдань які необхідно виконати у запланований час, використовується утиліта pgAgent. Дану утиліту варто

встановити на хості де розгорнуто СУДБ і виконати початкові налаштування, такі як: вказання користувача, пароля користувача та назву бази даних з якою слід працювати. pgAgent періодично опитує сервер на наявність нових завдань. В разі наявності невиконаних завдань відбувається ініціалізація окремого агенту для виконання поточного завдання. Сам процес виконання завдання можна покроково відслідкувати використовуючи користувацький інтерфейс. Дану утиліту доволі зручно використовувати для налаштування резервного копіювання даних.

Суттєвим недоліком є відсутність автентифікації використовуючи JWT. Оскільки, даний тип автентифікації було застосовано при розробці серверної частини, повторне використання даного підходу для усіх сторонніх сервісів є пріоритетним, проте не критичним. Враховуючи специфіку даного інструменту, було прийнято рішення використовувати його лише для адміністрування ресурсу, а в якості авторизації використовувати стандартну реалізацію у поєднанні із двохфакторною автентифікацією.

2.8.3 Java

Java – це мова програмування, яка базується на парадигмі ООП, є компільованою та строго типізованою[28]. Скомпільований вихідний код інтерпретується віртуальною машиною(JVM), реалізації якої розроблені під усі сучасні ОС.

Основою популярності даної мови програмування є те, що із самого початку Java проектувалась для виконання на різних платформах одного і того ж скомпільованого коду, фактично забезпечуючи незалежність від платформи. За рахунок тривалого часу використання даної мови як основної в більшості компаній, на сьогодні існує безліч проєктів які потрібно підтримувати та розвивати. Також, розробники JVM забезпечують зворотню сумісність старих програм. Для прикладу, програму, яка написана 20 років тому, можна із легкістю запустити на найновіших версіях віртуальної машини.

Не зважаючи на те, що дана мова програмування існує досить давно, динаміка використання(рис. 2.10) свідчить про стабільно високе використання в якості основного інструменту розробки.

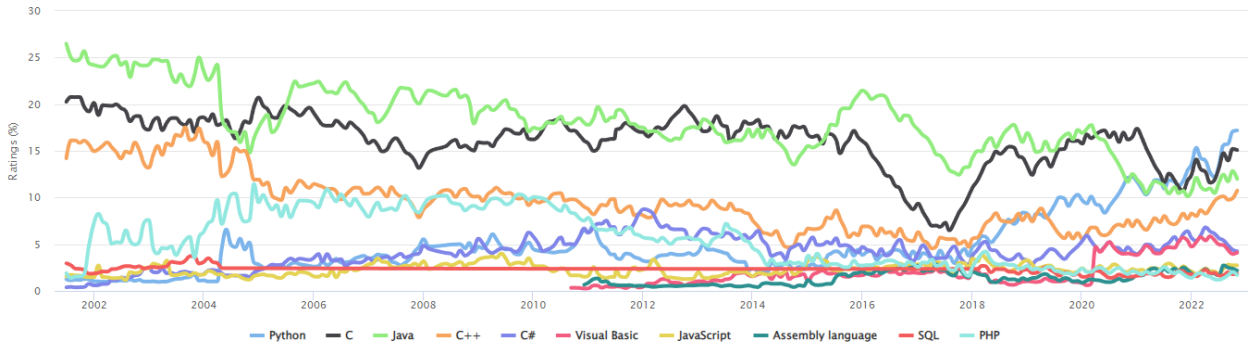


Рисунок 2.10 – Графік популярності мов програмування[29]

Java використовується у різних сферах розробки[30]:

- web-додатки – за рахунок стабільної і передбачуваної роботи, великі проекти реалізовується на Java(наприклад банківська система);
- android-додатки – більшість додатків на дану ОС були розроблені із використанням Java, оскільки Google визначав її як основну мову програмування для Android;
- desktop-додатки – безліч програмного забезпечення розроблено використовуючи дану мову програмування, щоб зберегти сумісність для різних ОС;
- нейромережі – за рахунок простоти у вивченні, на даній мові програмування було реалізовано велику кількість бібліотек із відкритим вихідним кодом для розробки та навчання нейромереж.

До сильних сторін Java також варто віднести наступне:

- велика і активна спільнота;
- чудово задокументований відкритий вихідний код;
- широкий перелік готових інструментів для розробки та інтегрованих середовищ розробки;
- можливість інтеграції із більшістю реляційних та нереляційних БД;
- обширний API, який досягається за рахунок безлічі бібліотек із відкритим вихідним кодом.

2.8.4 Фреймворк Spring

Spring – це фреймворк, написаний на Java і надає реалізацію ін'єкції(DI) залежностей та інверсії управління(IoC)[31]. Даний інструмент надає безліч готових рішень та додаткових сумісних бібліотек, які значно прискорюють розробку, а також дозволяють сконцентруватись виключно на бізнес-процесах додатку.

Головною причиною використання даного фреймворку є те, що він дозволяє зосередитись виключно на впровадженні бізнес-логіки додатку при цьому дотримуючись загальних правил чистої архітектури. У свою чергу це спрощує підтримку кодової бази і зменшує криву навчання для нових розробників у проєкті.

Spring є модульним фреймворком, що дозволяє використовувати лише ті компоненти, які вимагають поточні задачі. До основних модулів відносять:

- Core – це обов'язковий до імпорту модуль, оскільки в ньому міститься реалізація ін'єкції залежностей та інверсія управління, на яких базується фреймворк. Окрім цього модуль включає інтернаціоналізацію та аспектно-орієнтоване програмування;
- Data – це модуль що спрощує інтеграцію та обмін між додатком та базою даних;
- Web – це модуль, що дозволяє здійснювати взаємодію із web-додатками, включає в себе реалізацію паттерну MVC, реактивне програмування, а також підтримує WebSockets;
- Integration – це модуль, який дозволяє здійснювати ітеграцію із компонентами Java Enterprise, а саме службою повідомлень(JMS), віддаленими викликами методів(RMI) та розширеним управлінням(JMX);
- Testing – це модуль, який включає в себе підтримку unit та інтеграційних тестів[32].

Якщо розробник використовує тільки вищеперераховані модулі, це значно спрощує і пришвидшує розробку. Однак, при розгортанні додатку, необхідно буде визначити сервер для розгортанню додатку, такий як Tomcat або ж Glassfish. За рахунок тривалого розвитку даного фреймворку, було розроблено розширену версію, яка отримала назву Spring Boot[33]. Дана версія включає в себе вбудований сервер додатків і безліч автоконфігураційних параметрів.

За рахунок простоти розгортання Spring Boot набув поширення серед розробників та вважається корпоративним стандартом для розробки додатків. В свою чергу, на основі Spring Boot було розроблено безліч додаткових модулів, таких як[34]:

- Cloud – модуль, який створено для спрощення розробки мікросервісних додатків;
- Security – модуль, який надає безліч варіантів конфігурацій безпеки для додатку;
- Batch – модуль що забезпечує створення та запуск відкладених завдань, збір та зберігання метрик.

У автоматизованій системі побудови і агрегації туристичних маршрутів використано базові модулі Spring, а саме: Core, Data, Web, Testing. Окрім того до проєкту включено Spring Boot із модулями Cloud та Security.

За рахунок використання модуля Data було організовано взаємодію між додатком до СУБД Postgresql. Використовуючи модуль Web було реалізовано REST API для взаємодії із клієнтським додатком. Варто зазначити, що модулі Data та Web дозволяють у повній мірі здійснити програмну реалізацію архітектури додатку, яку було розроблено у розділі 2.6.

Модуль Security був використаний для забезпечення авторизації та автентифікації користувача. При цьому за основу даного механізму було взято авторизацію та автентифікацію, що базується на JWT токенах.

2.8.5 Prometheus

Prometheus – це система збору метрик та оповіщень із відкритим вихідним кодом розроблена компанією SoundCloud[35]. Дана система забезпечує збір та зберігання метрик у базу даних часових рядів.

Дана система має наступні головні особливості[35]:

- наявність власної, простої в освоєнні, мови запитів PromQL;
- збір метрик може відбуватись за рахунок використання HTTP протоколу, або ж використовуючи проміжний шлюз;
- проста інтеграція із зовнішніми системами;
- надає декілька варіантів налаштування графічного відображення метрик;
- все налаштування системи відбувається за рахунок використання статичної конфігурації;
- вбудована система сповіщень;
- метрики зберігаються до бази даних часових рядів, яка в свою чергу забезпечує багатовимірну модель даних.

Головною метою впровадження даної системи є те, що здійснювати моніторинг системи доволі складно, якщо наявно безліч компонентів, тому для зручності моніторингу сервісів та систем варто додати та налаштувати Prometheus. Завдяки цьому впровадженню час на пошук причини збою системи значно скорочується.

За рахунок досить розвинутої спільноти, для Prometheus було розроблено безліч експортерів даних, які дозволяють інтегрувати більшість систем для моніторингу. Експортери варто використовувати, якщо прямий експорт недоступний.

Для моніторингу хоста на якому буде розгорнуто систему автоматизованої побудови та агрегації туристичних маршрутів, а також усіх підсистем, утиліт та інших інструментів буде запроваджений збір метрик, який відображено на рисунку 2.11.

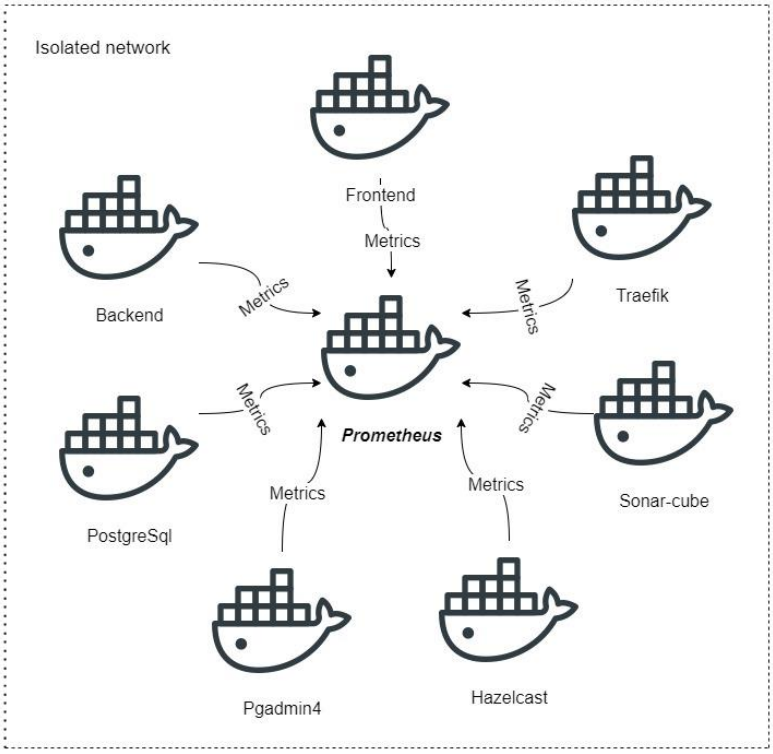


Рисунок 2.11 – Збір метрик системою Prometheus

Prometheus дуже зручно використовувати у поєднанні із Grafana. Оскільки Grafana підтримує мову запитів PromQL та має потужні можливості для побудови графіків. Для даної інтеграції необхідно додати нове джерело даних у відповідній вкладці користувацького інтерфейсу Grafana. Після чого ми обираємо графік та здійснюємо подальше налаштування, вказуючи які мітки варто відображати.

При інсталяції Prometheus було перевизначено ряд системних параметрів, таких як[36]:

- docker_sd_config – змінна, що дозволяє налаштувати збір метрик із вказаних контейнерів, для цього слід вказати id контейнеру, назву контейнеру, мережу контейнеру та її характеристики, порт на якому буде прослуховуватись контейнер;

- hetzner_sd_config – змінна, що дозволяє встановити сканування із хостингового сервісу Hetzner. Для цього слід вказати id серверу, назву серверу, датацентр розміщення серверу, а також визначити перелік метрик які варто обробляти.

Проте серед усіх вищезазначених переваг є наявність недоліків, основним із яких є відсутність авторизації використовуючи JWT. Проте, дану систему буде розгорнуто у ізольованій приватній мережі без прямого виходу до мережі інтернет, а також буде використана авторизація на основі логіна і пароля користувача.

