КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій
(факультет)

інформаційних технологій
(кафедра)

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему: «Розробка веб-системи для планування подорожей з інтеграцією в Google карти»

ШАНЬКО ВІКТОРІЯ ОЛЕКСАНДРІВНА
(прізвище, ім’я та по батькові студента повністю)

Київ, 2025

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ЗАТВЕРДЖУЮ
Завідувачка кафедри ІТ
д.т.н., професор Гончаренко Т.А.

_____
„___” _____2025 року

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему: «Розробка веб-системи для планування подорожей з інтеграцією в Google карти»

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач Шанько Вікторія Олександрівна
(прізвище, ім’я та по батькові повністю)
122 «Комп’ютерні науки»
(спеціальність)
Інформаційні управляючі системи і технології
(освітня програма)
Групи КН-21-1
Керівник Бородавка Є. В.
(прізвище та ініціали)
д.т.н.,професор
(вчене звання, науковий ступінь)
Рецензент Баліна О.І.
(Прізвище та ініціали)
к.т.н., доцент
(вчене звання, науковий ступінь)

Ідентичність підтверджую

Київ, 2025

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій
Випускова кафедра: інформаційних технологій
Ступінь вищої освіти: «бакалавр»
Спеціальність: 122 «Комп`ютерні науки»
Освітня програма: Інформаційні управляючі системи і технології

ЗАТВЕРДЖУЮ
Завідувачка кафедри ІТ
д.т.н., професор Гончаренко Т.А.

_____
„___” _____2025 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

Шанько Вікторія Олександрівна

1. Тема роботи: Розробка веб-системи для планування подорожей з інтеграцією в Google карти
затверджена наказом ректора КНУБА № 235 від «14» лютого 2025 року
2. Керівник роботи: Бородавка Євгеній Володимирович, д.т.к., професор кафедри інформаційних технологій
3. Термін подання Здобувачем роботи до захисту:_____
4. Зміст пояснювальної записки за розділами:
Р.1. Аналіз предметної області та постановка задачі
Р.2. Проєктування архітектури системи
Р.3. Програмна реалізація та тестування системи
Р.4. Впровадження системи
5. Інформаційні слайди:
С.1. Визначення цілей дослідження
С.2. Архітектурні рішення
С.3. Структура та моделювання бази даних
С.4. Результати тестування алгоритмів
С.5. Інтерфейс користувача

6. Консультанти розділів кваліфікаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта, представника комісії	дата	підпис
Розділ 1	Гончаренко Т. А.	15.02.2025	
Розділ 2	Бородавка Є. В.	28.04.2025	
Розділ 3	Маційовський О. О.	15.05.2025	
Розділ 4	Рябчун Ю. В.	31.05.2025	

7. Календарний план виконання кваліфікаційної випускної роботи

Види робіт та їх зміст	Дата виконання
Розділ 1	15.02.2025
Розділ 2	28.04.2025
Розділ 3	15.05.2025
Розділ 4	31.05.2025
Остаточне оформлення роботи	02.06.2025
Направлення роботи для перевірки на плагіат	03.06.2025
Направлення роботи на рецензування	03.06.2025
Попередній захист роботи на кафедрі	04.06.2025

8. Дата видачі завдання _____14 лютого 2025_____

Зав. кафедри _____ ____Гончаренко Т.А.___
(підпис) (прізвище та ініціали)
Керівник _____ _____Бородавка Є.В._____
(підпис) (прізвище та ініціали)
Здобувач _____ _____Шанько В. О._____
(підпис) (прізвище та ініціали)

<table>
<tr><td>РЕЗЮМЕ (SUMMARY)
до кваліфікаційної випускної роботи Здобувача:</td><td colspan="3">Шанько Вікторія Олександрівна

Viktoriia Shanko</td></tr>
<tr><td>ЗВО</td><td colspan="3">Київський національний університет будівництва і архітектури</td></tr>
<tr><td>Тема (українською та англійською)</td><td colspan="3">Розробка веб-системи для планування подорожей з інтеграцією в Google карти

Development of a web-based travel planning system with integration into Google maps</td></tr>
<tr><td>Освітній ступінь</td><td colspan="3">Бакалавр</td></tr>
<tr><td>Факультет</td><td colspan="3">Автоматизації і інформаційних технологій</td></tr>
<tr><td>Випускова кафедра</td><td colspan="3">Інформаційних технологій</td></tr>
<tr><td>Спеціальність</td><td colspan="3">122 «Комп’ютерні науки»</td></tr>
<tr><td>Освітня програма</td><td colspan="3">Інформаційні управляючі системи та технології</td></tr>
<tr><td>Керівник</td><td colspan="3">Бородавка Євгеній Володимирович</td></tr>
<tr><td rowspan="2">Обсяг роботи:</td><td>пояснювальна записка, стор.</td><td>розділів</td><td>креслень формату А</td></tr>
<tr><td>117</td><td>4</td><td>0</td></tr>
<tr><td>Розділ 1.</td><td colspan="3">Аналіз предметної області та постановка задачі</td></tr>
<tr><td>Розділ 2.</td><td colspan="3">Проєктування архітектури системи</td></tr>
<tr><td>Розділ 3.</td><td colspan="3">Програмна реалізація системи</td></tr>
<tr><td>Розділ 4.</td><td colspan="3">Впровадження системи</td></tr>
<tr><td>Ключові слова:

Keywords:</td><td colspan="3">веб-система, прокладання маршрутів, інтеграції, сервер, база даних, HTTP – запити, Google API, користувач, інтерфейс, планування подорожей, хмарні сервіси.

web system, route planning, integrations, server, database, HTTP-requests, Google API, user, interface, travel planning, cloud services.</td></tr>
</table>

Здобувач: _____ /Вікторія ШАНЬКО /

Керівник: _____/ Євгеній БОРОДАВКА /

“___” _____ 202_р.

АНОТАЦІЯ

Шанько В. О. Розробка веб-системи для планування подорожей з інтеграцією в Google карти.

Кваліфікаційна випускна робота бакалавра за спеціальністю: 122 «Комп'ютерні науки», освітньо-професійна програма: «Інформаційні управляючі системи і технології». – Київський національний університет будівництва та архітектури. – Київ, 2025.

Робота присвячена розробці веб-системи для планування подорожей з інтеграцією сторонніх сервісів за допомогою Google API. Описано алгоритми взаємодії з інтегрованими сервісами для побудови маршрутів, зберігання подій в Google Calendar, безпечної авторизації користувачів з використанням HTTP – запитів. Тестування алгоритмів підтвердило їх ефективність та надійність.

Ключові слова: веб-система, прокладання маршрутів, інтеграції, сервер, база даних, HTTP – запити, Google API, користувач, інтерфейс, планування подорожей, хмарні сервіси.

SUMMARY

Shanko V. O. Development of a web-based travel planning system with integration into Google maps.

Bachelor's thesis for a bachelor's degree in specialty: 122 “Computer Science”, specialization: "Information Management Systems and Technologies" - Kyiv National University of Construction and Architecture - Kyiv, 2025.

The work is devoted to the development of a web-based travel planning system with the integration of third-party services using Google API. The algorithms of interaction with integrated services for building routes, storing events in Google Calendar, and secure user authorization using HTTP requests are described. Testing of the algorithms has confirmed their efficiency and reliability.

Keywords: web system, route planning, integrations, server, database, HTTP-requests, Google API, user, interface, travel planning, cloud services.

ЗМІСТ

ВСТУП

Туристичний бізнес стрімко розвивається в сучасних реаліях використовуючи цифрові технології, зокрема в Інтернет просторі. Така тенденція змінила підхід до планування подорожей як у мандрівників, так і у компаній, які організовують різні туристичні заходи. Наразі такі дії як: огляд місць, бронювання квитків, готелів, прокладання маршрутів; не потребують додаткових зусиль, а тільки стабільного інтернет-підключення та доступу до відповідних сервісів. Зокрема, розвиток веб-систем став необхідним для цієї галузі. Мандрівники потребують зручних інструментів для планування поїздок, зрозумілого інтерфейсу та легкого оформлення всіх необхідних даних, які їм знадобляться в поїдці.

Актуальність розробки веб-системи для планування подорожей з інтеграцією в Google карти обумовлення зростанням попиту на використання веб-технологій у сфері туризму. Дана система повинна враховувати потреби користувачів на момент розробки, тенденції ринку, необхідні інтеграції для планування подорожі, зручність у використанні.

Метою роботи є *розробка моделі веб-системи для зручного планування подорожей з використанням інтеграцій сервісів Google.*

Об’єктом дослідження є *веб-система для планування подорожей*

Предметом дослідження є методи та технології, що використовують для розробки веб-систем та впровадження інтеграцій

Основні методи дослідження:

- Аналіз існуючих веб-систем та рішень
- Аналіз алгоритмів пошуку шляху
- Реалізація інтеграцій
- Тестування реалізованих алгоритмів

Робота передбачає створення моделі системи, визначення вимог, проектування архітектури системи, моделювання бази даних та реалізацію основних алгоритмів для забезпечення визначеного функціоналу.

Перший розділ включає в себе аналіз популярних додатків для подорожей, розглянуто їх функціонал та інтерфейс, визначено плюси та недоліки. Також розглянуто алгоритми створення маршрутів, які використовуються у Google картах.

У другому розділі визначено та повністю описано основні вимоги до системи. Проведено архітектурне проектування, що включало визначення підсистем системи, їх модулі та функції. Побудовано діаграми, які відображають роботу системи. Також визначено тип бази даних та виконано моделювання даних.

Третій розділ описує програмну реалізацію системи. Обрано стек технологій та середовище розробки. Детально розглянуто роботу та виконано тестування необхідних алгоритмів для правильного функціонування системи.

У четвертому розділі розраховано необхідний мінімум ресурсів для впровадження системи на ринку. Обрані сервіси для розміщення системи та бази даних, проаналізовано проблеми з якими можна стикнутись при розробці продукту. Також детально розглянуто інтерфейс користувача, задля забезпечення отримання гарного користувацького досвіду та зручного використання системи.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Постановка й аналіз проблеми

З кожним роком все більше людей починають подорожувати світом. Мандрівники ретельно планують свої пригоди і з розвитком інформаційних технологій знаходять для себе корисні додатки для їхньої майбутньої подорожі. Однак, попри вже існуючі на ринку продукти, їх все одно не достатньо для сучасного туриста.

Світові компанії представляють свої продукти щоб полегшити планування подорожей. В цю категорію входять карти, сервіси бронювання, календарі та інші, але мало існує додатків які містили б в собі функціонал раніше наведених продуктів.

Актуальність розробки такої системи зумовлена зростанням кількості мандрівників. Згідно зі статистикою компаній, що надають послуги турів та організації відпочинку за кордоном та в межах країни для українців, навіть в умовах війни частина людей, що подорожують значно збільшилась (рис. 1.1.)

Рис. 1.1. Попит на подорожі 2023

Також компанія провела аналіз які формати поїздок потребують українці на 2023 рік (рис. 1.2.). За їх інформацією значну частину мандрівників складають

жінки та діти, саме ця аудиторія подорожує за кордон, а чоловіча частина в поставлених умовах подорожує рідними землями.