2.8.6 Grafana

Grafana – це система для візуалізації та аналітики даних із відкритим вихідним кодом[37]. За рахунок інтеграції із більшістю популярних БД, дана система надає можливість створювати інтерактивні графіки використовуючи мову запитів, яка відповідає інтегровані БД.

Дана система активно використовується для візуалізації метрик, моніторингу інфраструктури і дозволяє більш наочно аналізувати поведінку системи. Також Grafana використовують при наукових дослідженнях, для красивого відображення даних, а також усюди де потрібна наочність даних.

Дана система має наступні головні особливості[37]:

- підтримка більшості наявних БД;
- можливість відправлення сповіщень до корпоративних сервісів;
- за допомогою фільтрів можна здійснювати дослідження метрик;
- достатньо велика стандартна бібліотека готових інформаційних панелей;
- збереження готових графіків у JSON формат;
- можливість інсталяції та використання великої кількості безкоштовних плагінів;

- можливість створення звітів, однак дана функція недоступна у версії із відкритим вихідним кодом.

У системі автоматизованої побудови та агрегації туристичних маршрутів Grafana буде використана як засіб моніторингу метрик, за рахунок інтеграції із Prometheus. Також, буде використано для відображення перепаду висот для маршруту, візуалізація маршрутів на карті, а також як інструмент відображення інтерактивних графіків для аналізу активності користувача. Враховуючи вищеперераховані фактори загальний вигляд архітектури компонентів відображено на рисунку 2.12.

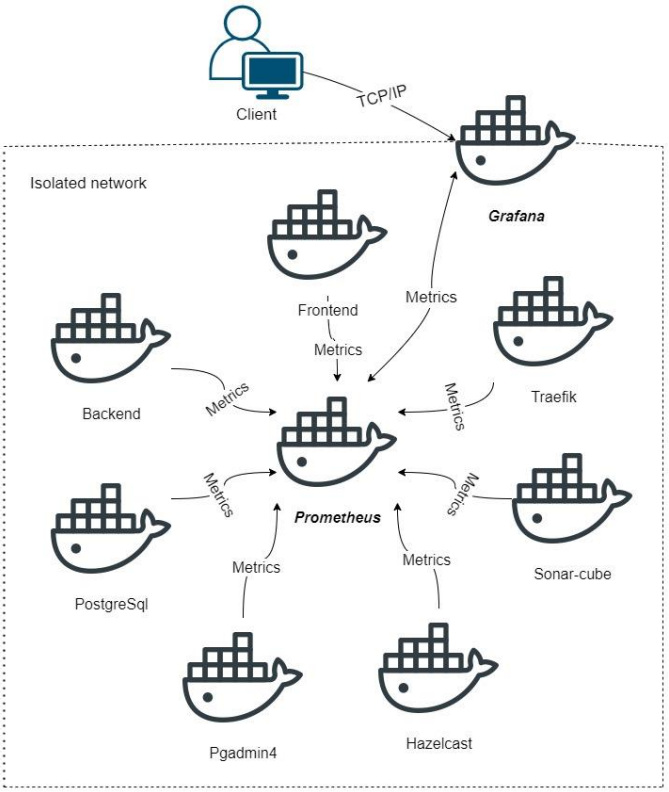


Рисунок 2.12 – Архітектура взаємодії збору метрик

Grafana надає безліч методів для авторизації користувача, такі як Oauth2, LDAP, SAML, Auth Proxy, JWT. Варто приділити увагу JWT авторизації, оскільки даний тип використовується для серверної частини додатку. Для впровадження даного типу авторизації, необхідно згенерувати публічний ключ на основі приватного ключа, доступ до приватного ключа здійснюється лише серверним додатком. Надалі, при розгортанні варто перевизначити параметри, які знаходяться у секції auth.jwt[38], а саме:

- enabled – булева змінна, яка вказує чи використовувати даний метод авторизації, за замовчуванням false;
- header_name – змінна, яка визначає назву HTTP заголовка, який буде містити JWT токен;
- username_claim – змінна, яка визначає назву атрибуту, що містить назву користувача;
- email_claim – змінна, яка визначає назву атрибуту, що містить електронну пошту користувача;
- auto_sign_up – булева змінна, яка вказує чи ввімкнутий механізм автоматичної реєстрації користувачів, якщо поточний користувач не знайдений у внутрішній БД Grafana. За замовчуванням даний функціонал вимкнений;
- url_login – булева змінна, яка визначає чи дозволяти авторизацію використовуючи JWT токен, який був переданий в url адресі;
- key_file – змінна, яка вказує розміщення публічного ключа, який буде використовується для перевірки достовірності наданих JWT токенів;
- role_attribute_path – змінна, яка визначає назву атрибуту, що містить назву ролі або переліку ролей, які визначені користувачеві.

2.8.7 Hazelcast

Hazelcast – це система, що дозволяє здійснювати ефективну обробку даних за рахунок зберігання даних у оперативній пам’яті[39]. Завдяки даній системі можна забезпечити збільшення продуктивності застосунків, доволі легко здійснювати масштабування, а також забезпечується стабільність виконання систем.

Дана система дозволяє здійснювати обробку даних кластеризованих систем, при цьому оперативна пам’ять має бути спільною. Такий підхід до обробки даних значно прискорює процеси обчислення, оскільки дані не

зберігаються у периферійній пам’яті із подальшим читанням, що економить досить багато часу.

Hazelcast забезпечує підтримку додатків, які були реалізовані на мовах програмування Java, .Net, GO, C++, Python та Node.js. Також реалізована підтримка протоколів REST та Memcached.

У Hazelcast реалізовано створення сповіщень про поширення потоків даних, зрозумілий і зручний API для виконання операцій над потоками даних, а також інструменти для взаємодії із застарілими API та додатками[39]. Даний функціонал можна використати як агрегацію потоків даних, при цьому дотримуючись одноразової обробки.

При інсталяції Hazelcast було перевизначено ряд системних параметрів, таких як:

- prometheus_config – змінна, яка визначає хост та порт на якому знаходиться система збору метрик Prometheus;
- logging_level – змінна, яка визначає рівень моніторингу логів;
- java_opts – змінна, яка визначає перелік змінних для конфігурації JVM[39].

У автоматизованій системі побудови і агрегації туристичних маршрутів використано сумісність Hazelcast із фреймворками. А саме: замінено стандартну реалізацію кешування 2 рівня від Hibernate та використано в якості основного інструменту кешування для фреймворку Spring Boot, оскільки Spring Boot дозволяє визначати нових постачальників кешування які сумісні із JCache.

2.8.8 Sonarcube

Sonarcube – це система, що дозволяє здійснювати автоматизовану перевірку коду на відповідність загальноприйнятим стандартам чистого коду(наприклад SOLID, DRY та ін.)[40]. Підтримка кодової бази із дотриманням принципів чистого коду значно спрощує підтримку проєкту,

додавання нового функціоналу і зменшує час на освоєння проєкту для нових учасників команди.

Окрім перевірки якості коду, дана система виконує запуск автоматичних тестів для додатку. Всі результати та дані про тестування зберігаються у локальній базі даних[40]. Також надається зручний користувацький інтерфейс для перегляду звітності, який за замовчуванням використовує автентифікацію користувача за логіном та паролем.

Дана система дозволяє виконати інтеграцію із більшістю репозиторіїв, таких як GitLab, GitHub, Bitbucket та інші. При використанні git здійснюється аналіз коду лише на поточній гілці, а також при виконанні merge запиту[40].

Завдяки вищевказаним можливостям Sonarcube можна легко інтегрувати до системи безперервної доставки та безперервного розгортання(CI/CD). Завдяки такій інтеграції можна уникнути ситуацій, коли буде розгорнуто систему із наявними помилками при виконанні автоматизованих тестів або помилками безпеки(наприклад пароль прописаний у конфігурації, а не передається за рахунок env-змінних).

Sonarcube здійснює ініціалізацію проекту під час першого аналізу і зберігає потягом усього часу функціонування системи, однак якщо потрібно видалити аналіз для певних проектів передбачається спеціальна функціональність для адміністраторів ресурсу. Після ініціалізації проекту, можна здійснити додаткові налаштування використовуючи користувацький інтерфейс або ж REST API яке доступне одразу після встановлення системи.

Кожний створений проект для подальшого аналізу, за замовчуванням позначається як загальнодоступний та є доступним для користувачів із будь-якими системними ролями і навіть анонімних користувачів. Тому після ініціалізації проєкту варто здійснити налаштування доступу, щоб обмежити перегляд показників проекту, знайдених недоліків та перегляд проблематичних фрагментів коду проєкту.

За замовчуванням Sonarcube здійснює збереження звіту по проведеним скануванням проекту. Проте зберігаються не усі звіти та не усі метрики які

містить зміст, оскільки це б призвело до збитковості даних. Натомість новий звіт зберігається із позначкою «snapshot». Така позначка вказує на те що даний звіт видалити неможливо.

При кожному повторному скануванню проекту, немає необхідності перевіряти код, що не зазнав змін у порівняні із попередньою версією. Для усунення цієї проблеми було розроблено інструмент Clean as You Code. Даний інструмент дозволяє сканувати лише новий код, при цьому можна визначити області, в яких буде здійснено пошук нового коду. Передбачається наявність трьох областей сканування[40]:

- глобальний рівень – пошук здійснюється у всьому проекті, а також у всіх гілках проєкту;
- проектний рівень – пошук здійснюється в поточному проєкті, а також у кількох гілках проєкту;
- рівень гілки – пошук здійснюється виключно на поточні гілці.

При інсталяції Hazelcast було перевизначено ряд системних параметрів, таких як:

- sonar.jdbc.* - змінні, що визначають перелік параметрів, які відповідають за з’єднання із БД, таких як хост, порт, ім’я користувача та пароль;
- sonar.path.data – змінна, що встановлює шлях для зберігання даних;
- sonar.web.host – змінна, яка вказує хост на якому розміщено поточний екземпляр системи;
- sonar.web.port – змінна, яка вказує порт на поточному хостові, за замовчуванням використовується 9000 порт.

У автоматизованій системі побудови і агрегації туристичних маршрутів дана система використана для перевірки якості коду, запуску і аналізу результатів автоматизованого тестування, а також є складовим елементом CI/CD системи.

2.8.9 Reactjs

Reactjs – це веб-фреймворк реалізований на мові програмування Javascript(JS), розроблений компанією Facebook[41]. Головною метою є спрощення розробки користувацького інтерфейсу.

Базовою концепцією фреймворку є віртуальна об’єкта модель документу(DOM). За рахунок даного підходу надається можливість створювати JS компоненти із яких буде складатись користувацький інтерфейс. При цьому Reactjs здійснює мінімум дій із компонентами підтримуючи весь віртуальний DOM у актуальному стані[41]. Для цього використовується спеціальний алгоритм по знаходженню відмінностей для кожного вузла, після чого здійснюється пошук найоптимальнішого оновлення компонентів. Це призводить до підвищення продуктивності додатку.

Для реалізації однонаправленого потоку даних із певними подіями та прослуховувачами було розроблено архітектурний шаблон під назвою Flux. Таким чином досягається прогнозована поведінка оновлення внутрішнього сховища даних і створення подій додатку. Це значно спрощує забезпечення синхронізації даних у глобальній області видимості.

Оскільки Flux не є конкретною реалізацією, а лише архітектурним шаблоном, розробниками Reactjs було створено Redux, який є реалізацією даного шаблону[41]. Redux також можна розглянути як глобальний контейнер даних, який кешує, оновлює та синхронізує інформацію, а також спрощує створення внутрішніх сповіщень.

Reactjs є одним із трьох(Angularjs, Vuejs) найпопулярніших фреймворків для розробки користувацького web-інтерфейсу. За рахунок такої поширеності, була сформована досить велика спільнота розробників, а також створено безліч бібліотек із відкритим вихідним кодом. Хоч Reactjs є проєктом із відкритим вихідним кодом, проте корім звичайних розробників підтримкою займаються безпосередньо спеціалісти із компанії Facebook.

У автоматизованій системі побудови і агрегації туристичних маршрутів даний фреймворк використаний для побудови користувацького інтерфейсу. В якості інструменту авторизації та автентифікації було використано JWT токени. Для взаємодії із серверним додатком, використовується REST API.

2.8.10 Traefik

Traefik – це проксі-сервер із відкритим вихідним кодом. Головною метою є спрощення розгортання та конфігурування системи для маршрутизації запитів[42].

Перевагою даної системи є присутність автоматичної конфігурації, дані якої отримуються безпосередньо із служб. Така можливість досягається за рахунок сумісності із найпопулярнішими кластерними технологіями, такими як: Docker, AWS, K8s та іншими[42].

Traefik підтримує налаштування трьома шляхами:

1) використовуючи конфігураційний файл – для цього варто створити файл traefik.yml і сказати його при запуску системи через аргументи командного рядка;
2) використовуючи аргументи командного рядка – для цього необхідно вказати ряд параметрів зазначених у документації, якщо параметр не вказано, буде використане значення за замовчуванням;
3) використовуючи змінні середовища – для цього необхідно вказати відповідні змінні при запуску кожної підсистеми, за рахунок даних змінних відбувається автоконфігурація[42].

Варто зауважити, що кожний метод конфігурації є взаємовиключним. Тобто можливо використовувати лише один типу конфігурації.

Оскільки, в якості інструменту контейнеризації використовується Docker, тому Traefik використовує мітки в якості даних для автоконфігурації. При цьому, дана система отримує приватну IP адресу та порт кожного контейнера.

При цьому, якщо контейнер відкриває один або декілька портів для обміну даних, саме ці порти будуть використані для передачі даних, якщо ж відкритих портів не зазначено – необхідно встановити мітку в контейнері, який порт варто використати.

Traefik здійснює маршрутизацію вхідних запитів до відповідних служб, за рахунок співставлення адреси із правилом, вказаним в якості мітки контейнера. За замовчування здійснюється прийом із усіх наявних ip-адрес, проте за необхідно, можна встановити перелік адрес, для яких певний ресурс буде доступний. Щоб уникнути колізії маршрутизації, під час встановлення автоматичної конфігурації здійснюється сортування усіх наявних маршрутів у порядку спадання і виклик певного ресурсу співставляється у такому ж порядку.

Для автоматизованої системи побудови та агрегації туристичних маршрутів було встановлено перелік правил маршрутизації(рис. 2.13).

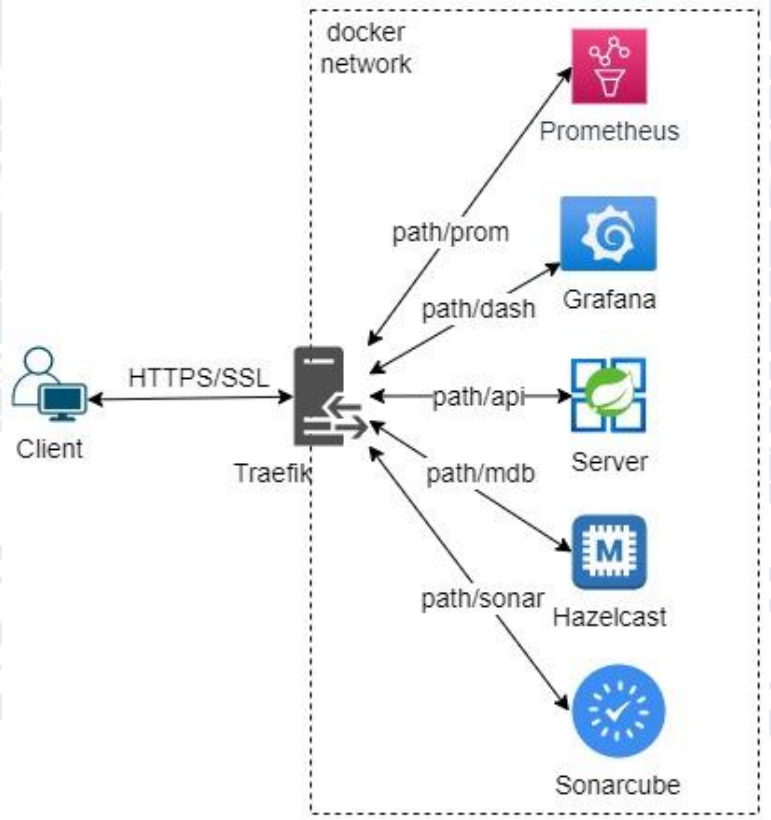


Рисунок 2.13 – Маршрутизація до систем, використовуючи Traefik

Traefik підтримує HTTPS та TLS. При цьому за замовчуванням використовується HTTP з’єднання та 80 порт. Для використання HTTPS або

TLS необхідно вказати це перевизначивши відповідну змінну при запуску контейнера. Важливою особливістю даної системи є наявність власного постачальника автоматичного отримання та обслуговування відповідних сертифікатів(ACME)[42].

Також, варто згадати про наявність плагінів. На офіційному ресурсі Traefik розміщено каталог, який містить безліч готових рішень що реалізують більшість задач, вирішення яких не передбачається у базовій версії системи. Плагіни можуть мати статичні та динамічні конфігурації, а також містити певні налаштування у файлі конфігурації.

2.8.11 Docker та Docker-compose

Docker – це інструмент *запуску та розгортання додатків за рахунок контейнеризації[43]. Контейнеризація передбачає розгортання додатків у ізольованому середовищі із власними конфігураціями.

Даний підхід дозволяє налаштувати безліч середовищ для різних система та додатків на одній машині. При цьому, кожне із налаштованих середовищ не впливає на середовище самої машини та інших контейнерів. Docker є незалежним від ОС. Це досягається за рахунок використання контейнерами Docker API яке в свою чергу реалізоване для більшості найпопулярніших ОС[43].

Окрім ізоляції різних додатків, Docker надає можливість керувати ресурсами які будуть виділені для кожного контейнеру. Для деяких підсистем було встановлено ліміти у використанні системних ресурсів за рахунок перевизначення наступних змінних:

- memory – змінна що встановлює максимальний обсяг периферійної пам’яті що може бути використаний контейнером;
- memory-swap – змінна що встановлює максимальний обсяг файлу підвантаження;

- cpus – змінна, що встановлює максимальну кількість ядер процесора, які зможе використовувати контейнер;
- gpus – змінна що встановлює кількість ядер відеокарти, які зможе використовувати контейнер[43].

Також, Docker дозволяє створювати та об’єднувати у підмережі контейнери. Підмережі дозволяють ізолювати доступ певних контейнерів що знаходяться на одній машині. Оскільки Docker завжди створює власну мережу, кожний контейнер має доступ до усіх інших що знаходяться у поточній мережі. За замовчування в якості хоста призначається назва контейнеру, а в якості порту за замовчуванням не публікується нічого. Тому, якщо потрібно оголосити порт, це варто вказати через аргументи командного рядка або ж використовуючи файл із конфігураціями.

Docker compose – це інструмент, який оснований на Docker API та дозволяє керувати багатоконтейнерними системами[44]. Даний інструмент не є заміною Docker, а швидше розширенням його можливостей.

Головна ідея полягає у створенні єдиного файлу конфігурацій для усіх контейнерів(docker-compose.yml). Це дозволяє за рахунок виконання однієї команди в терміналі виконувати маніпуляції із усіма контейнерами одночасно(запуск, зупинка, видалення та інше).

При запуску контейнерів відбується ініціалізація внутрішньої мережі між контейнерами[44]. Якщо у конфігураційному файлі не було зазначено яким чином має бути сконфігурована мережа – це відбудеться автоматично і приховано від користувача. При цьому зберігається можливість спілкування між контейнерами в одній мережі, використовуючи в якості хосту – назву контейнера.

Файл docker-compose.yml містить налаштування для кожного сервісу автоматизованої системи побудови та агрегації туристичних маршрутів. Детальний лістинг файлу наведений у Додатку Б. Також варто зазначити, що для кожного сервісу було встановлено мітки, які дозволяють забезпечувати автоконфігурацію Traefik.

2.9 Висновки до розділу 2

У даному розділі було здійснено огляд перспективності автоматизованої системи побудови та агрегації туристичних маршрутів, визначено архітектуру системи в цілому та деяких підсистем, розроблено модель бази даних, встановлено функціональні та нефункціональні вимоги, а також проведено огляд використаних технологій. Наступний розділ буде присвячений підсистемі безперервної інтеграції та безперервної доставки, а також буде детально розглянуто користувацький та адміністративний інтерфейс.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ АГРЕГАЦІЇ ТА ПОБУДОВИ ТУРИСТИЧНИХ МАРШРУТІВ

3.1 Безперервна доставка та розгортання додатку(CI/CD)

Процес безперервної доставки та розгортання дозволяє додати автоматизацію у ручне розгортання додатку. За рахунок автоматизації даного процесу розробники можуть здійснювати швидке оновлення та тестування нових версій додатку.

В першу чергу, для впровадження даного процесу, необхідно визначити стратегію ведення git-гілок. Оскільки проєкт не ведеться великою командою розробників, немає сенсу впроваджувати складну систему git-гілок, яка включає декілька основних гілок(master, release, develop, hotfix), а також безліч другорядних, що продемонстровано на рисунку 3.1.

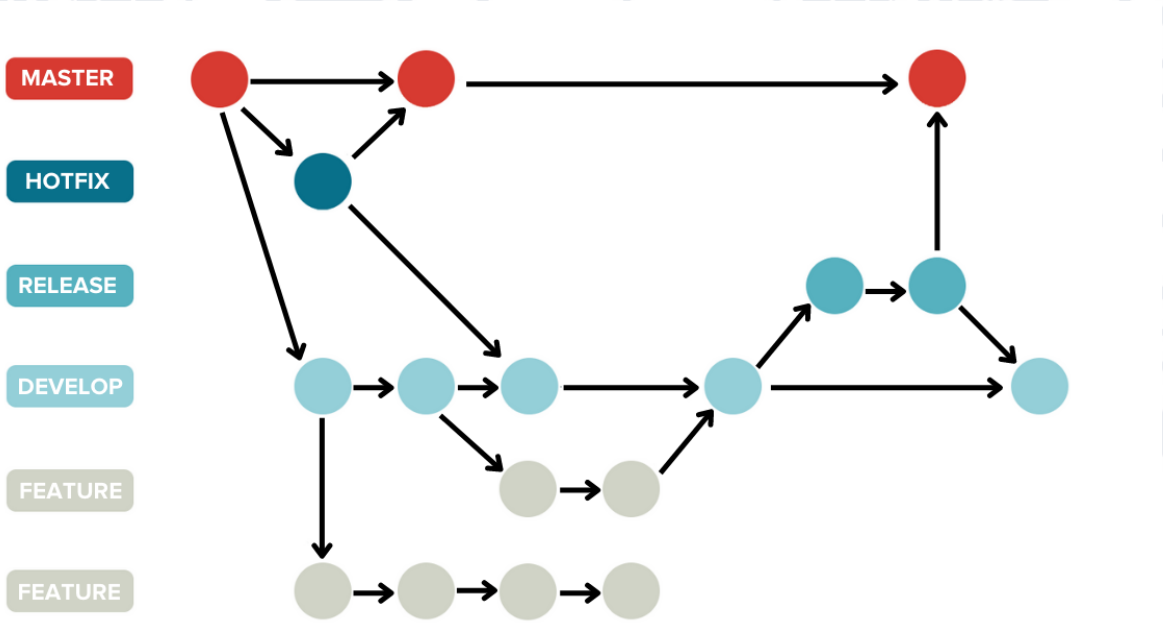


Рисунок 3.1 – стратегія ведення git-гілок[45]

Тому систему буде спрощено до трьох основних гілок, які включають наступні: master, stage(release), develop. Головні вимоги до ведення даних гілок:

- гілка master повинна містити лише робочий та протестований код, кожний commit відображає певну версію продукту;

- у гілку master можливо здійснити зливання лише із гілки release;
- гілка release містить кодову базу, яка включає останні оновлення, проте не є відтестованою за рахунок мануальних тестів;
- код із гілки release буде використаний для розгортання додатку на тестовому сервері;
- у гілку release можливо здійснити зливання лише із гілки develop, при умові що усі автоматичні тести не виявили помилок.

Безперервна доставка(CI) – це процес, що включає поєднання нових змін у кодовій базі із наявною кодовою базою, при цьому виконується автоматичне unit та інтеграційне тестування, у разі відсутності виявлених помилок – буде виконано автоматичну збірку проєкту, в іншому випадку буде повідомлено про помилку у тестуванні[46]. CI дозволяє виявляти і виправити помилки у додатку набагато раніше.

Процес перевірки кодової бази включає декілька етапів, а саме: статичний аналіз коду, автоматизована перевірка залежностей, компіляція проекту, автоматизована збірка. Надалі отриманий побудований додаток розміщується у вказаному репозиторії додатків.