Рис. 1.2. Затребувані формати відпочинку

На рахунок міжнародного туризму теж є великі прогреси у кількості мандрівників. На платформі UN Tourism Tracker наведено чіткий графік звітності щодо показників туризму (рис. 1.3.). За їх даними був різкий занепад туризму у 2020 році, через поширення COVID-19, але вже у 2022 році туризм набув пікового зростання, попри на разі ситуація більш стабільна та немає великих відхилень статистики. Також визначено, що у 2024 році кількість туристів, які мають міжнародні подорожі збільшилась на 11%. Загальний показник, який оцінює готовність людей до подорожей становить 39, це свідчить про поступове зростання попиту у сфері туризму. Дані щодо цього показника збираються через різні опитування у соціальних мережах та кількості пошукових запитів щодо подорожей. [19-21]

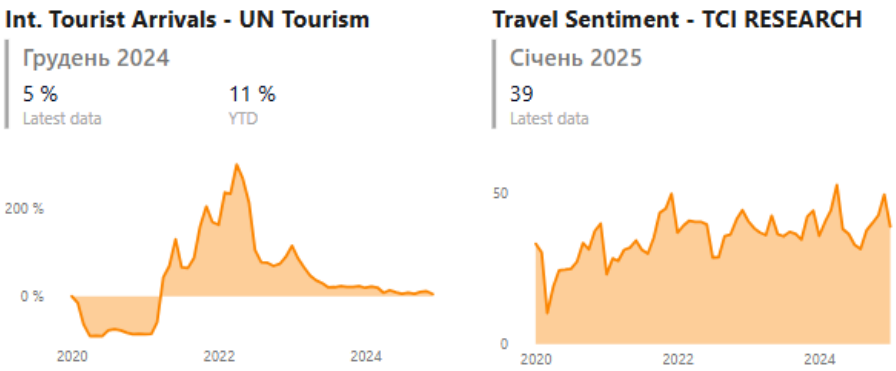

Рис. 1.3. Ключові показники міжнародного туризму

Застосування технологій серед мандрівників теж не стоїть на місці. Компанія Phocuswright займається аналізом туристичної індустрії і з їх останніх досліджень “Travelers and Tech 2024: Attitudes and Usage” та показують як швидко мандрівники впроваджують нещодавно випущені технології у свої подорожі (рис. 1.4.).

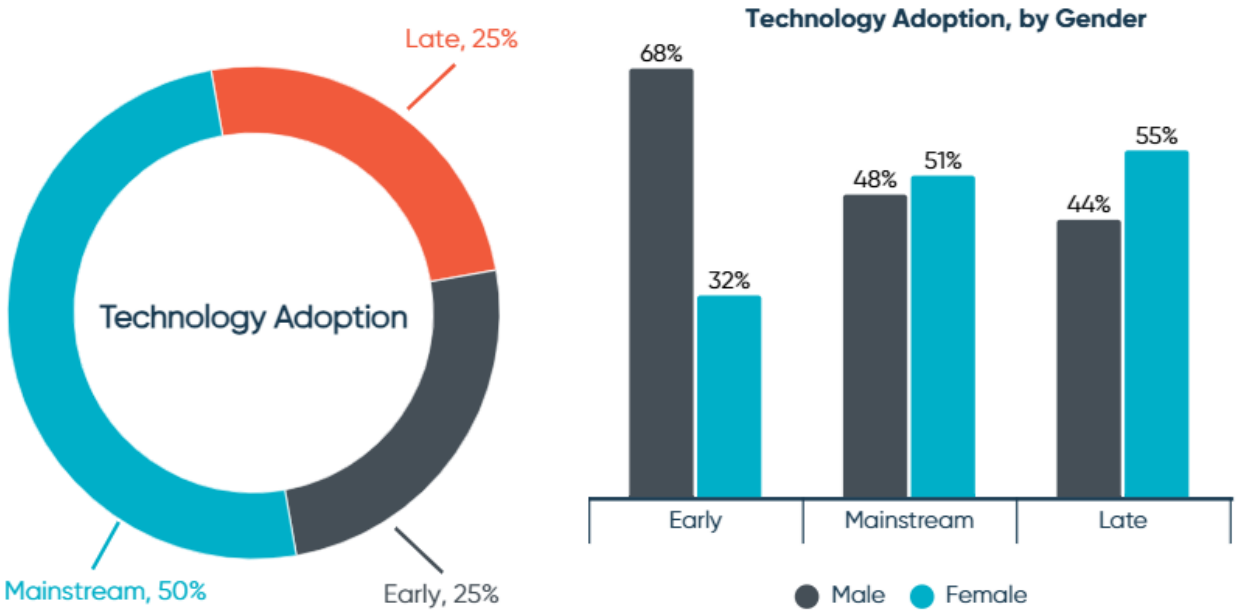

Рис. 1.4. Статистика впровадження технологій мандрівниками в подорожі

Згідно за цією діаграмою 25% мандрівників активно шукають нові технології для своїх подорожей, та швидко впроваджують їх задля планування свого відпочинку. Ще 50% починають користуватись новими технологіями, коли вони стають більш перевіреними та популярними серед інших туристів. Але також залишаються 25%, які не намагаються знайти нових технологій, а починають впроваджувати їх у свої подорожі за необхідності, тільки коли будуть впевнені, що це допоможе їх оптимізувати свої подальші дії.

Про тренди серед технологій для подорожей описано в статті “Travel Innovation and Technology Trends 2025”, за даними статті, не перестає набирати популярність штучний інтелект. Майже половина користувачів GenAI вже застосовують його для планування подорожей, що робить туристичні подорожі найкращим випадком використання та сферою зростання на наступний рік. Наступна хвиля зриву стосується автономних агентів, здатних бронювати та

керувати поїздками, оскільки такі великі компанії, як Google і OpenAI, просуваються з інноваціями на основі ШІ. Статистика обізнаності та застосування людьми різних технологій наведено на рис.1.5.

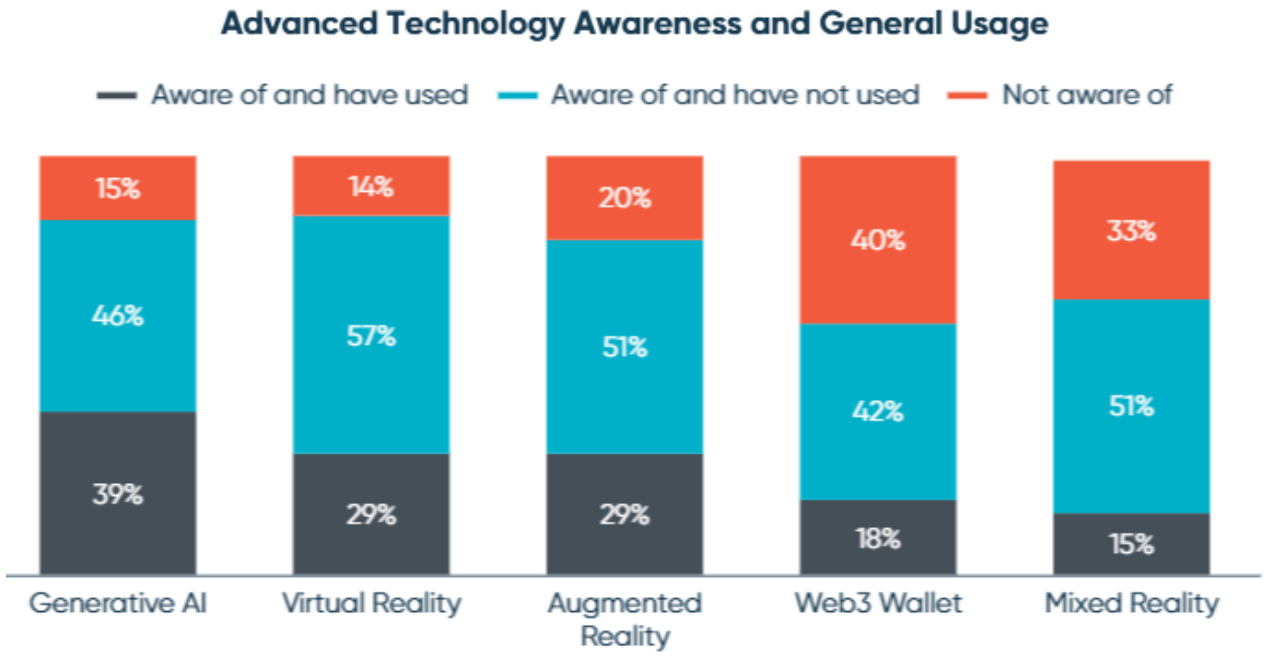

Рис. 1.5. Статистика використання технологій мандрівниками

Темно-сірим відображено відсоток людей, які активно використовують технологію. Блакитним – знають про технологію, але зазвичай не використовують її для своїх подорожей. Червоним – туристи не знають та не використовують дану технологію. [22]

Розвиток веб-систем за останні роки значно прискорився завдяки новим технологіям і підходам до розробки. Однією з головних тенденцій стала популяризація хмарних сервісів, які дозволяють зберігати та обробляти дані на віддалених серверах, забезпечуючи доступність з будь-якої точки світу. Веб-системи є доступними для будь-якого пристрої, вони масштабуються та не вимагають додаткового місця на пристрої, а також легко інтегруються з різними сервісами. Тому розробка саме веб-системи для створення єдиного сервісу для планування подорожей є оптимальною.

1.2. Аналіз існуючих додатків

1.2.1. TripIt: Travel Planner

Додаток TripIt: Travel Planner був створений у 2006 році американською компанією TripIt, як програма для планування подорожей. З моменту виходу TripIt: Travel Planner постійно розвивається та на даних момент має інтеграцію з багатьма сервісами та додатками. Ось деякі з них:

- Google Calendar, Apple Calendar, Outlook.
- Apple Watch.
- Google Maps, Apple Maps.
- Різні сервіси бронювання.

Станом на 2025 рік за рейтингами Google Play програма має середню оцінку 4.4/5 з кількістю 87 тис. відгуків [4] (рис. 1.6-1.8).

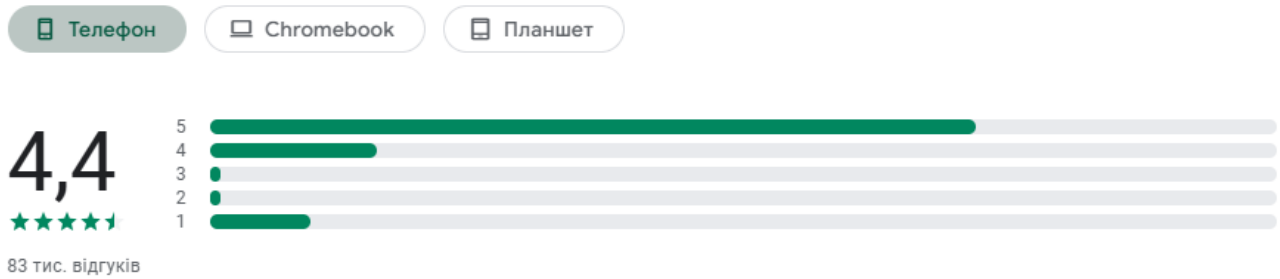

Рис. 1.6. Відгуки про TripIt: Travel Planner у Google Play (мобільна версія)

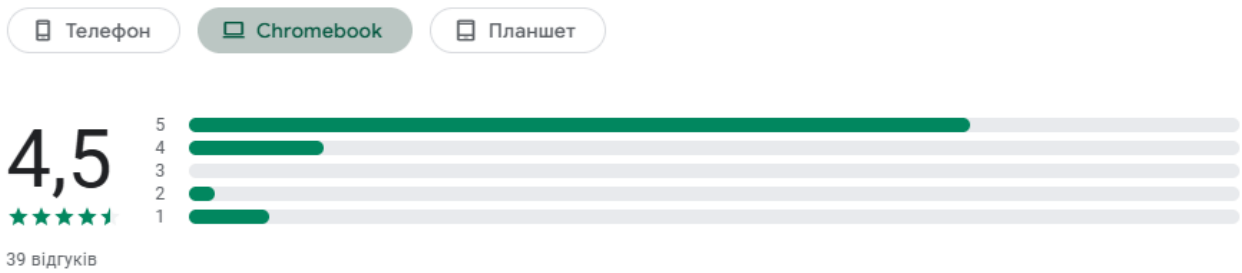

Рис. 1.7. Відгуки про TripIt: Travel Planner у Google Play (десктоп версія)

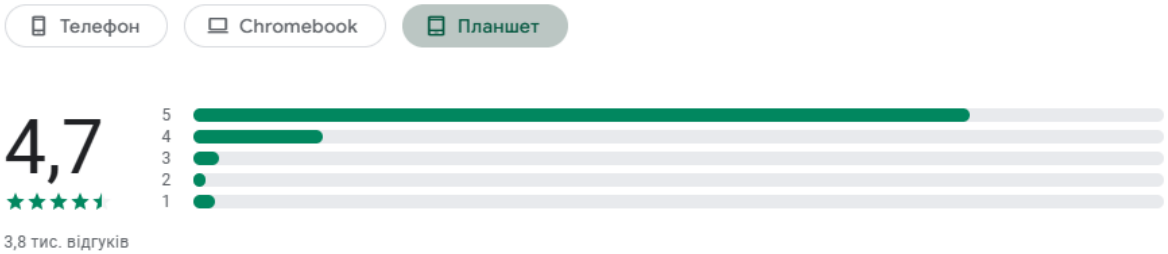

Рис. 1.8. Відгуки про TripIt: Travel Planner у Google Play (версія для планшету)

Кращу статистику демонструє App Store. Рейтинговий бал становить 4.8/5 з кількістю відгуків близько 273 тис. відгуків [5] (рис. 1.7.)