Даний процес дозволяє в значній мірі оптимізувати розробку, оскільки зменшується імовірність виникнення конфліктів у кодовій базі, зменшується час на виявлення помилок та побудови проекту. Таким чином швидкість розробки та виходу нових версій продукту має позитивну динаміку розвитку.

Безперервне розгортання(CD) – це процес, який включає автоматизацію випуску нових версій програмних продуктів і працює у поєднанні із безперервною доставкою[47]. Зазвичай, розробники самі встановлюють які саме етапи потрібно включити до безперервного розгортання.

Після автоматичного тестування коду та побудови необхідно здійснити розгортання або оновлення додатку із усіма компонентами та системами. Безперервне розгортання включає в себе автоматичне розгортання додатку на тестовому сервері, а також на сервері виробництва. Сам процес розгортання є повністю автоматизований.

Для забезпечення максимальної ефективності впровадження CI/CD у виробництво було виокремлено вісім основних правил:

- проект має мати єдине сховище вихідного коду. При цьому проект повинен містити усі необхідні компоненти для збірки, такі як: додаткові бібліотеки, загальні налаштування, кодову базу і сценарій побудови;
- необхідно забезпечити чітку стратегію ведення гілок у системі контролю версій(SCM);
- файли побудови проекту повинні містити усе необхідне, щоб побудову можна було виконати за рахунок виконання єдиної команди у терміналі. Це дозволить виконати збірку придатної для використання програми із мінімальним ризиком пропустити додаткові кроки;
- у разі виникнення некоректних результатів при статичному чи автоматизованому тестуванню, варто гарантувати що збірка буде позначена як невдала;
- зміни варто впроваджувати невеликими порціями, що дозволить зменшити імовірність виникнення непередбачуваної поведінки про розгортанні нової версії;
- тестове середовище варто створювати як копію виробничого, щоб забезпечити сумісність змін про розгортанні нової версії на виробничому сервері;
- необхідно забезпечити доступ до усіх останніх побудов для учасників проекту, це дозволить виявляти потенційні проблеми у певних збірках;
- забезпечити процес розгортання максимально прогнозованою поведінкою, щоб автоматизоване розгортання можна було виконати у будь-який час[48].

Враховуючи стратегію ведення git-гілок, яку було розглянуто на початку розділу, можна визначити наступні правила для неперервного розгортання та неперервної доставки:

- при додаванні коду у гілку release необхідно здійснити статичний аналіз коду, виконати автоматичне тестування, здійснити збірку проекту,

результат збірки розмістити у відповідному репозиторії збірок, після чого здійснити розгортання на тестовому сервері, що має таке ж середовище, як і сервер виробництва, після усіх попередніх етапів варто виконати мануальне тестування;

- при додаванні коду у гілку master необхідно переконатись, що версія додатку на тестовому сервері не містить помилок. Після чого виконати наступні етапи: здійснити статичний аналіз коду, виконати автоматичне тестування, здійснити збірку проекту, результат збірки розмістити у відповідному репозиторії збірок, після чого здійснити розгортання на сервері виробництва.

Функціонування CI/CD системи для розгортання додатку на виробничому сервері відображено на рисунку 3.2.

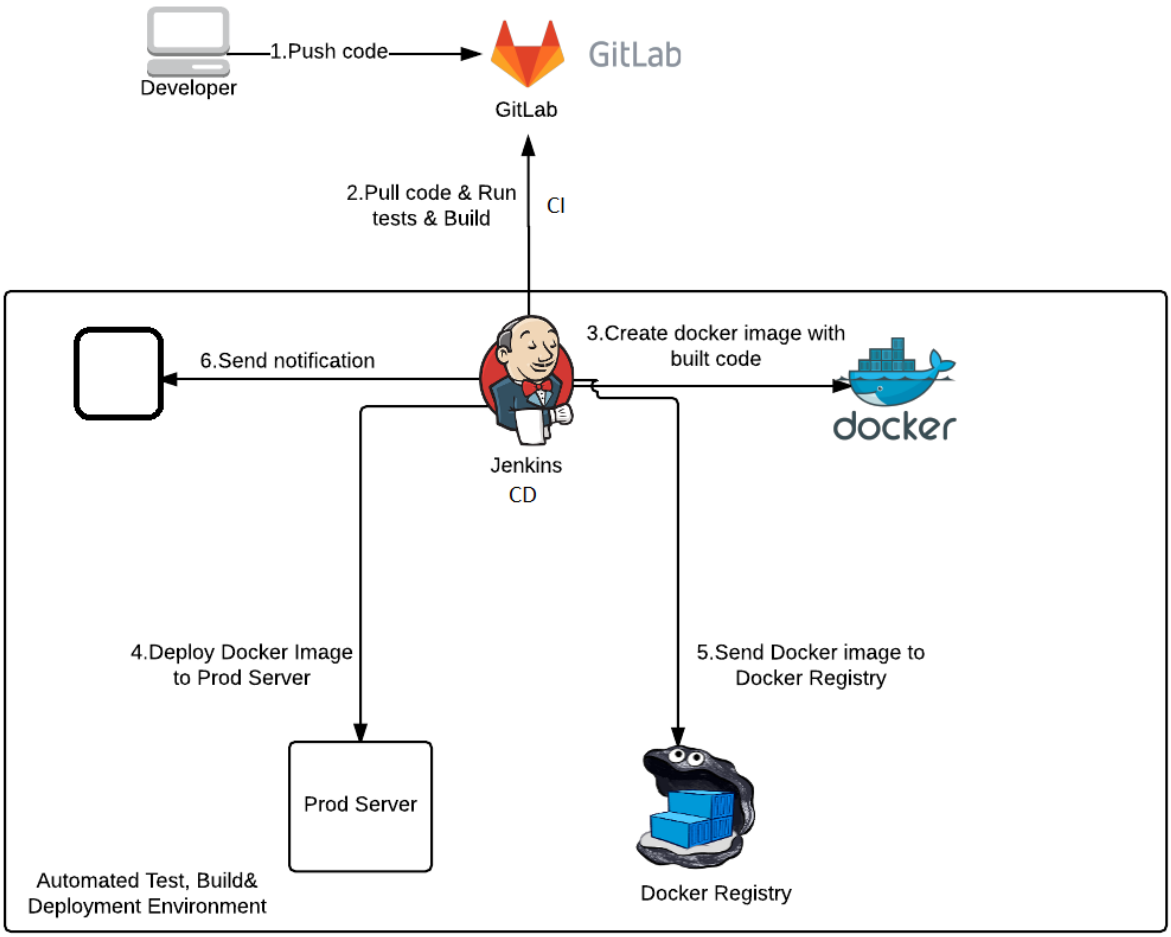


Рисунок 3.2 – Функціонування CI/CD системи[49]

3.2 Опис користувацької частини додатку

Оскільки дана система була розроблена як web-додаток, для перегляду користувацької частини необхідно здійснити перехід на відповідний сайт у браузері. При відкритті сайту для користувача буде доступна авторизація, а також перегляд карти із маршрутами(рис. 3.3).

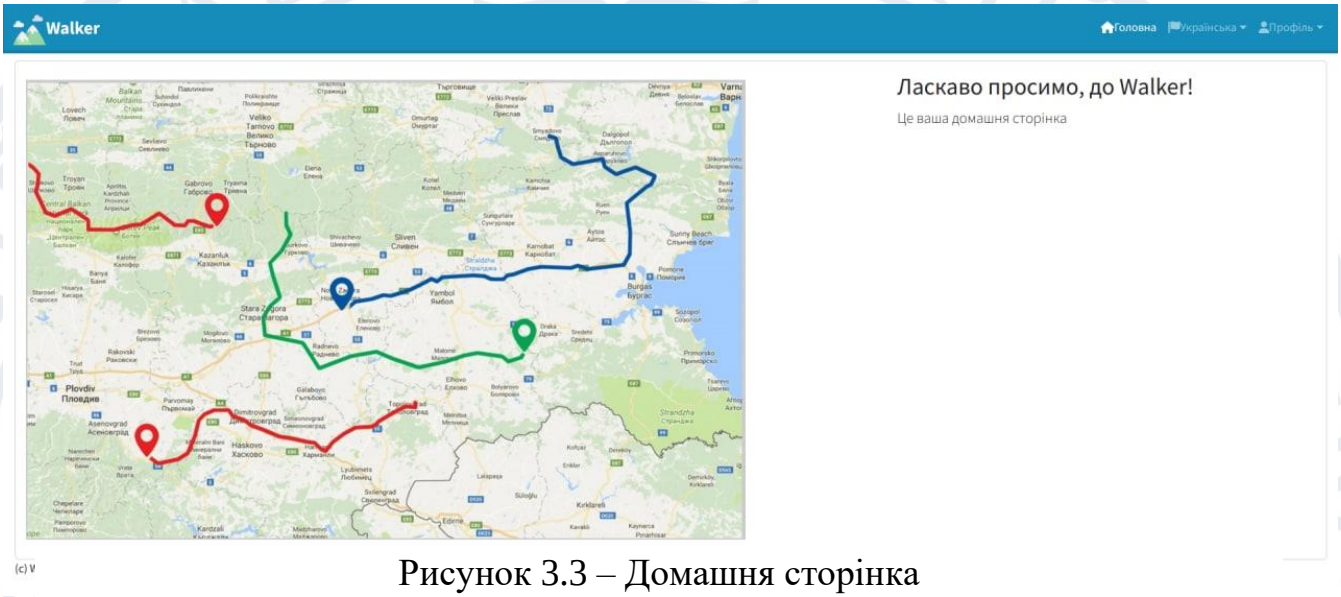


Рисунок 3.3 – Домашня сторінка

Також для користувача є доступним перегляд готових маршрутів, що були згенеровані попередньо, а саме при натисканні на будь-який із маршрутів на карті, користувачеві буде відображено поточний маршрут із детальним

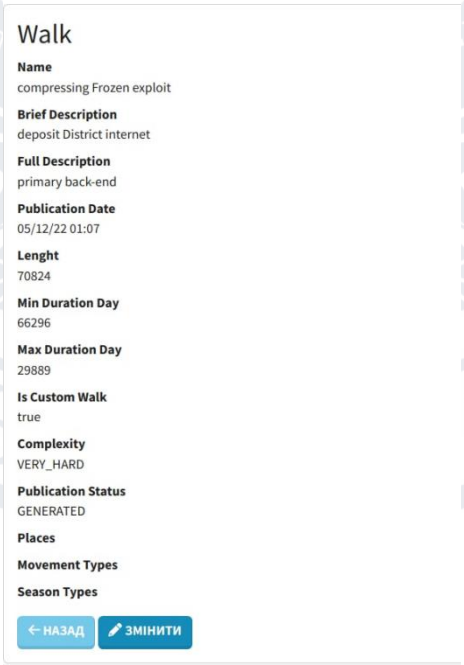


Рисунок 3.4 – Детальний опис маршруту

описом(рис. 3.4). Оскільки ми здійснюємо перегляд в якості анонімного користувача, більшість функціоналу є прихованою. Для неавторизованих користувачів доступною є реєстрація у системі(рис. 3.5).

Рисунок 3.5 – Вікно для реєстрації нових користувачів

Здійснимо авторизацію користувачем, для якого встановлено роль User. Для цього варто у спливаючому вікні(рис. 3.6) ввести відповідну інформацію.

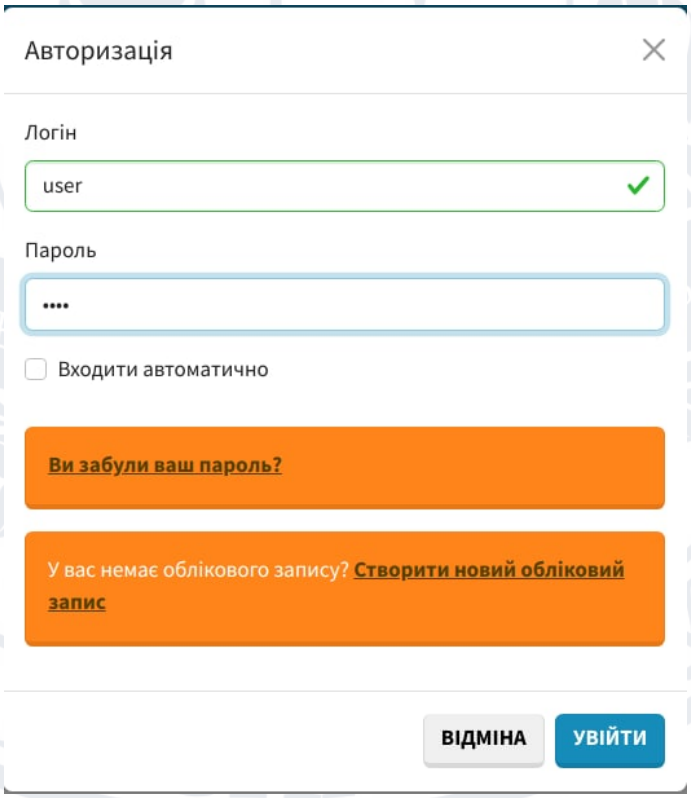


Рисунок 3.6 – Вікно для авторизації користувачів

Також, при розробці було реалізовано підтримку декількох мов, а саме української та англійської. Змінити мову можна у правому верхньому куті користувацького інтерфейсу(рис. 3.7).