Рис. 1.7. Відгуки про TripIt: Travel Planner у App Store

TripIt: Travel Planner створює маршрути на основі заброньованих місць, а також надає можливість ділитись цими маршрутами з іншими людьми через різні програми для обміну повідомленнями. Додаток має можливості для користування офлайн, що є значним плюсом в дорозі.

Для планування маршруту потрібно надіслати електронний лист до TripIt: Travel Planner з підтвердженням бронювання квитків на транспорт, житло, інші події. Програма автоматично сформує маршрут, який можна оновлювати, редагувати, зберігати до календаря, а також багато інших корисних функцій, які забезпечать комфортну поїздку. Але в деяких джерелах та відгуках користувачів зауважено, що TripIt: Travel Planner не завжди правильно інтерпретує інформацію з електронного листа, тому потрібно ретельно перевіряти згенерований маршрут [7].

Програма має дві версії платну та безкоштовну. TripIt Pro відмінний своєю гнучкою системою сповіщень. Додаток надсилає повну інформацію про поїздку в реальному часі, а також постійні нагадування про важливі деталі. Також великим

плюсом платної версії є зберігання до 25 документів на одну подорож, у відмінності від безкоштовної – до трьох документів. [6].

TripIt: Travel Planner має зрозумій функціональний дизайн (рис. 1.8.). Інтерфейс додатку TripIt: Travel Planner). З мінусів додаток має обмеженість у мовах. Інтерфейс налічує всього англійську, японську, німецьку, французьку, іспанську мову.

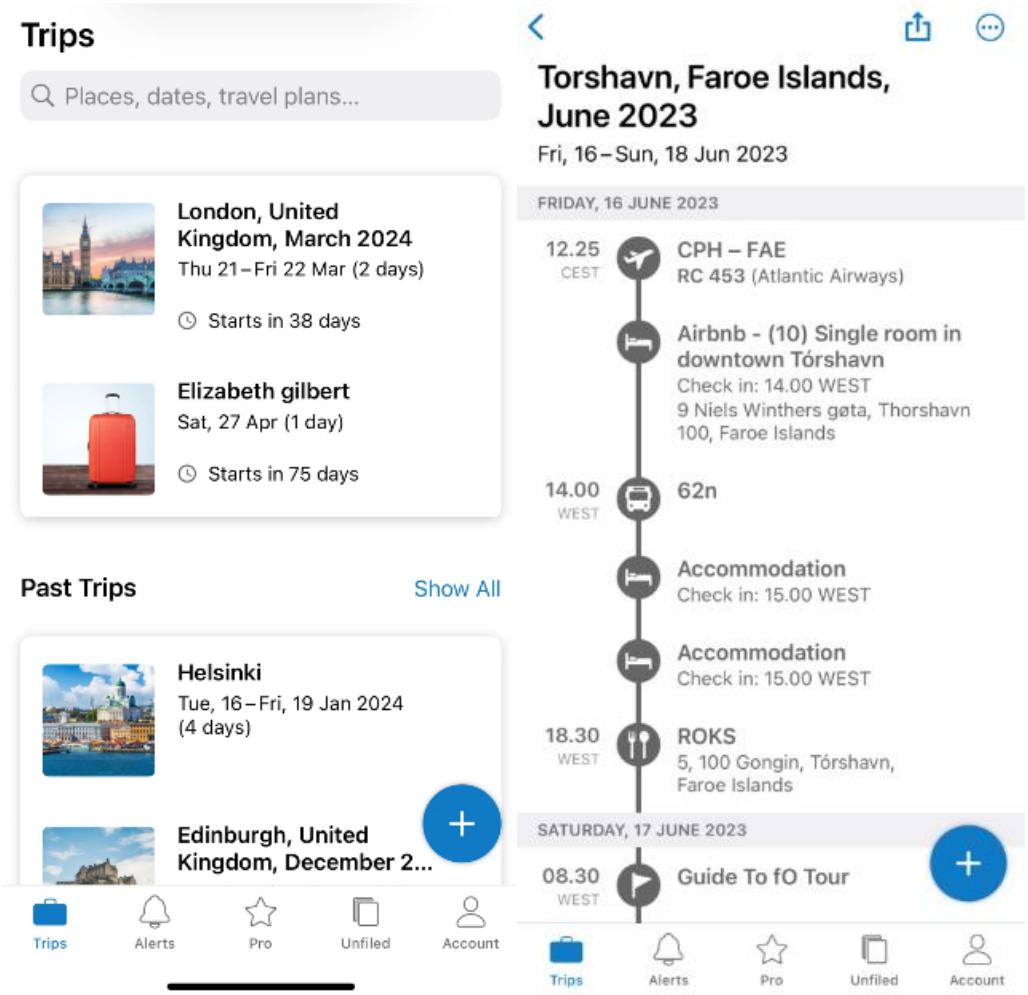

Рис. 1.8. Інтерфейс додатку TripIt: Travel Planner.

1.2.2. Roadtrippers - Trip Planner

В 2012 році компанія Roadtrippers випустила додаток для IPhone Roadtrippers - Trip Planner. В подальшому з’явились версії для різних операційних систем та пристроїв, і звісно це дуже позитивно для будь-якого додатку. Основною функцією

програми було створення та керування маршрутом за допомогою точок на карті. Спочатку додаток працював з Google Maps API, але потім перейшов на MapBox. Програма налічує більше 50000 місць, але цього недостатньо для більшості користувачів по всьому світу, що робить програму більш локалізованою. А саме найбільшою популярністю вона користувалась у США .[8]

Проєкт має гарну перспективу, але проаналізувавши відгуки можна відзначити що додаток має певні технічні проблеми. Так, наприклад, для операційної системи Android, за даними Google Play Roadtrippers - Trip Planner має оцінку 2.8/5 з кількістю більше 8 тис. відгуків від користувачів [10]. (рис. 1.9.-1.10)

Рис. 1.10. Відгуки про Roadtrippers - Trip Planner у Google Play (мобільна версія)

Рис. 1.10. Відгуки про Roadtrippers - Trip Planner у Google Play (для планшету)

Але відгуки користувачів з операційною системою IOS мають позитивну динаміку. Оцінка додатку на App Store становить 4.6/5 з кількістю 60 тис. відгуків [9]. (рис. 1.10.)

Рис. 1.10. Відгуки про Roadtrippers - Trip Planner у App Store

Але в незалежності від операційної системи користувачі відзначають проблеми зі збереженням даних, та незручність роботи у фоновому режимі. Також немає можливості користуватись застосунком офлайн, що викликає багато проблем у дорозі без інтернет-покриття.

Roadtrippers - Trip Planner має платну та безкоштовну версію і з виходом оновлень безкоштовна версія має набагато менше потрібних мандрівнику функцій, це непокоїть користувачів, так як додаток і так має багато технічних проблем.

Додаток має інтуїтивно зрозумілий зручний інтерфейс під будь-який пристрій. (рис. 1.11.)

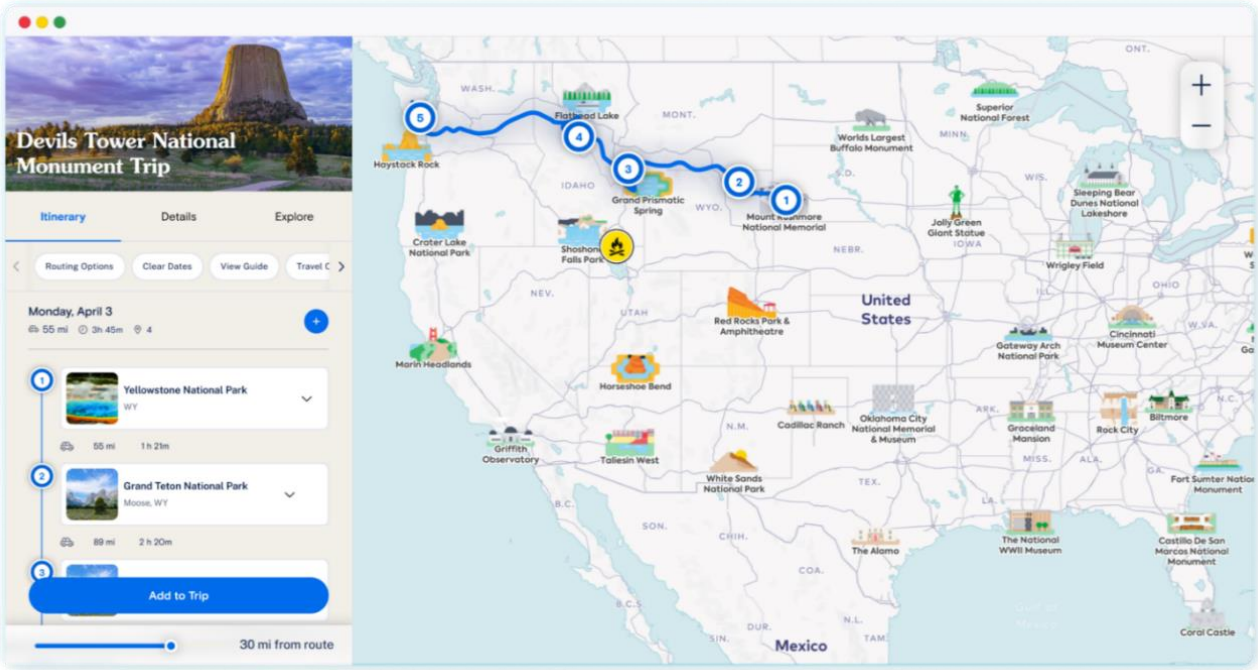

Рис. 1.11. Інтерфейс додатку Roadtrippers - Trip Planner

1.2.3. TripAdvisor

Ще одним американським продуктом є додаток Tripadvisor: Plan & Book Trips, випущений у 2008 році, але продовжує розвиватись і у 2025 році. На відміну Roadtrippers - Trip Planner, TripAdvisor, надає можливість бронювання житла, ресторанів, екскурсій, та багато іншого, необхідного мандрівникам [11].

Tripadvisor: Plan & Book Trips доступний для багатьох країн, та має 20 мов, що робить його більш популярним серед схожих додатків. Програма доступна на різних операційних системах та пристроях.

Компанія Tripadvisor за час свого існування неодноразову була затягнута у скандали на тему фейкових відгуків [12], тому я не буду спиратись на відгуки користувачів в Інтернеті про Tripadvisor: Plan & Book Trips. Також у компанії виникали проблеми з витоком даних користувачів, що говорить про недостатню захищеність даних, яка дуже негативно впливає на програмний продукт.

Інтерфейс програми чіткий та має гарний користувацький досвід. (рис. 1.12.)

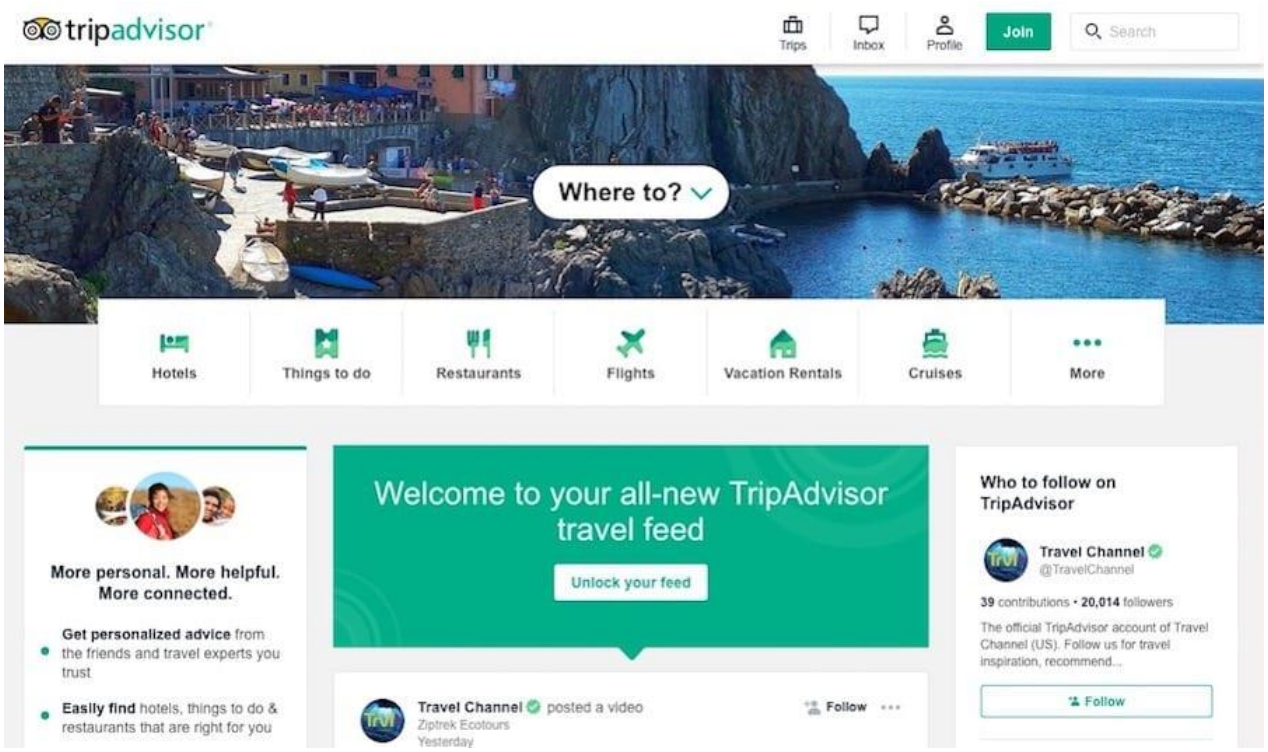

Рис. 1.12. Інтерфейс додатку Tripadvisor: Plan & Book Trips

На жаль, компанії не розкривають напряму які методи та моделі вони використовували для проєктування своїх додатків, але підсумувавши доступну в Інтернеті інформацію можна скласти порівняльну таблицю з важливих функцій та інтеграцій впроваджених в дані застосунки (табл.1.1.).

Таблиця 1.1.

Порівняння характеристик існуючих застосунків

Характеристика	Roadtrippers	TripIt	TripAdvisor
Застосування	Автомобільні подорожі	Створення планів подорожі на основі планувань	Для визначення місць та бронювання
Планування маршрутів	Побудова маршруту на основі доданих зупинок	Автоматичне створення маршруту на основі бронювань	Немає побудови маршруту
Інтерактивні карти	Використовує власну картографічну систему	Інтегрує карти Google Maps для навігації	Інтегрує карти Google Maps для перегляду місць
Допомога в плануванні	Має готові маршруту для подорожі, огляди та відгуки місць	Інтеграція з календарями, система нагадувань	Можливість зберігати обрані місця, відгуки та рейтинги місць
Офлайн-доступ	Преміум версія	Так	Ні

Продовження таблиці 1.1.

Характеристика	Roadtrippers	TripIt	TripAdvisor
Порівняння цін на готелі та ресторани	Ні	Так	Так
Онлайн бронювання	Ні	Так	Так
ОС на мобільних пристроях	IOS, Android	IOS, Android	IOS, Android
Веб-версія	Так	Так	Так

Додатки мають різний, але і в деяких факторах схожий, функціонал. Застосунок TripIt має найоптимальніші рішення для планування поїздок, так як має безпосередньо бронювання не покидаючи додаток, прокладення шляху, додавання подій у календар, а також порівняння цін, що дуже зручно для сучасних мандрівників.

1.3. Аналіз Google Maps

Історія застосунку Google Maps почалась до 2004 року, спочатку це була настільна програма, розроблена Ларсом та Єнсом Ейлструпом Расмуссенами, в австралійській компанії Where 2 Technologies. Картографічна технологія Where 2 зародилася як додаток під назвою Expedition, який був написаний мовою C++. Його було розроблено для завантаження та встановлення, як і більшість програмного забезпечення того часу. Але у 2004 році компанія Google викупила Where 2 Technologies і випустила вже веб-версію застосунку Google Maps. Карти Google базувалися на групі веб-технологій, які сьогодні широко відомі як Ajax. Це стало технологічним проривом, але це все ще був застосунок для комп'ютерів. Це змінилось у 2005 році, коли компанія влаштувала демонстрацію продукту для мобільних пристроїв, написаний мовою Java, з назвою "Google Maps for Mobile". Але вже в 2009 році додаток був випущений для Motorola, включаючи багато функціоналу для зручності користування, такого як голосовий ввід, перегляд вулиць, тощо.[14, 15].

Алгоритми, методи та технології, які використовуються в Google Maps, є передовими та високотехнологічними. Інженери Google аналізують величезні обсяги даних, включаючи історичну та реальну інформацію в режимі реального часу, що робить Google Maps настільки точним і прогресивним.

Моделі прогнозування використовують функції з безперервними значеннями, тобто передбачають нові дані. Крім того, моделі машинного навчання можуть застосовуватися разом із традиційними обчислювальними методами, що дозволяє покращувати точність існуючих прогнозів.

Google Maps використовує графові структури даних для обчислення найкоротшого шляху від початкової точки (A) до місця призначення (B). Графова структура даних складається з різних вузлів (вершин) та множини ребер, що з'єднують ці вузли. До таких алгоритмів входять алгоритм Дейкстри та алгоритм A*. [16]

1.3.1. Алгоритми пошуку шляху

Алгоритм Дейкстри – це алгоритм пошуку найкоротшого шляху у зваженому графі від одного вузла до всіх інших, але використовується лише на графах з невід’ємними вагами ребер. В реальному житті він часто використовується для вирішення маршрутизації в комп’ютерних мережах, також маршрутизації транспорту та навігації, так як основна задача алгоритму це знайти найоптимальніших варіант шляху з поміж інших за допомогою часової складності.

Для реалізації графу потрібно два масиви, логічний та чисельний. Для графу G = (V, E) кожна вершина початково не відвідана і має значення в логічному масиві false. Для обраної вихідної точки (вершини) s у чисельному масиву її дистанція записується як 0, так як алгоритм Дейкстри не розрахований на петлі. Далі розглядаються вершини які мають ребра до початкової вершини (позначимо як t і u). Обирається оптимальне ребро s – має найменшу відстань до початкової вершини, вага ребра записується як значення в чисельний масив. Обчислення відбуваються за формулою дистанція[t]=дистанція[s]+ вага ребра[s-t], аналогічно щодо іншої вершини. Після обробки вершини її значення в логічному масиві стає true, і здійснюється перехід до іншої вершини [17]. Загальний алгоритм використання:

1. Вибір початкової вершини у зваженому графі, яка слугуватиме вихідною вершиною.
2. Відстань до всіх інших вершин ініціалізується як нескінченність відносно вихідної вершини.

3. Порожня множина N містить вершини, для яких знайдено найкоротший шлях.
4. Відстань вихідної вершини відносно самої себе дорівнює нулю. Вона додається до N і називається «активною вершиною».
5. Розгляд сусідніх вершин активної вершини, які з'єднані з нею зваженим ребром. Підсумуйте відстань із вагою ребра.
6. Якщо сусідні вершини вже мають певну відстань, вона потребує оновлення, якщо нова відстань, отримана на кроці 5, є меншою за поточне значення.
7. Обрати вершину , якій присвоєно мінімальну відстань, і додати її до N. Нова вершина стає «активною вершиною».
8. Повторення кроків 5-7, доки цільова вершина не буде включена в множину N або поки в N не залишиться жодних позначених вершин.

Алгоритм Дейкстри має варіанти різної часової складності в залежності від поставленої задачі. Часова складність показує, наскільки швидко зростає час на обробку алгоритму при збільшенні кількості елементів в системі. Для зберігання простого списку відстаней використовується часова складність (O) [18]:

$$O(V^2), \tag{1.1}$$

де V – кількість вершин.

Для списку зі зберіганням предків та відстаней:

$$O(V), \tag{1.2}$$

Також існує формула для вирішення задач з бінарною купою:

$$O((V + E)\, log\, V), \tag{1.3}$$

Де E – кількість ребер.

Графічне представлення алгоритму зображено на рис. 1.13.

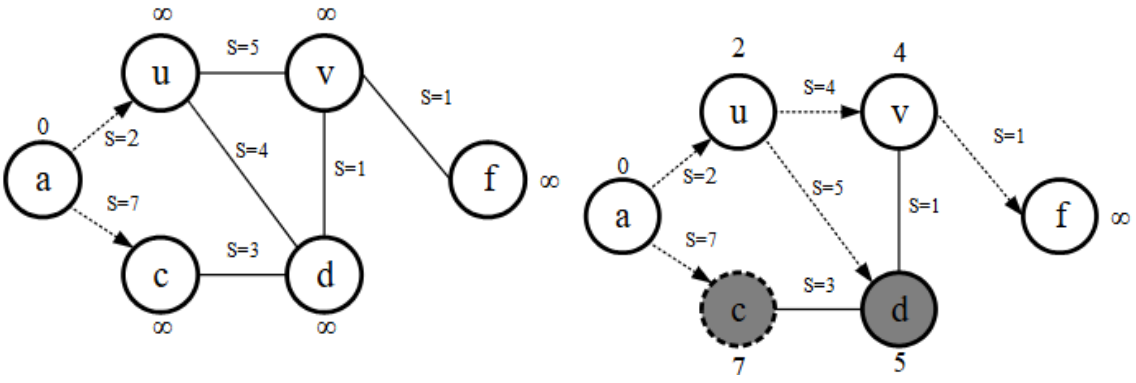

Рис. 1.13. Кроки алгоритму Дейкстри

Алгоритм A* — гнучкий, ефективніший і передовий алгоритм, який наразі використовується Google Maps для пошуку найкоротшого шляху між заданим джерелом і пунктом призначення. Він може оброблювати набагато більші об'єми інформації на відміну від алгоритму Дейкстри. Алгоритм A* поєднує в собі інформацію, що надає алгоритм Дейкстри, та інформацію алгоритму Greedy Best-First Search (жадібний пошук). Ця комбінація робить його чітким та швидким з пошуку рішень поставленого завдання.

Одна з відмінностей між алгоритмом A* та алгоритмом Дейкстри полягає в тому, що алгоритм A*, будучи евристичним алгоритмом, використовує дві функції вартості:

- g(x) – фактична вартість досягнення вершини
- h(x) – приблизна вартість досягнення цільового вузла від поточного вузла n
- f(x) – загальна оцінена вартість кожного вузла.