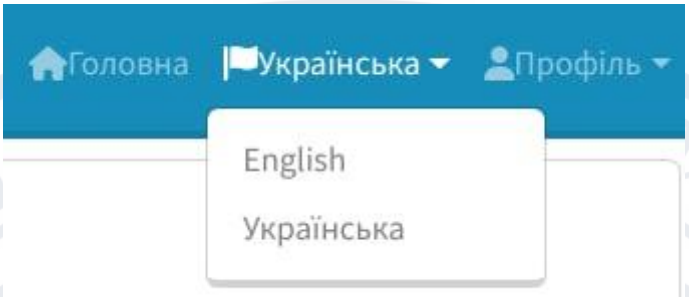


Рисунок 3.7 – Зміна мови користувацького інтерфейсу

Після здійснення авторизації стартовий екран буде таким же, як і у анонімного користувача. Проте, у верхні частині сайту буде доступне меню, що значно розширює можливості користувача. А саме, пункти із меню, які дозволяють перемикатися між різними екранами. Наприклад, наприклад щоб переглядати маршрути(рис. 3.8), необхідно натиснути “Walk” на підменю.

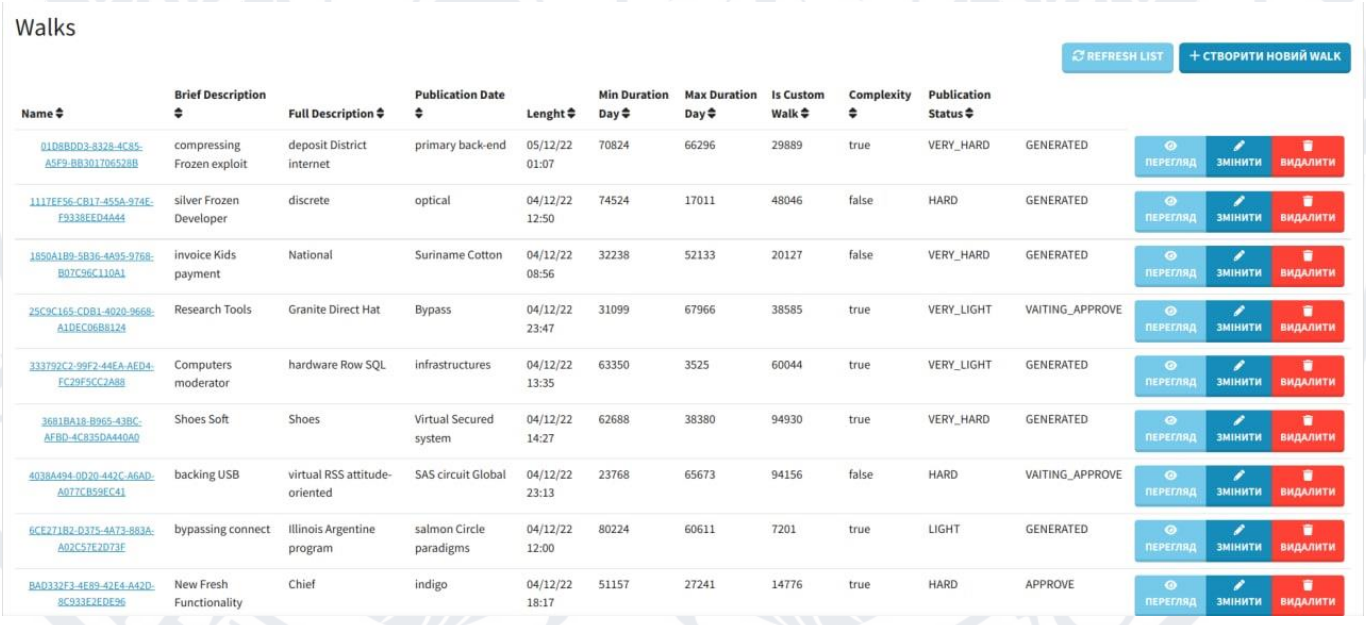


Рисунок 3.8 – Перегляд туристичних маршрутів

Для користувача доступні для перегляду усі маршрути, проте редагувати та видаляти користувач може лише маршрути автором яких є він сам. Окрім базових доступів, для користувача передбачається налаштування його власного аккаунту(рис. 3.9). Для цього слід перейти у пункт налаштування.

Користувачеві буде запропоновано змінити ім’я або прізвище, а також можливо змінити електронну пошту, або мову системи.

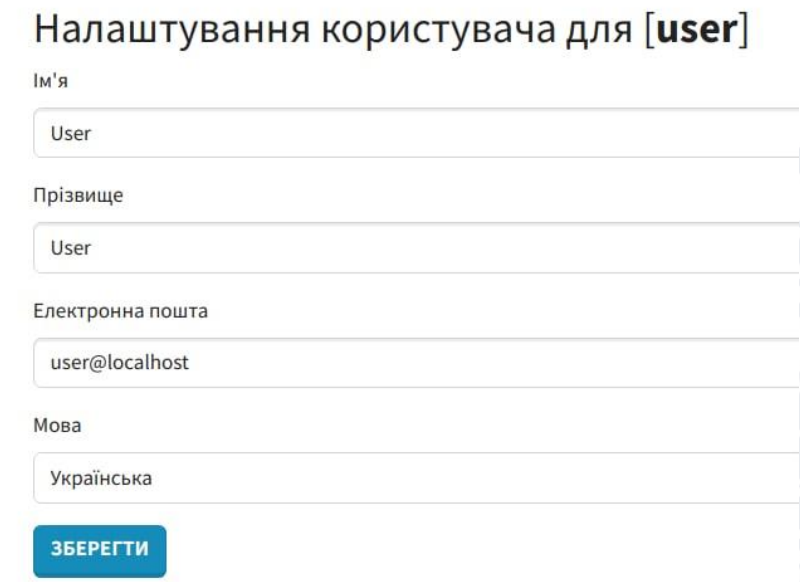


Рисунок 3.9 – Налаштування аккаунту користувача

3.3 Опис адміністративної частини додатку

Для користувачів із роллю ADMIN(адміністратор ресурсу) передбачається більш широкий доступ до системи. Відразу після авторизації у правому верхньому куті з’явиться підменю «Адміністрування»(рис. 3.10), що недоступне для звичайних користувачів.

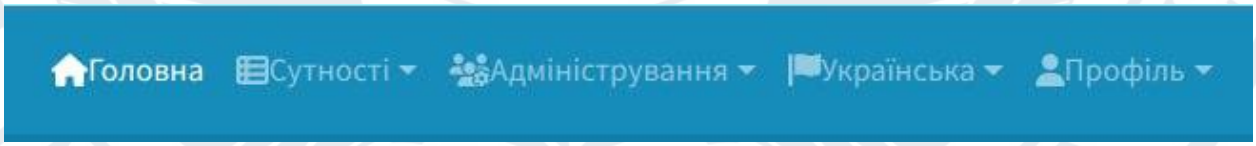


Рисунок 3.10 – Доступне меню для адміністратора ресурсу

Окрім даного підменю, для адміністраторів доступний перегляд усіх створених сутностей із урахуванням системних атрибутів(рис. 3.11), таких як унікальний ідентифікатор, хто та коли редагував останній раз допис, а також чи є допис активним та не видаленим.

Щоб спростити управління користувачами було розроблено відповідний користувацький інтерфейс(рис. 3.12), що надає змогу здійснювати певні маніпуляції із користувацькими аккаунтами.

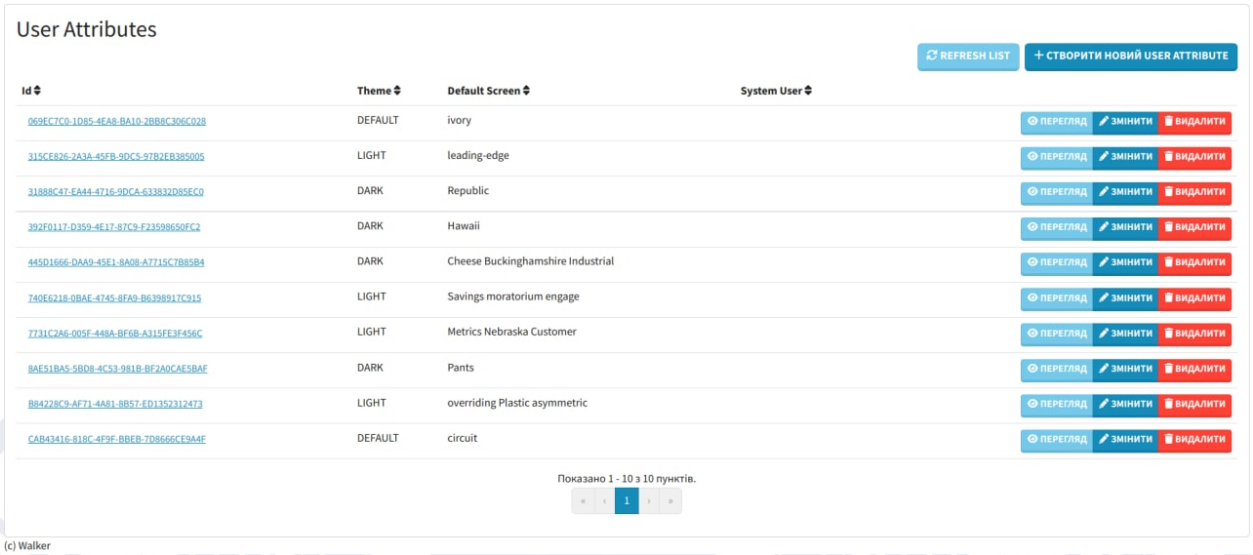


Рисунок 3.11 – Відображення системних даних для адміністратора

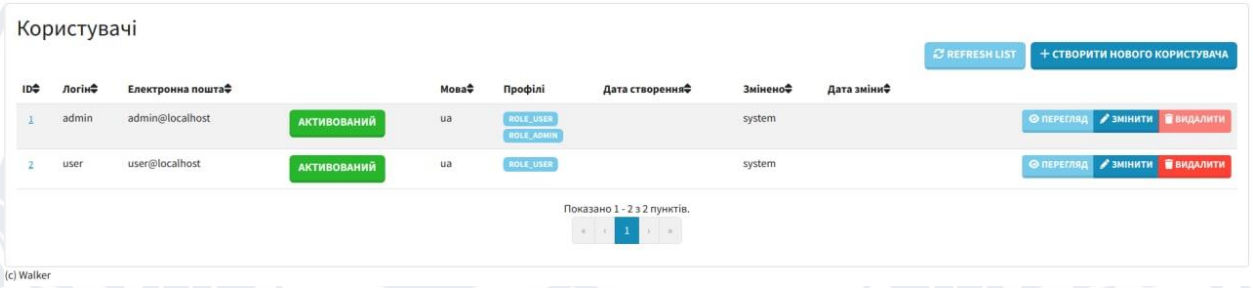


Рисунок 3.12 – Інтерфейс управління користувачами

Оскільки система працює у поєднанні із декількома іншими системами, варто перевіряти чи є активні з’єднання. Також, це дозволяє швидше встановити причину неполадок у системі. Саме із цією метою було реалізовано меню «Стан»(рис. 3.13).