Вартість, розрахована алгоритмом A*, не повинна бути переоціненою, оскільки це евристичний алгоритм. Отже, фактична вартість досягнення цільового вузла з будь-якого даного вузла n повинна бути більшою або рівною h(x). Це називається допустимою евристикою.

Таким чином, загальну оцінену вартість кожного вузла можна представити як:

$$f(x) = g(x) + h(x) \tag{1.4}$$

Донецький національний університет імені Василя Стуса провів експеримент порівняння цих двох алгоритмів за допомогою реалізації на мові Python. За вхідні дані була взята початкова точка (0,0) та кінцева (499, 499) з 300 перешкодами. Результатом експерименту стало, що алгоритм A* є швидшим за алгоритм Дейкстри в більше ніж 8 разів. Але також зазначено, що для такого результату потрібна чітка зазначена евристична інформація. Кожен алгоритм ефективно використовувати в залежності від задачі, так як алгоритм Дейкстри перебирає кожен варіант шляху, що може затримати час його виконання, а алгоритм A* може оцінювати конкретну ділянку, відкидаючи вже завчасно прогнозовані не оптимальні варіанти шляху [23].

1.3.2. API Google Maps

API Google Maps створений для розробників задля легкої інтеграції в свої проєкти. Це дозволяє відтворювати карти, додавати до них різні об’єкти, такі як позначки, написи, налаштування стилей відображення. Це працює за допомогою мови програмування JavaScript, подання HTTP запросу на сервер зі створеним ключем на сервісі Google.

API Google Maps має широкий спектр застосування, це також можуть бути окремі знімки у проєктах, надання географічних координат, обчислювання відстаней від точки А до точки Б, інформацію про різні місця на карті, робота з даними доріг, панорамні знімки та багато іншого, що можна використати в своїх проєктах компаніям, або окремим розробникам.

1.4. Визначення цілей дослідження

Розробити веб-систему для планування подорожей, яка інтегрується з Google картами, забезпечуючи користувачів зручними інструментами для створення персоналізованих маршрутів. Основна мета роботи наведена в дереві цілей (рис. 1.14)

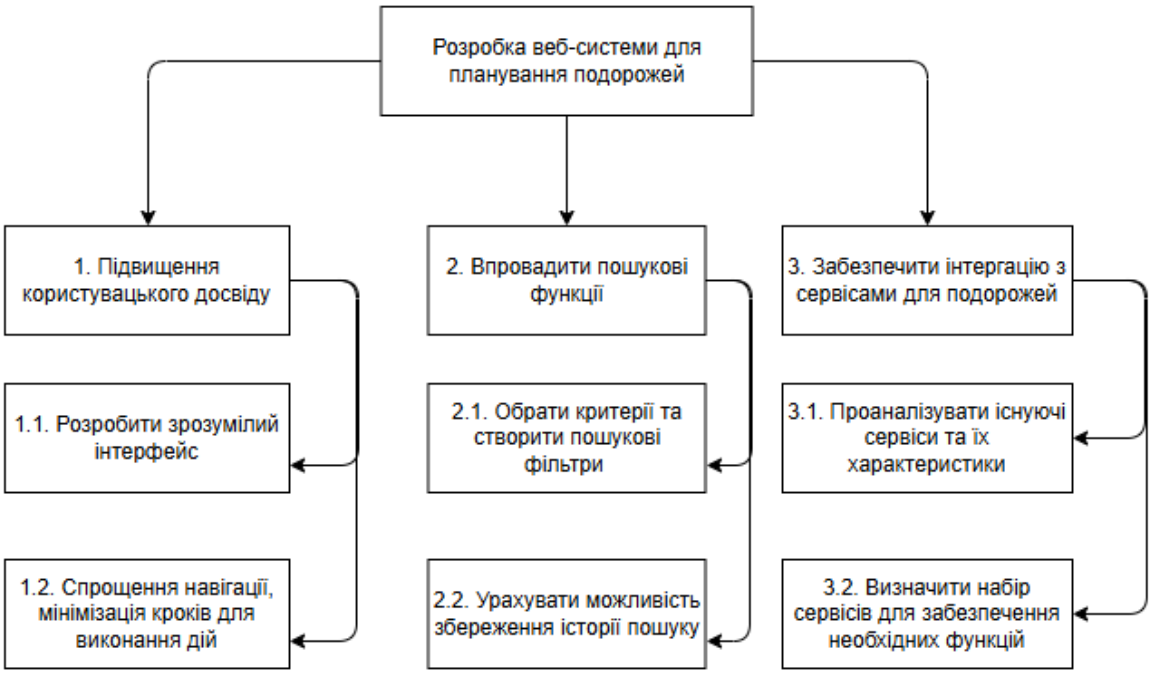

Рис. 1.14. Дерево основних цілей розробки веб-системи для планування подорожей

Досягнення поставлених цілей здійснюється реалізацією наступний поставлених завдань:

- Оцінити поточні потреби користувачів у системах для планування подорожей і визначити основні функції.
- Розробити архітектуру веб-системи, яка інтегрується з Google картами та враховує особливості маршрутного планування.
- Визначити та реалізувати ключові функціональні можливості, зокрема для збереження та редагування маршрутів.
- Провести тестування для перевірки функціональності, зручності та безпеки системи.
- Створити та підключити базу даних для збереження маршрутів і даних користувачів

Дерево основних задач наведено на рис. 1.15

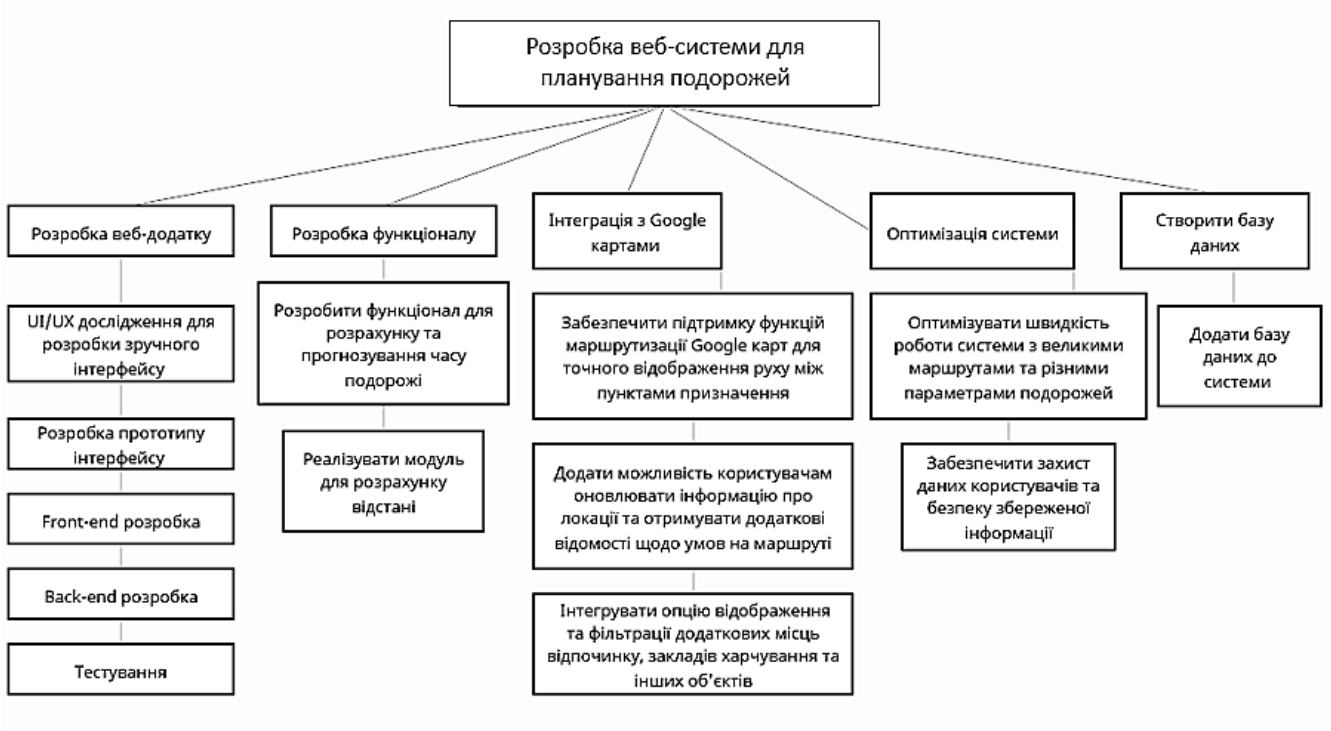

Рис. 1.15. Дерево основних задач розробки веб-системи для планування подорожей

1.5. Аналіз веб-систем

Веб-системи – це збірний термін, у веб-систему входять як веб-додатки так і веб-сайти. У них є суттєві відмінності в характеристиках та функціях. Кожен з них використовується з різними цілями. Цілями використання можуть бути: демонстрація певного продукту або ж його продаж, особистий блог, надання та продаж послуг різного характеру, комунікація в компанії між робітниками або клієнтами, навчальні платформи та інше. Для кращого розуміння пропонується розглянути кожен з цих понять.

Веб-додатки мають основне призначення взаємодію з користувачем, вони є інтерактивним ресурсом та мають різні види. За функціональністю вони включають в себе обробку, редагування, обмін та збереження даних.

Види веб-застосунків наведено в табл.1.3.

Таблиця 1.3.

Види веб-додатків

Вид	Пояснення	Переваги	Недоліки
SPA	Односторінковий застосунок, має автоматичне оновлення	Має швидку обробку дій, зазвичай не потребує звернення до серверу	Потрібний стабільний доступ до мережі Інтернет. В деяких випадках занадто велике навантаження для браузеру
MPA	Багатосторінковий застосунок, має постійну взаємодію з сервером	Оптимізація сторінок, робота з великими об’ємами даних	Дорожчі та складніші в обслуговуванні
PWA	Гібрид веб-сторінки та веб додатка	Може працювати на різних ОС, швидка розробка за допомогою окремого модуля на веб-сайти	Деякі браузери не підтримують даний формат

Але також кожен з цих видів поділяється на види за типом застосування (табл. 1.4), а також за призначенням (табл.1.5.) [24,25]

Таблиця 1.4.

Класифікація веб-додатків за типом взаємодії з користувачем

Вид	Призначення
Статичні	Інформаційні додатки, не мають взаємодії з користувачем
Динамічні	Містять функціонал для взаємодії, наприклад, пошукову систему
Інтерактивні	Мають тісну взаємодію, прикладом є навчальні додатки

Таблиця 1.5.

Класифікація веб-систем за призначенням

Система	Призначення
E-commerce	Використовується для Інтернет-магазину
CRM	Створені для поліпшення співпраці з клієнтами, робітниками. Також є функція для розробки бази даних
ERP	Реалізовують можливість управління підприємством

На відміну від веб-додатків, веб-сайти, призначені для подання інформації користувачеві, без суттєвої взаємодії з ним. Це статичні веб-сторінки, але вони також можуть бути динамічними за допомогою елементів інтерактиву та анімацій. Зазвичай у них немає тісної взаємодії з сервером, так як немає обробки великих обсягів інформації. А також для створення веб-сайтів достатньо тільки базових знань технологій веб-програмування.

Постановка задачі

В результаті проведено аналізу виявлено проблему низької кількості веб-систем, які б забезпечували повний необхідний функціонал для планування подорожі. Основна задача – розробити веб-додаток виду MPA, в якому будуть враховані потреби користувачів. Для цього необхідно вирішити наступні підзадачі:

- Збір вимог користувачів.
- Розробка архітектури системи.
- Інтеграція необхідних сервісів
- Моделювання бази даних.
- Створення інтерфейсу користувача.

Вхідні дані:

- Інформація про користувачів при реєстрації в системі

- Дані для побудови маршрутів. Дані про місця призначення, отримані координати.
- Інформація від інтегрованих сервісів.

Вихідні дані:

- Маршрути подорожі
- Історія поїздок
- Записані дані в інтегрованих сервісах

Висновки до розділу 1

У цьому розділі було здійснено детальний аналіз додатків, які пропонуються для планування подорожі, а саме Roadtrippers, TripIt, TripAdvisor. Виявлено слабкі та сильні сторони цих застосунків. Найкращим варіантом для планування подорожей обрано TripIt, він забезпечує весь необхідний функціонал: побудову маршруту, додавання подій в календар, бронювання готелів та інших місць через додаток, але також зазначено що інколи, через технічні помилки, додаток не правильно прокладає маршрут. Тому TripIt не є системою, яка повністю задовільняє потреби користувачів.

Також було визначено поставлені цілі та задачі, які необхідно виконати для розробки функціональної веб-системи.

Поставлена задача має на меті створення високофункціональної веб-системи, яка буде інтегрована з картографічними та календарними сервісами для забезпечення зручності користувачів при плануванні подорожей.

2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ВЕБ-СИСТЕМИ

2.1. Вибір методології розробки

Для коректної розробки програмного забезпечення потрібно дотримуватись певних етапів розробки, щоб упорядкувати ці етапи в певний алгоритм існують методології розробки. Найпопулярніші з них є Agile (також має декілька фреймворків), та Waterfall. Методології потрібні для чіткої організації робочих процесів під час розробки та допомагають уникати незапланованих складнощів. Agile та Waterfall мають протилежні особливості.

Agile є гнучкою, дозволяє швидко адаптуватись до змін та враховувати нові вимоги, навіть всередині циклу розробки. Її особливість в тому, що вона ітеративна, дозволяє виявляти недоліки системи на кожній ітерації, що є дуже важливим в проєктах. Кожна ітерація представляє працюючу версію продукту, яка з кожним новим «колом» розробки вдосконалюється. Але також методологія містить і великий мінус, так як у неї відсутній чіткий план розвитку проєкту, через можливість змінності вимог.

Рис. 2.1. Модель Agile

Agile також має декілька фреймворків, такі як Scrum та Kanban. Хоч вони і схожі між собою, так як на базі однієї методології, але виконують різні функції. Scrum дозволяє поділяти розробку на спринти, зазвичай один спринт розраховано на 2-4 тижні, передбачає регулярні зустрічі команди, фіксовані ролі, та контроль над процесами. Kanban в свою чергу не має ітерацій розробки, які обмежені в часі, також немає чіткого розподілу ролей в команді, але натомість містить більше статусів задачі, що надає змогу більш точно відслідковувати виконання роботи.

Головна відмінність Agile від Scrum, те що Agile це набір принципів та цінностей, які наголошують на гнучкості та співпраці в команді, в той час як Scrum має прописану структуру з визначеними ролями в команді [26].

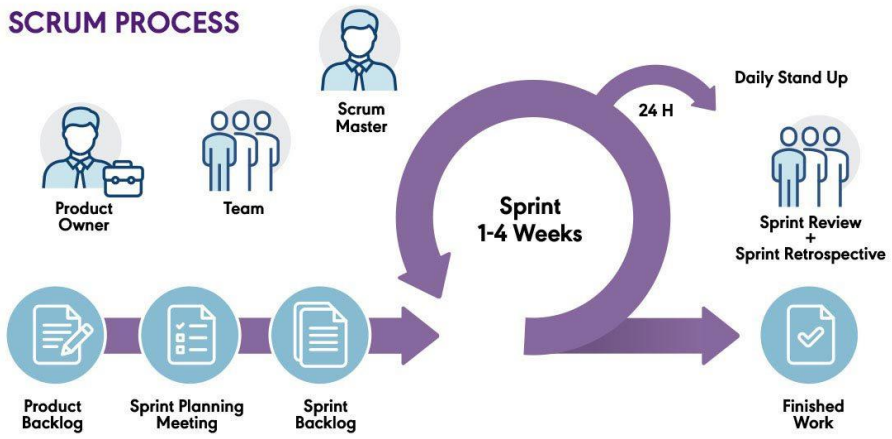

Рис. 2.2. Модель Scrum

Модель Waterfall є стабільною, вона має жорсткі вимоги, які не можна змінювати в ході розробки, також чітке розподілення на етапи, які не перекриваються та повинні бути завершені перед переходом на інший етап. Оригінальна модель «водоспаду» містила в собі 7 етапів: системні вимоги, вимоги до програмного забезпечення, аналіз, дизайн, кодування, тестування, операції. Але звичайно з того часу методологія стала більш гнучкою в плані того, що кожен проєкт має право сам обирати етапи розробки. Чіткою особливість є те, що реалізація цієї моделі вимагає наявність великого обсягу документації після кожної фази.

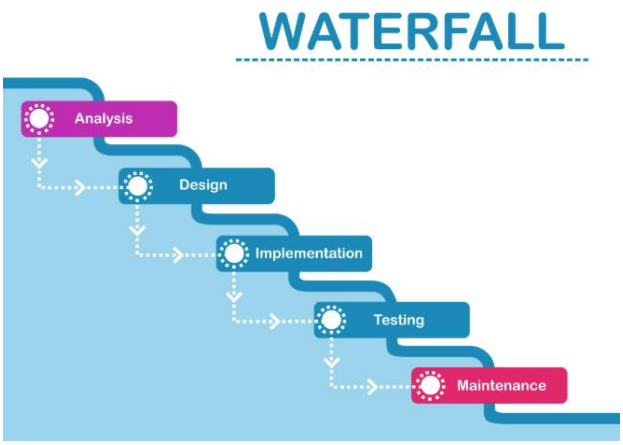

Рис. 2.3. Модель Waterfall

Характеристики розглянутих методологій представлені в табл.2.1.

Таблиця 2.1.

Порівняльна таблиця методологій розробки

Характеристика	Agile	Waterfall	Scrum	Kanban
Тип	Гнучка	Лінійна	Фреймворк Agile	Фреймворк Agile
Структура робочих процесів	Спринти (цикли)	Послідовні етапи	Спринти	Потік задач
Часові рамки	Виконання спринтів, потім внесення корективів	Поетапна розробка з фіксованими термінами	Фіксовані терміни для спринта	Немає обмеження в часі
Ролі в команді	Не визначено	Залежить від організації	Чіткі ролі в проєкті	Не визначені
Документація	Мінімальна	На кожному етапі роботи	Мінімальна, в межах спринта	Мінімальна, орієнтовано на потік
Особливість	Адаптація до змін	Уникнення змін	Робота по спринтам	Потік задач
Інструменти для управління	Jira, Trello, Asana, бази даних	Без сторонніх інструментів	Jira, Trello, Asana	Trello, Kanban Board, Jira
Для проєктів з	Швидкими змінами	Чіткими вимогами та термінами	Командами, які працюють поетапно	Великими обсягами завдань

Для розробки веб-системи планування подорожей, на основі табл.2.1., була обрана методологія Agile з фреймворком Scrum, так як вона дозволяє поділяти розробку на спринти та має високу гнучкість. Адаптація до змін в системі такого плану надзвичайно важлива, так як завчасно передбачити всі вимоги користувачів щодо подорожей неможливо. В ході розробки можуть з’явитись нові вимоги, які будуть необхідні для комфортного планування майбутніх пригод. Agile безпосередньо дозволяє контролювати якість виробленого продукту, за допомогою тестування на кожному етапі розробки.

Враховуючи вище сказане потрібно скласти план розробки проєкту за обраною методологією. Scrum передбачає створення продукту за допомогою ітераційних спринтів з розділенням завдань по цим спринтам (табл. 2.2.)

Таблиця 2.2.

Етапи розробки спринту

Етап	Задача	Пояснення
Планування спринту	Оцінка завдань	Оцінка завдань, відповідно списку вимог
		Визначення завдань для поточного спринта
		Оцінка складності
	Розподіл завдань	Розподіл завдань між членами команди відповідно до їх кваліфікації
Виконання	Розробка функцій	Реалізація визначених функцій за пріоритетністю
	Фіксація прогресу	Визначення що реалізовано повністю, часткова, потребує реалізації
	Тестування	Тестування виконаних функцій
Демонстрація результату	Демонстрація виконаної роботи	Демонстрація реалізованих функцій
	Отримання зворотнього зв’язку	Рекомендації від замовників щодо виконаної роботи, якщо такі є
Покращення	Аналіз процесу	Аналіз спринта для наступних покращень
Підготовка до наступного спринта	Оновлення списку вимог	На основі отриманого зворотнього зв’язку внесення нових (або оновлених) вимог

2.2. Збір та аналіз вимог

Незважаючи на те, що за обраною методологією Agile(Scrum) вимоги до проєкту можуть змінюватись в процесі розробки, необхідно чітко визначити «вхідні» вимоги. Це потрібно для подальшого планування спринтів, часу на виконання задач, оцінки їх важливості та пріоритетності. Вимоги є ключовими для формування функціоналу, інтерфейсу, розробки архітектури. Вони бувають різних типів та повинні дотримуватись набору характеристик

2.2.1. Характеристики вимог

Характеристики є як для окремої вимоги, розглянуті в табл. 2.3., так і для всього набору специфікації, детальніше в табл. 2.4. [27]. Вони є важливими для того, щоб забезпечити якість та ефективність розробки, оптимізувати витрати часу та ресурсів, уникнути конфліктів між окремими частинами системи та інші аспекти. Вимоги складені не за характеристиками будуть хаотичні та викликати технічні складнощі, і не тільки.

Таблиця 2.3.

Характеристики до вимог

Характеристика	Опис
Повнота та коректність	Вимога повинна бути довершеною, повністю описувати функціонал, який вона передбачає. Якщо вимога має «пробіли», тобто певну недостачу інформації для повного опису, то це має бути зазначено.
Однозначність	Вимога повинна бути тільки з одним прозорим сенсом
Реалізованість та можливість протестувати	Її можна реалізувати та протестувати декількома тестами
Необхідність	Кожна вимога повинна бути потрібною
Пріоритетність	Вимоги оцінюються за важливістю, щоб уникнути непередбачуваних ситуації та дотримуватись чіткого розрахунку ресурсів

Таблиця 2.4.

Характеристики до набору специфікацій вимог

Характеристика	Опис
Повнота	Жодна з вимог не повинна бути пропущеною
Не конфліктність	В наборі не повинно бути конфлікту між вимогами, в незалежності від їх рівня
Здатність до модифікації	Адаптація вимог під певні умови в ході розробки
Можливість аналізу	Забезпечити можливість відслідкувати звідки і для чого певна вимога в специфікації

2.2.2. Типи вимог

Загалом, вимоги до розробки ПЗ поділяються на «функціональні» та «нефункціональні», «бізнес-вимоги» та «вимоги користувачів». А також вони мають певні характеристики, які визначають наскільки добре прописана вимога.

Бізнес-вимоги містять мету, цілі та побажання замовників. Передбачають розмір бюджету та очікування від проєкту. Це нетехнічні вимоги, в даній категорії не обговорюють як працює продукт, а акцентують увагу на те, що повинен він робити.

Вимоги користувачів містять потреби та очікування користувачів від продукту. В цій категорії проводиться аналіз цільової аудиторії задля ефективної взаємодії з майбутніми користувачами. Включає необхідний функціонал, а також характеристики системи для вдалого користувацького досвіду.

Функціональні вимоги описують функціональну частину продукту, а також поведінку. Включають конкретні функції, описують дії та операції, наприклад, збереження даних, які використовувались користувачем. Ці вимоги відрізняються від «нефункціональних» тим, що не описують внутрішні аспекти, наприклад, продуктивність, а містять лише опис поведінки системи, що розробляється.