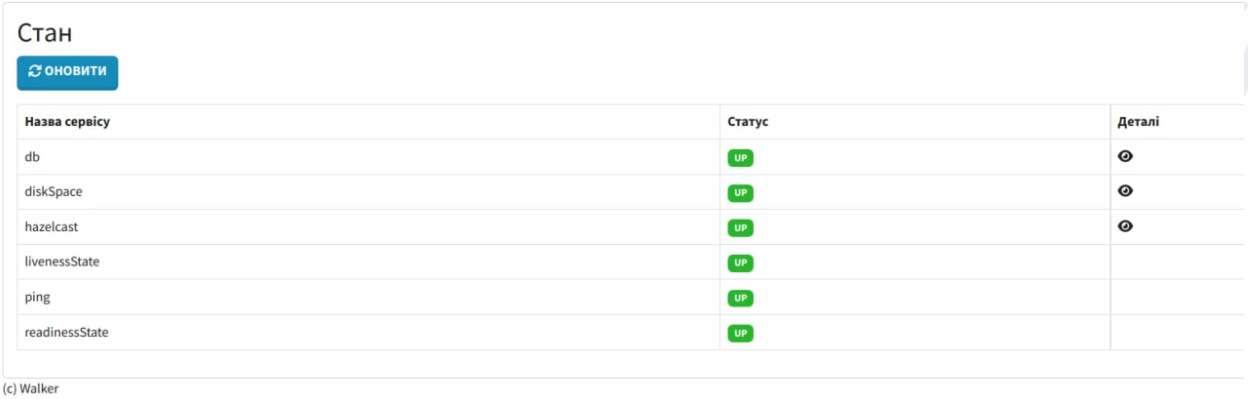


Рисунок 3.13 – Перегляд стану інтегрованих систем

Окрім контролю за зовнішніми системами, варто реалізувати детальний огляд використання ресурсів та проведення аналізу метрик власної системи. Для цього було реалізовано одразу декілька функціональних можливостей.

Для моніторингу стану JVM, було створено меню «Метрики додатку», при перегляді яких можна в режимі реального часу переглянути стан автоматичного очищення пам’яті, потоки які були створені та вираховування кількості потоків в залежності від стану, а також споживання системних ресурсів(рис. 3.14).

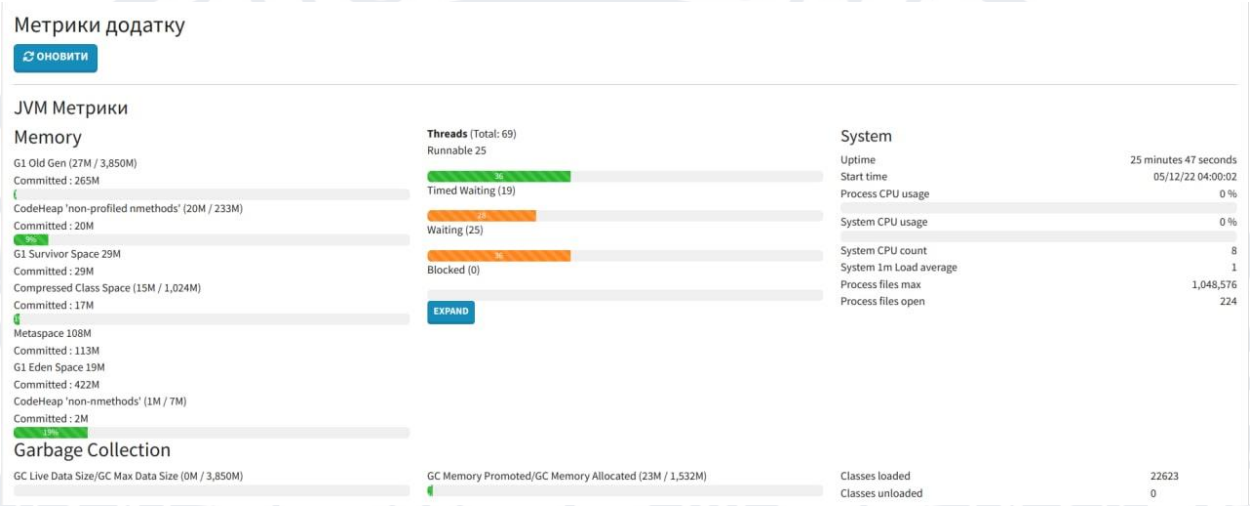


Рисунок 3.14 – Інтерфейс перегляду стану JVM

Окрім моніторингу JVM, здійснюється моніторинг за http запитами(рис. 3.15). Здійснюється відображення кількості запитів в залежності від коду відповіді, а також підраховується кількість запитів на кожний url запит.

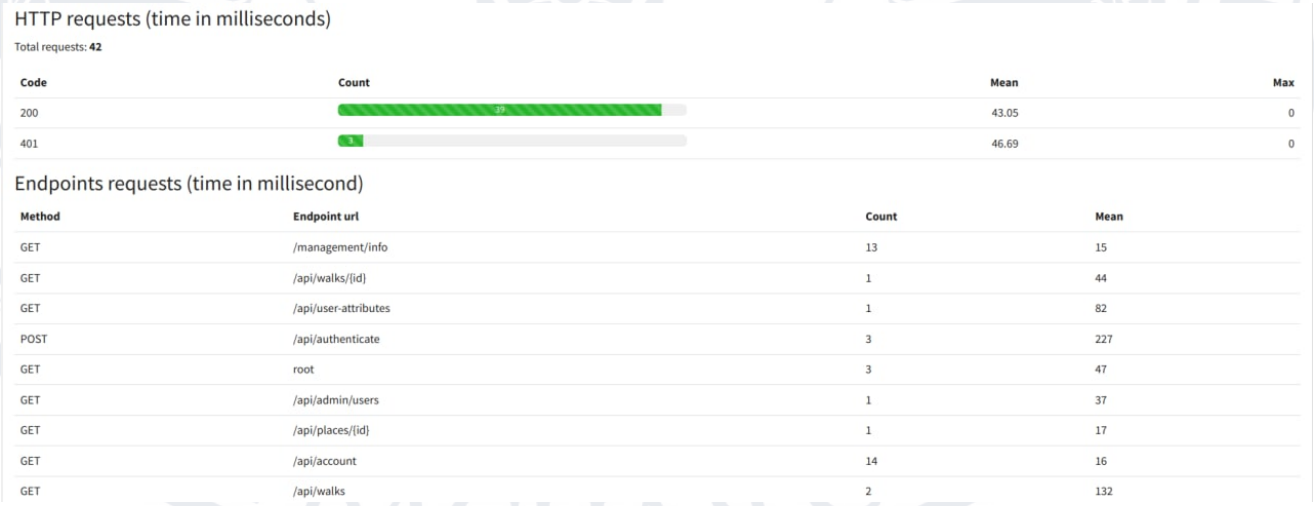


Рисунок 3.15 – Інтерфейс перегляду стану http запитів

Щоб спростити перегляд системних налаштувань у реальному часі, було створено меню «Системні параметри». Перейшовши в дане меню, адміністратор зможе переглянути усі системні налаштування(рис. 3.16), проте здійснювати редагування неможливо, щоб не порушувати функціонування системи.

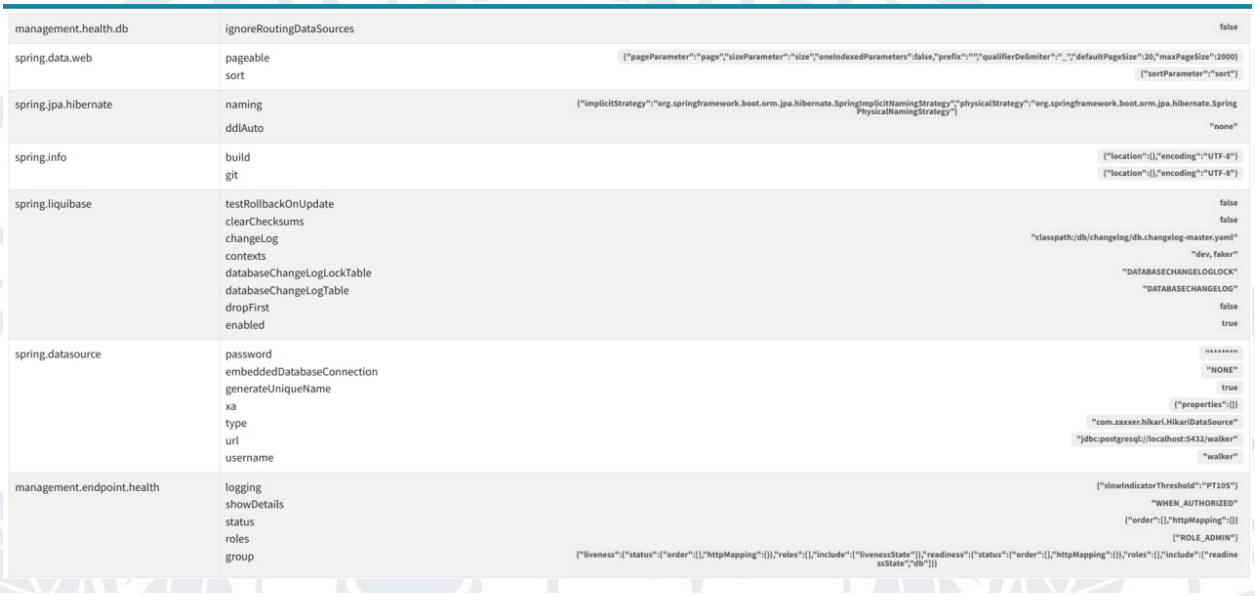


Рисунок 3.16 – Інтерфейс перегляду системних параметрів

Використовуючи користувацький інтерфейс, можливо змінити рівень збору метрик. Для цього необхідно перейти у відповідне меню, де можливо встановити відповідний рівень(рис. 3.17) в залежності від бібліотеки.

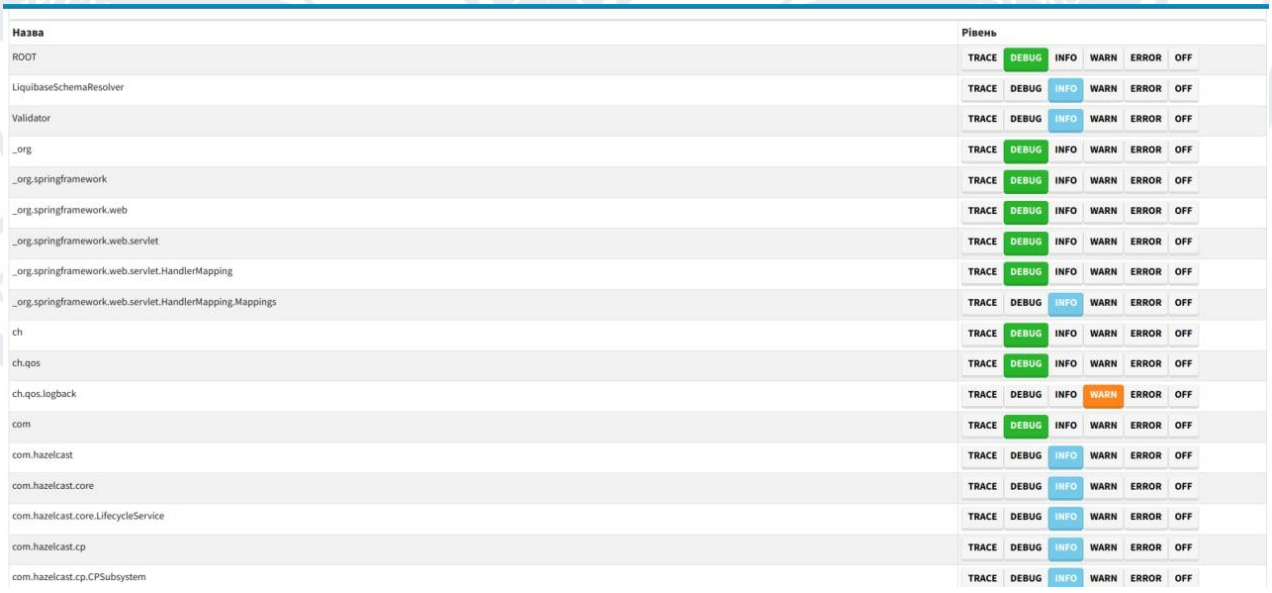


Рисунок 3.17 – Інтерфейс перегляду рівня збору метрик

Не менш важливим, при розробці користувацького інтерфейсу є наявність зручної та інформативної документації по усім кінцевим точкам серверної частини. Для вирішення даної задачі було розроблено відповідне меню(рис. 3.18).