Нефукціональні вимоги пояснюють внутрішню роботу, але це не впливає на функціонал. Вони містять недоліки або обмеження продукту, а також вимірювальні характеристики, такі як кількість одночасних користувачів, час обробки запитів. Ці вимоги дуже важливі, бо включають аспекти надійності, зручності користування, тощо.

Всі ці типи пов’язані між собою (рис.2.4.) і не можна виключити якусь з категорій, так як тоді багато аспектів буде упущено, що веде до неправильного планування та можливих наслідків, які негативно вплинуть на розробку продукту в цілому.

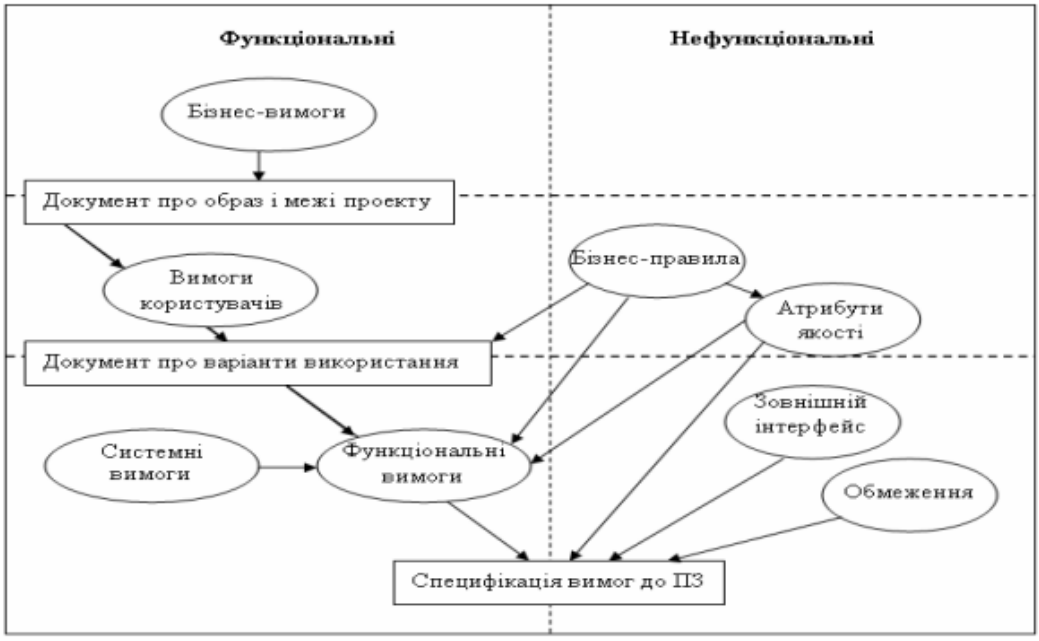

Рис. 2.4. Взаємодія вимог

2.2.3. Вимоги до веб-системи планування подорожей

Враховуючи правила побудови бази вимог та необхідні характеристики, зібрані вимоги до веб-системи планування подорожей відображено у табл. 2.5. Таблиця побудована за шаблоном повного опису варіантів використання по А. Коберну [27]. Цей шаблон надає можливість повністю описати вимогу з дотриманням правил, варіантами різних ситуацій використання, що потрібне для розробки системи.

Таблиця 2.5.

Опис вимог до системи

Назва	Реєстрація	Вхід	Створити подорож	Додати точки	Зберегти подорож
Викор.	Реєстрація в системі	Вхід в систему	Створення подорожі з всіма точками та іншими даними	Додавання точок у маршрут при створені нової подорожі	Збереження даних про поїздку в систему
Зона	Підсистема авторизації	Підсистема авторизації	Підсистема планування подорожі	Підсистема маршрутів	Підсистема бази даних
Рівень	Цілі користувача	Цілі користувача	Цілі користувача	Цілі користувача	Узагал.
Дійова особа	Користувач	Користувач	Користувач	Користувач	Користувач
Передумова	Користувач має акаунт гугл або пошту	Користувач має акаунт гугл	Користувач створює маршрут	Користувач додає дані про зупинки	Всі необхідні дані про поїздку заповнені
Гарантії успіху	Користувач успішно доданий до системи	Успішний вхід в систему	Побудова маршруту	Збереження точок, побудова маршруту	Маршрут успішно збережено до історії поїздок
Тригер	Користувач натискає кнопку «зареєструватись»	Користувач натискає кнопку «увійти»	Користувач натискає «створити подорож»	Користувач додає мітку на карті, або прописує назву місця у відповідне поле	Користувач натискає «зберегти» в бланку даних про поїздку
Основний Сценарій	Заповнює форму реєстрації або вхід за допомогою гугл акаунту	Заповнює дані для входу або вхід по гугл акаунту	Користувач вказує всі необхідні дані. Додавання даних в систему	Додавання точок, запис назв у відповідні поля, мітки на карті	Збереження поїздки без втрат даних

Продовження таблиці 2.5.

Назва	Інтеграція з Google Maps	Інтеграція з Google Calendar	Зберігати події в Google Calendar	Перегляд історії поїздок
Викор.	Інтеграція з гугл картами для відображення карт, побудови маршрутів	Інтеграція з гугл календарем для додавання подій поїздки в календар	Збереження та перегляд подій в календарі	Користувачі повинні мати змогу переглядати створені поїздки
Зона	Підсистема інтеграцій	Підсистема інтеграцій	Підсистема бази даних	Підсистема бази даних
Рівень	Узагал.	Узагал.	Узагал.	Цілі користувача
Дійова особа	Система	Система	Система	Користувач
Передумова	Google Maps API	Google Calendar API	Успішна інтеграція з Google Calendar	Створена поїздка
Гарантії успіху	Система надає маршрути та карти	Система додає події на обрані дати в календар	Події додано в Google Calendar	Історія поїздок доступна для користувача
Тригер	Взаємодія з картою при створені поїздки	Взаємодія з календарем	Додавання дати та точки	Користувач натискає кнопку «історія поїздок»
Основний сценарій	Система успішно надає маршрути по заданим точкам, відображає карту, працює пошук	Система підключена до календаря	Користувач додає мітку для поїздки, дати. Створення події в календарі	Користувач переглядає створені поїздки з усіма вхідними даними

В таблиці 2.5 . здебільшого наведені функціональні вимоги, але система має також бізнес-вимоги, вимоги користувачів та нефункціональні вимоги.

Інші типи вимог наведені в табл. 2.6.

Таблиця 2.6.

Опис вимог до системи

Вимога	Опис
Продуктивність та швидкість	Час завантаження сторінки та карти не повинен перевищувати 2-3с.
	Забезпечити здатність обробки великого обсягу даних
Масштабованість	Забезпечити вміння масштабуватись для обробки великих обсягів даних та кількості користувачів
Безпека	Забезпечення захисту персональних даних
	Реєстрація та вхід через запити до гугл акаунту
Зручність використання	Розроби інтуїтивно зрозумілий інтерфейс
	Забезпечити адаптивність дизайну під різні екрани
Сумісність	Система повинна працювати на різних пристроях, браузерах, ОС
Надійність	Цілодобова доступність для користувачів
	Резервне копіювання даних
Багатомовність	Підтримка різних мов для інтерфейсу
Функціональне планування	Забезпечити легке створення та редагування маршруту
Відповідність юридичним вимогам	Система повинна відповідати всім вимогам щодо безпеки та захисту користувачів
	Дотримання всіх законодавчих норм в галузі туризму

2.3. Системне проєктування

Внаслідок урахування всіх недоліків існуючих методів проєктування інформаційних систем з’явилось поняття системного підходу до проєктування. Цей підхід використовує поняття системи, що складається з компонентів, які пов’язані загальною метою.

Системне проєктування має певний набір принципів:

- Кінцева мета – в системі все повинно бути впорядкованим до глобальної мети

- Єдність – розгляд системи, як об’єкт надсистеми, або як саму надсистему, що складається з підсистем, які знаходяться в певних відносинах
- Ієрархічність – зв’язки між елементами системи повинні бути впорядковані по рівням
- Зв’язність – кожен елемент системи розглядається разом з його зв’язками з іншими елементами
- Модульна побудова – виділення окремих модулів з яких складається система
- Функціональність – спочатку необхідно з’ясувати функції системи
- Розвиток – можливість вдосконалення
- Децентралізація – вибір декількох головних елементів
- Невизначеність – можливі невідомі моменти в системі

Розглядаючи веб-систему планування подорожей потрібно розуміти що вона входить до надсистеми онлайн-сервісів для подорожей і картографічних сервісів. Вона взаємодіє з іншими елементами цієї системи, а саме: веб-серверами, користувачами, сервісами, які будуть інтегровані, для отримання необхідних функцій. Взаємодія елементів показана на рис. 2.5.

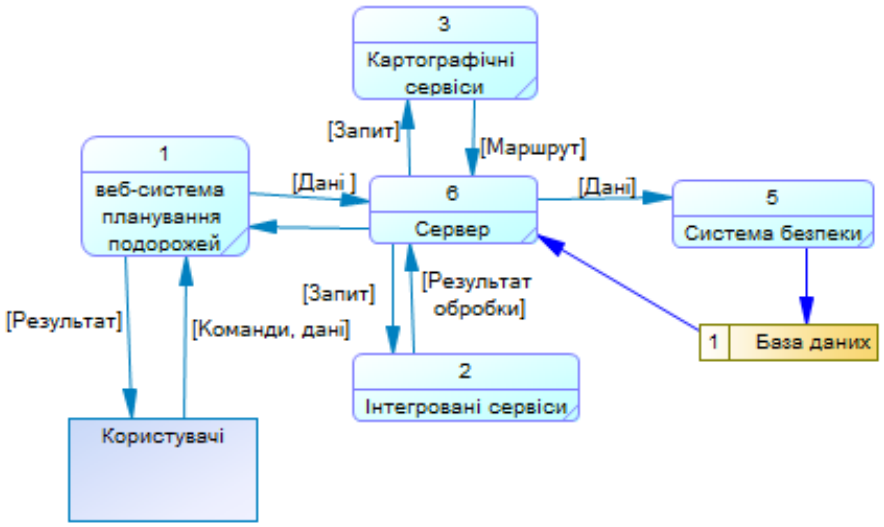

Рисунок 2.5. Взаємодія веб-системи планування подорожей з іншими елементами надсистеми

Метою аналізу є зрозуміти чітку структуру системи, що розглядається, тому потрібно розглядати систему як надсистему, незалежну одиницю. Веб-система планування подорожей складається з підсистем, а ті в свою чергу з необхідних модулів, які забезпечують необхідний функціонал, для коректної роботи. Схему розбиття системи на підсистеми та модулі представлено на рисунку 2.6.