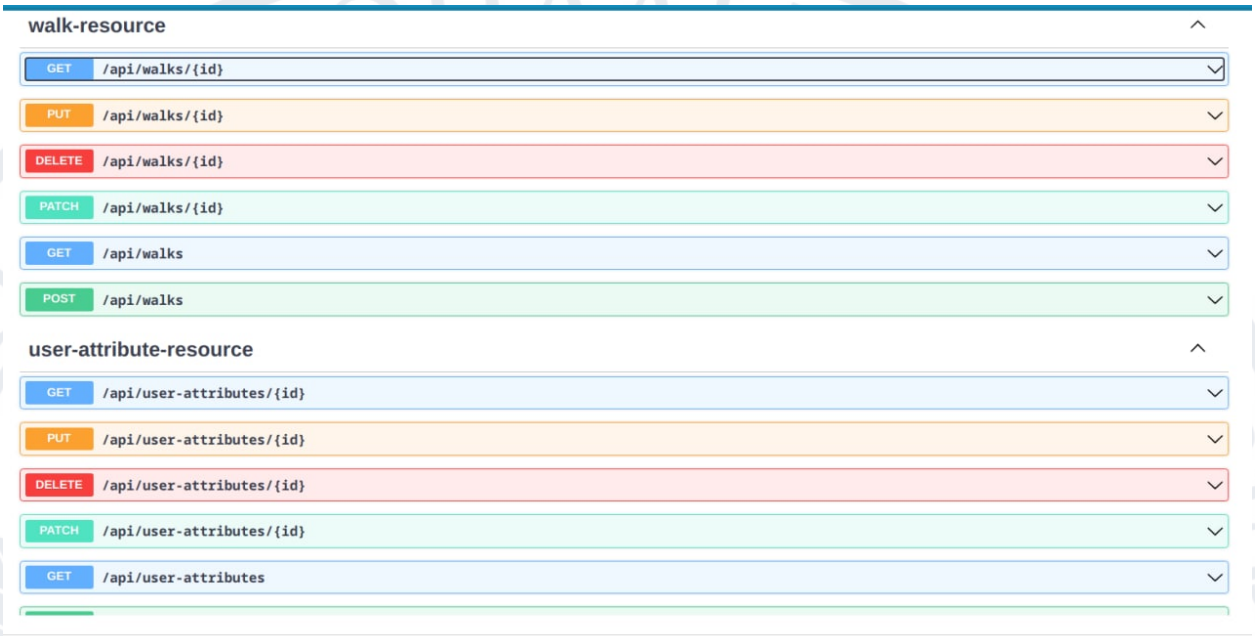


Рисунок 3.18 – Інтерфейс перегляду REST API

При натисканні на відповідну кінцеву точку буде відображено більш детальну інформацію(рис. 3.19). А саме: тип запиту, можливі коди відповідей, а також список необхідних параметрів.

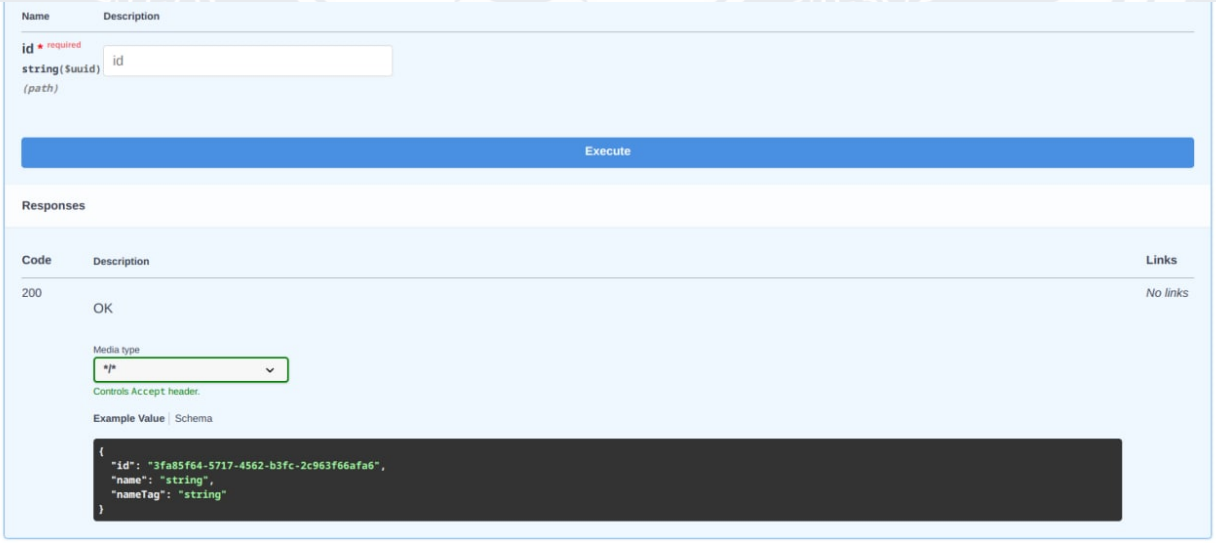


Рисунок 3.19 – Детальний перегляд інформації необхідної для взаємодії із кінцевою точкою

3.4 Висновок до розділу 3

У даному розділі було визначено стратегію ведення git-гілок, етапи та вимоги до впровадження CI/CD системи. Також було детально розглянуто користувацький інтерфейс для трьох типів користувачів: неавторизованого користувача, користувача із роллю USER та адміністратора ресурсу.

ВИСНОВКИ

В поточні кваліфікаційні роботі було розроблено систему автоматичної побудови та агрегації туристичних маршрутів. Сама система представлена у вигляді web-додатку і дозволяє виконувати побудову туристичного маршруту враховуючи ряд критерії та туристичних місць, які бажає відвідати турист. Було проаналізовано аналоги даної системи, визначено переваги та недоліки. Це дозволило створити відмовостійку та просту у підтримці систему, яка містить більшість переваг інших систем, а також враховує досвід недоліків.

Розроблений web-додаток надає наступний функціонал для анонімних користувачів:

- перегляд мапи маршрутів;
- перегляд існуючих маршрутів;
- перегляд блогів;
- перегляд коментарів.

Для авторизованих користувачів доступно наступне:

- можливість автоматичної генерації маршруту;
- збереження згенерованого маршруту;
- ведення блогу;
- написання коментарів;
- створення рекомендованих маршрутів.

Створена система може бути розгорнута як повноцінний web-сервіс, який надає змогу покращити планування майбутніх подорожей.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Tourism | Definition, History, Types, Importance, & Industry. URL: https://www.britannica.com/topic/tourism (дата звернення: 16.06.2022)
2. ст. 4 Закону України «Про туризм». URL: https://zakon.rada.gov.ua/laws/show/324/95-%D0%B2%D1%80#Text (дата звернення: 17.09.2022)
3. What is automation? – IBM. URL: https://www.ibm.com/topics/automation (дата звернення: 03.09.2022)
4. Automated System Operations (ASO). URL: https://www.techopedia.com/definition/31065/automated-system-operations-aso (дата звернення: 16.10.2022)
5. Google подорожі. URL: https://www.google.com/travel/?dest_src=al&authuser=0 (дата звернення: 10.06.2022)
6. Wishtrip. URL: https://www.wishtrip.com/home (дата звернення: 11.09.2022)
7. Triplt. URL: https://www.tripit.com/web (дата звернення 11.09.2022)
8. Sygic Travel Trip Planner. URL: https://travel.sygic.com/en (дата звернення: 12.11.2022)
9. International arrivals by world region. URL: https://ourworldindata.org/tourism (дата звернення 26.11.2022)
10. Л. А. Бондаренко(2016), Міжнародний туризм в україні: проблеми та перспективи подальшого розвитку. Ефективна економіка, 11
11. Robert M. Clean Architecture. Pearson; 1st edition (September 10, 2017). 432 pages.
12. Toby J.T., Sam S.L., Tom N. Database Modeling and Design: Logical Design (The Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann (February 24, 2011). 352 pages.
13. Candra Y.S. Introduction To Client Sever Computing. New Delhi: New age international(P) limited, publishers, 213 page.

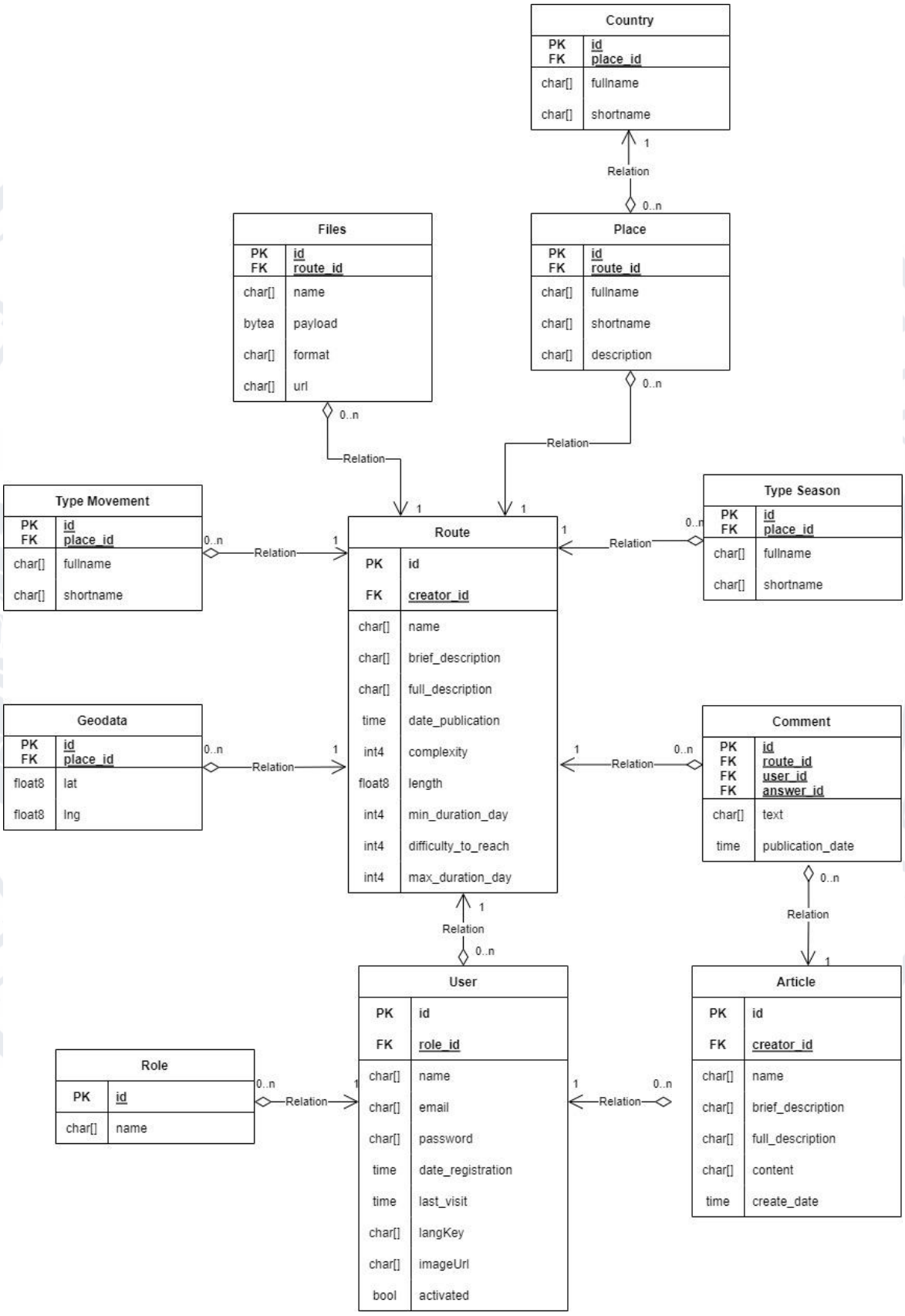
14. Ryan V., Muhammad S.S. MVVM Survival Guide for Enterprise Architectures in Silverlight and WPF. Birmingham B3 2PB, 490 page.
15. Peter W. Object-Oriented and Mixed Programming Paradigms. Sintra (May 9-11, 1994). 198 pages.
16. What is object-oriented programming? OOP explained in depth. URL: https://www.educative.io/blog/object-oriented-programming (дата звернення 11.10.2022)
17. Abhishek N., Manisha B. Reinforcement Learning: With Open AI, TensorFlow and Keras Using Python. West Bengal(2018), 165 pages.
18. In-depth guide to reinforcement learning. URL: https://www.techtarget.com/searchenterpriseai/definition/reinforcement-learning (дата звернення: 15.10.2022)
19. Reinforcement learning. URL: https://www.geeksforgeeks.org/what-is-reinforcement-learning/ (дата звернення: 03.09.2022)
20. Marco D., Thomas S. Ant Colony Optimization. London (2004). 319 pages.
21. Stefka F. Ant Colony Optimization and Applications. Sofia (2021). 124 pages.
22. Introductory Chapter: Ant Colony Optimization. . URL: https://www.intechopen.com/chapters/81065 (дата звернення: 07.09.2022)
23. Korry D., Susan D. PostgreSQL: A Comprehensive Guide to Building, Programming and Administrating databases. Indianapolis (February 2003). 244 pages.
24. pg_dump PostgreSQL Client Applications. URL: https://www.postgresql.org/docs/current/app-pgdump.html (дата звернення: 17.11.2022)
25. Setting Parameters. URL: https://www.postgresql.org/docs/current/config-setting.html (дата звернення: 17.11.2022)
26. pgAdmin 4 6.17 documentation. URL: https://www.pgadmin.org/docs/pgadmin4/development/index.html (дата звернення: 11.11.2022)

27. The config.py File. URL: https://www.pgadmin.org/docs/pgadmin4/development/config_py.html (дата звернення: 12.10.2022)
28. Herbert S. Java The Complete Reference, 8th Edition. McGraw Hill; 8th edition (February 7, 2011). 1152 pages.
29. TIOBE Index for December 2022 https://www.tiobe.com/tiobe-index/м (дата звернення: 29.09.2022)
30. What is Java Used For? URL: https://w. URL:ww.javatpoint.com/what-is-java-used-for (дата звернення: 03.10.2022)
31. Iuliana C., Rob H., Chris S. Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools. Apress; 5th edition (October 11, 2017). 1717 pages.
32. Spring core, data, web, testing. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/ (дата звернення: 27.10.2022)
33. Craig W. Spring Boot in Action. Manning; 1st edition (January 3, 2016). 264 pages.
34. Spring Boot Reference Documentation. URL: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ (дата звернення: 07.10.2022)
35. Prometheus overview. URL: https://prometheus.io/docs/introduction/overview/ (дата звернення: 13.11.2022)
36. Prometheus INSTALLATION. URL: https://prometheus.io/docs/prometheus/latest/installation/ (дата звернення: 13.11.2022)
37. Eric S. Learn Grafana 7.0: A beginner's guide to getting well versed in analytics, interactive dashboards, and monitoring. Packt Publishing (June 25, 2020). 410 pages.\
38. Configure field overrides. URL: https://grafana.com/docs/grafana/latest/panels-visualizations/configure-overrides/ (дата звернення: 29.11.2022)

39. Hazelcast Documentation. URL: https://docs.hazelcast.com/home/ (дата звернення: 29.11.2022)
40. SonarQube Documentation. URL: https://docs.sonarqube.org/latest/ (дата звернення: 16.11.2022)
41. Mark T. React in Action. Manning; 1st edition (July 8, 2018). 360 pages.
42. Traefik Proxy Documentation. URL: https://doc.traefik.io/traefik/ (дата звернення: 26.10.2022)
43. Jeff N., Stephen K. Docker in Action. Manning; 2nd edition (December 10, 2019). 350 pages.
44. Overview Docker compose. URL: https://docs.docker.com/compose/ (дата звернення: 09.10.2022)
45. Git V: An Optimal Git Branching Model. URL: https://mergebase.com/blog/git-v-branching-model/ (дата звернення: 08.11.2022)
46. What is Continuous Integration. URL: https://www.atlassian.com/continuous-delivery/continuous-integration (дата звернення: 08.11.2022)
47. CI/CD vs. DevOps: Understanding 8 Key Differences. URL: https://www.spiceworks.com/tech/devops/articles/cicd-vs-devops/ (дата звернення: 22.10.2022)
48. CI/CD explained. URL: https://about.gitlab.com/topics/ci-cd/ (дата звернення: 06.09.2022)
49. Continuous Deployment via GitLab, Jenkins, Docker and Slack. URL: https://medium.com/@ahmetatalay/continous-deployment-via-gitlab-jenkins-docker-and-slack-5d08836d01e0 (дата звернення: 22.10.2022)

ДОДАТОК А

Структура бази даних.



ДОДАТОК Б

Лістинг файлу налаштувань для Docker Compose

```
version: '3.3'

services:

  traefik:
    image: "traefik:v2.7"
    restart: unless-stopped
    command:
      - "--log.level=${TRAEFIK_LOG_LEVEL}"
      - "--api.insecure=false"
      - "--api.dashboard=true"
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
      - "--entrypoints.web.address=:${TRAEFIK_WEB_ADDRESS}"
      - "--entrypoints.web.http.redirections.entrypoint.to=websecure"
      - "--entrypoints.web.http.redirections.entrypoint.scheme=https"
      - "--entrypoints.websecure.address=:${PORT}"
      - "--certificatesresolvers.myresolver.acme.httpchallenge=true"
      - "--certificatesresolvers.myresolver.acme.httpchallenge.entrypoint=web"
      - "--certificatesresolvers.myresolver.acme.storage=/letsencrypt/acme.json"
    ports:
      - ${TRAEFIK_WEB_ADDRESS}:${TRAEFIK_WEB_ADDRESS}
      - ${PORT}:${PORT}
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock:ro"
      - /letsencrypt:/letsencrypt:rw
    networks:
```

```
    default:
      aliases:
        - ${HOST}

db:
  image: timescale/timescaledb:2.8.0-pg14
  restart: unless-stopped
  environment:
    - POSTGRES_USER=${DB_USER}
    - POSTGRES_PASSWORD=${DB_PASS}
    - POSTGRES_HOST_AUTH_METHOD=trust
    - PGDATA=${PG_DATA}

pgadmin4:
  image: dpage/pgadmin4
  restart: unless-stopped
  volumes:
    - /var/pgadmin4-data:/var/lib/pgadmin
  environment:
    - PGADMIN_DEFAULT_EMAIL=${PGADMIN_USER}
    - PGADMIN_DEFAULT_PASSWORD=${PGADMIN_PASS}
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.pgadmin4.rule=Host(`pga.${HOST}`)"
    - "traefik.http.routers.pgadmin4.entrypoints=${TRAEFIK_ENTRYPOINT}"
    - "traefik.http.routers.pgadmin4.tls.certresolver=myresolver"

prometheus:
  image: prom/prometheus:v2.38.0
  restart: unless-stopped
```

```
    volumes:
      - ./prometheus/:/etc/prometheus/
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
    expose:
      - "9090"

  grafana:
    image: grafana/grafana:9.1.0
    restart: unless-stopped
    depends_on:
      - db
      - client
    volumes:
      - /var/grafana-data:/var/lib/grafana
    environment:
      -
GF_SERVER_ROOT_URL=${SCHEME}://${GRAFANA_SUB_DOMAIN}.${
HOST}:${PORT}

      - GF_LOG_MODE=console
      - GF_LOG_LEVEL=debug
      - GF_LOG_FILTERS=alerting.scheduler:info migrator:info

      - GF_SECURITY_ALLOW_EMBEDDING=true
      - GF_SECURITY_ADMIN_USER=${GRAFANA_SUPERADMIN}
      - GF_SECURITY_ADMIN_PASSWORD=${GRAFANA_PASSWORD}

      - GF_AUTH_BASIC_ENABLED=true
      - GF_AUTH_DISABLE_LOGIN_FORM=true
```

```
    - GF_AUTH_OAUTH_AUTO_LOGIN=true
    - GF_AUTH_DISABLE_SIGNOUT_MENU=true
    - GF_SECURITY_ADMIN_PASSWORD=admin
    - GF_USERS_ALLOW_SIGN_UP=false
    - GF_INSTALL_PLUGINS=grafana-piechart-panel

  labels:
    - "traefik.enable=true"
    -
"traefik.http.routers.grafana.rule=Host(`${GRAFANA_SUB_DOMAIN}.${HOST
}`)"
    - "traefik.http.routers.grafana.entrypoints=${TRAEFIK_ENTRYPOINT}"
    - "traefik.http.routers.grafana.tls.certresolver=myresolver"

  networks:
   default:
    aliases:
     - ${GRAFANA_SUB_DOMAIN}.${HOST}

 hazelcast:
  image: hazelcast/management-center:5.1.4
  restart: unless-stopped
  ports:
   - 8180:8080

sonar:
  image: sonarqube:9.6.0-community
  restart: unless-stopped
  environment:
   - sonar.forceAuthentication=false
```

```
    expose:
      - 9080

  server:
    image: server:0.0.5
    restart: unless-stopped
    environment:
      - _JAVA_OPTIONS=-Xmx512m -Xms256m
      - SPRING_PROFILES_ACTIVE=prod,api-docs
      -
PROMETHEUS_METRICS_EXPORT_ENABLED=${PROMETHEUS_METRI
CS_EXPORT_ENABLED}
      - DB_HOST=${DB_HOST}
      - DB_PORT=${DB_PORT}
      - DB_USER=${DB_USER}
      - DB_PASS=${DB_PASS}
      - DB_NAME=${DB_NAME}
      - DB_JAVA_DRIVER=${DB_JAVA_DRIVER}
      - LIQUIBASE_CONTEXTS=${LIQUIBASE_CONTEXTS}
      - SMTP_HOST=${SMTP_HOST}
      - SMTP_PORT=${SMTP_PORT}
      - SMTP_USER=${SMTP_USER}
      - SMTP_PASS=${SMTP_PASS}
      - BACKEND_PORT=${BACKEND_PORT}
      -
HAZELCAST_CACHE_TIME_TO_LIVE_IN_DAYS=${HAZELCAST_CACHE
_TIME_TO_LIVE_IN_DAYS}
      -
HAZELCAST_CACHE_TIME_TO_LIVE_SECONDS=${HAZELCAST_CACH
E_TIME_TO_LIVE_SECONDS}
```

```
    -
HAZELCAST_CACHE_BACKUP_COUNT=${HAZELCAST_CACHE_BACK
UP_COUNT}
    - JWT_BASE_64_SECRET=${JWT_BASE_64_SECRET}
    -
JWT_TOKEN_VALIDITY_IN_SECOND=${JWT_TOKEN_VALIDITY_IN_SE
COND}
    -
JWT_TOKEN_VALIDITY_IN_SECOND_FOR_REMEMBER_ME=${JWT_TO
KEN_VALIDITY_IN_SECOND_FOR_REMEMBER_ME}
    - LOGSTASH_ENABLED=${LOGSTASH_ENABLED}
    - LOGSTASH_HOST=${LOGSTASH_HOST}
    - LOGSTASH_PORT=${LOGSTASH_PORT}
    - LOGSTASH_QUEUE_SIZE=${LOGSTASH_QUEUE_SIZE}
  ports:
    - 8085:8080
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.server.rule=Host(`${HOST}`) && PathPrefix(`/api`)"
    - "traefik.http.routers.server.entrypoints=${TRAEFIK_ENTRYPOINT}"
    - "traefik.http.routers.server.tls.certresolver=myresolver"
    - "traefik.http.services.server.loadbalancer.server.port=8085"
  depends_on:
    - db

client:
  image: client:0.0.7
  restart: unless-stopped
  environment:
    - SERVER_API_URL=${SERVER_API_URL}
```

```
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.client.rule=Host(`${HOST}`)"
      - "traefik.http.routers.client.entrypoints=${TRAEFIK_ENTRYPOINT}"
      - "traefik.http.routers.client.tls.certresolver=myresolver"
      - "traefik.http.services.client.loadbalancer.server.port=80"
    depends_on:
      - server
      - traefik

networks:
  default:
```

Мазурук Олег Володимирович
Прізвище, ім’я, по батькові

Інформаційних і прикладних технологій
Факультет

122 «Комп’ютерні науки»
Шифр і назва спеціальності

«Комп’ютерні технології обробки даних (Data Science)»
Освітня програма

ДЕКЛАРАЦІЯ АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (магістерська) робота на тему: «Дослідження автоматизованої системи агрегації та побудови туристичних маршрутів» є написаною мною особисто.

Одночасно заявляю, що ця робота:

- не передавалась іншим особам і подається до захисту вперше;
- не порушує авторських та суміжних прав, закріплених статтями 21-25 Закону України «Про авторське право та суміжні права»;
- не отримувалась іншими особами, а також дані та інформація не отримувалась у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».

(дата) (підпис здобувача)