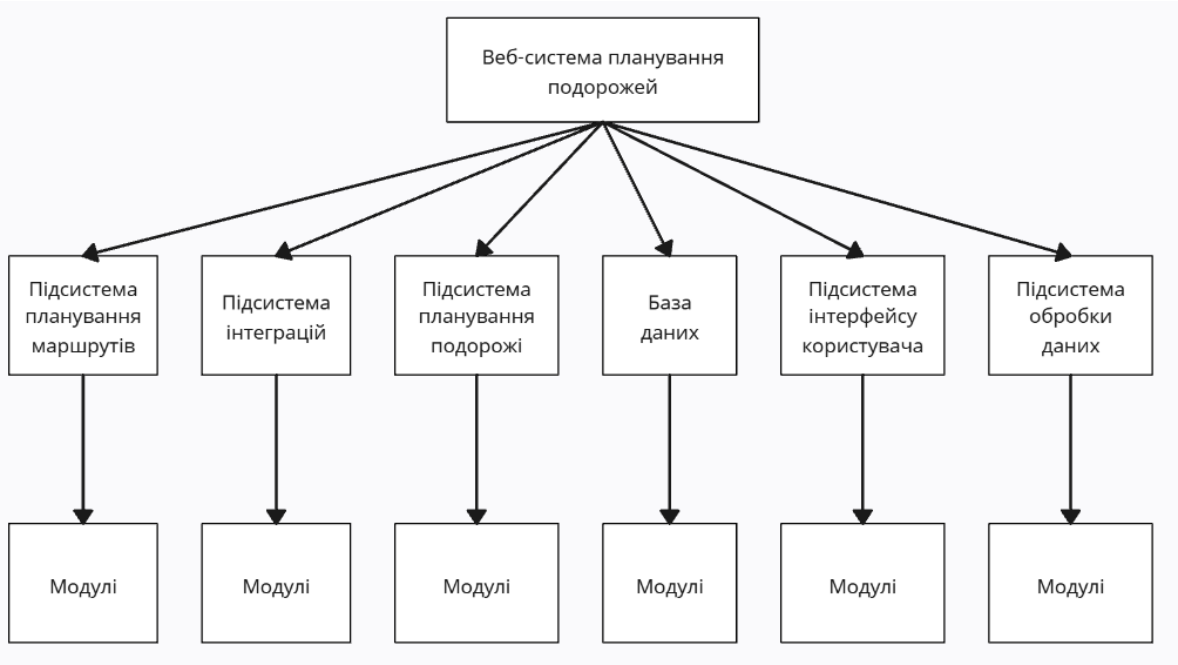

Рисунок 2.6. Схема декомпозиції системи

Як вище зазначено, кожна підсистема складається з певних модулів, для кращого розуміння потрібно окремо розглянути кожну з них окремо.

Підсистема планування маршрутів містить модулі (рис. 2.7.): алгоритми для розрахунку маршруту, налаштування та модифікації маршруту. Основні зв’язки для реалізації даного функціоналу має з підсистемою інтерфейсу та підсистемою інтеграцій.

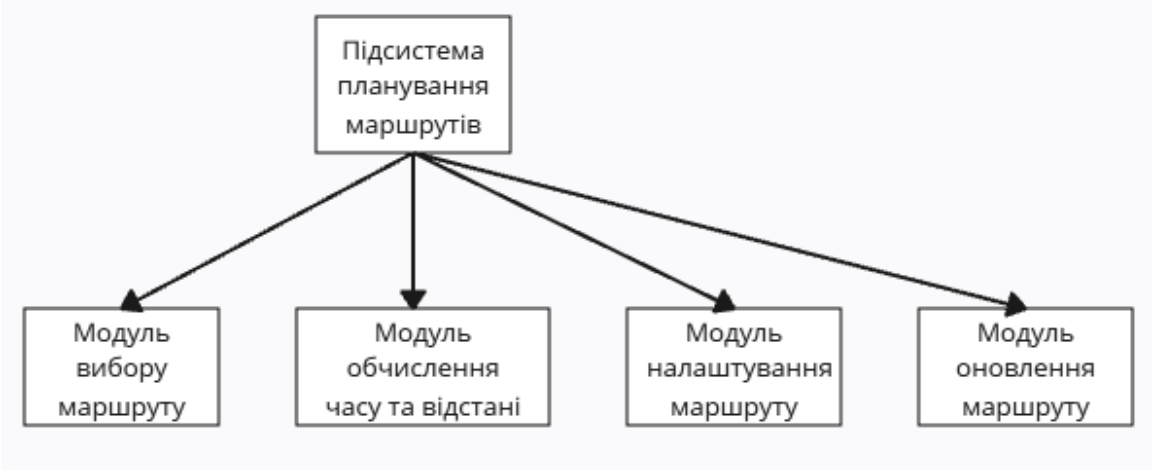

Рис. 2.7. Модулі підсистема планування маршрутів

Підсистема інтеграцій (рис.2.8.) включає в себе інтеграцію з Google Maps API та Google Calendar. Основною функцією є забезпечення коректної роботи підсистем планування маршруту та планування поїздки, так як вони виконують свої основні задачі за допомогою функціоналу який надають інтегровані сервіси.

Рис. 2.8. Модулі підсистеми інтеграцій

Підсистема планування подорожі відповідає за створення, редагування та збереження подорожей, а також за перегляд історії поїздок. Схема необхідних модулів зазначена на рис. 2.9.

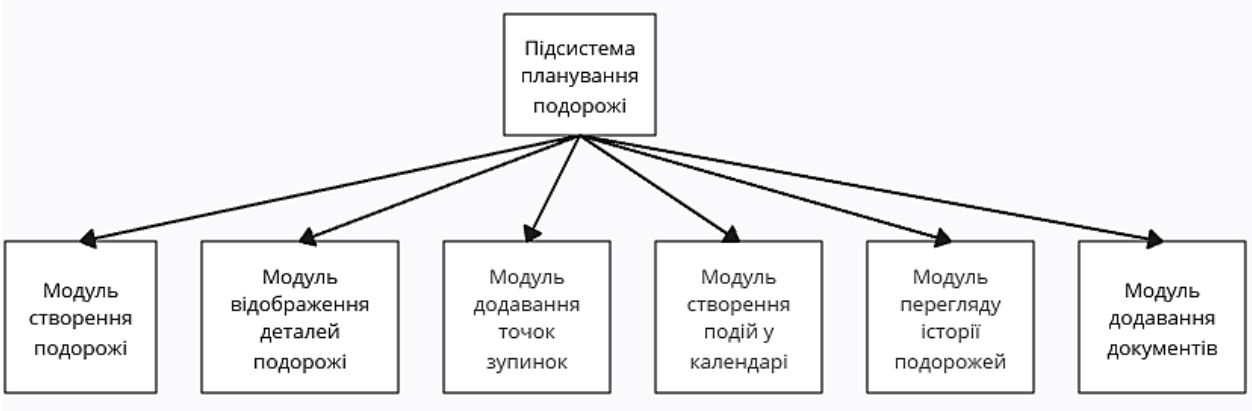

Рис.2.9. Модулі підсистеми планування подорожі

База даних потрібна для збереження всіх введених та обчислених даних. Схема необхідних модулів бази даних зазначена на рис. 2.10.

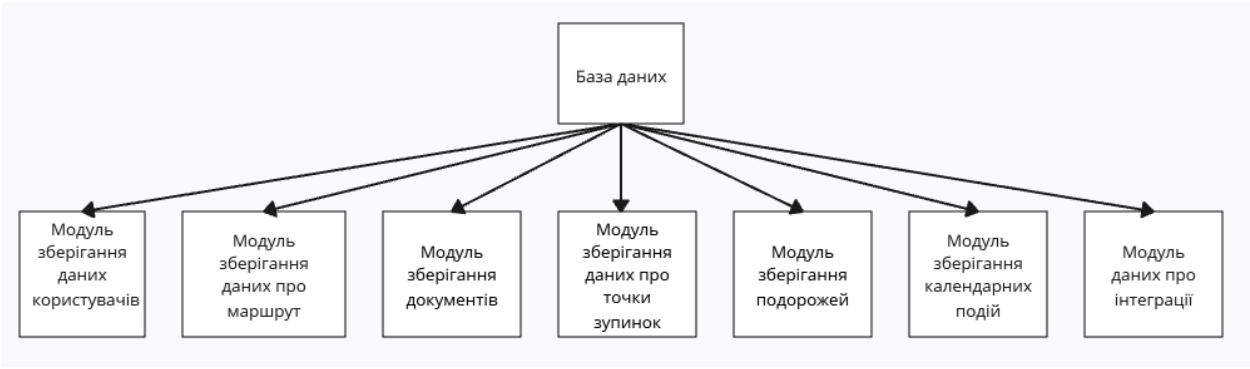

Рис. 2.10. Модулі бази даних

Підсистема інтерфейсу користувача є головною у взаємодії користувача і системи, так як саме вона містить необхідні модулі (рис. 2.11.) задля забезпечення відображення навігації, карт, профілю користувача та все інше, що користувач бачить на своєму екрані.

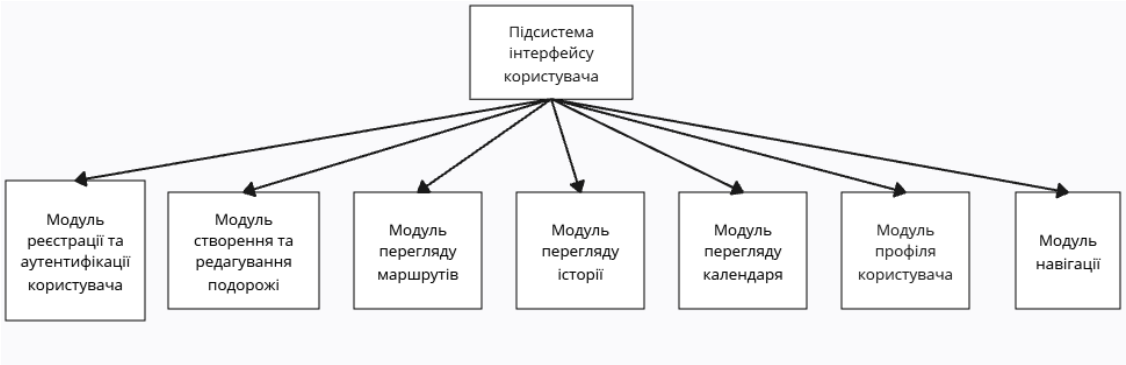

Рис. 2.11. Модулі підсистеми інтерфейсу користувача

Підсистема обробки даних відповідальна за управління даними, які надходять з різних джерел в систему, їх обробку, перевірку на коректність введеного формату, підготовку до подальшого використання.

Схема декомпозиції підсистеми наведена на рис. 2.12.

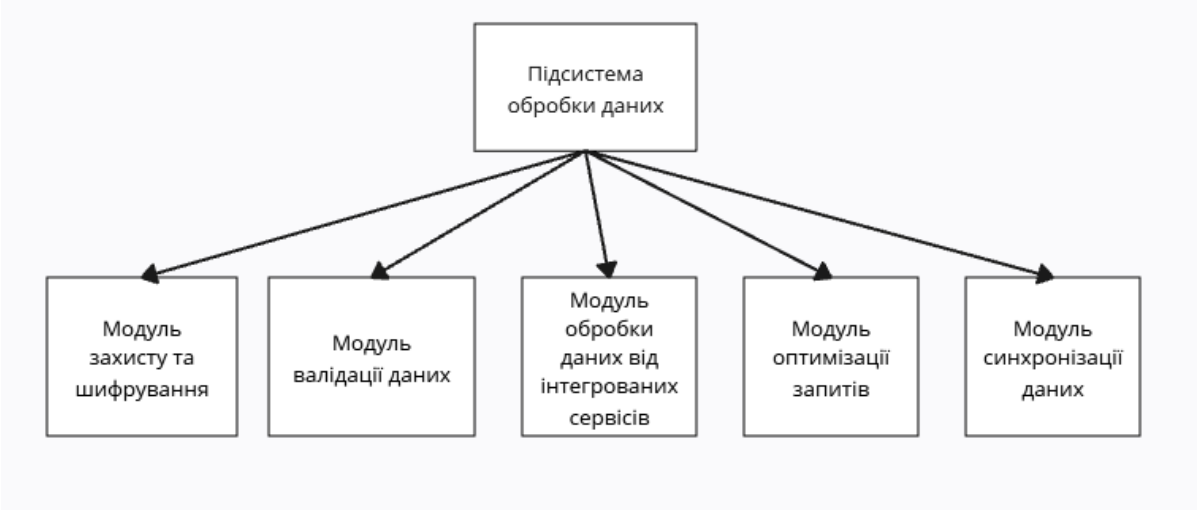

Рис. 2.12. Модулі підсистеми обробки даних

Після детального розгляду необхідних підсистем та їх модулів потрібно визначити їх взаємозв’язок в системі.

Візуалізацію взаємозв’язку підсистем показано на рис. 2.13.

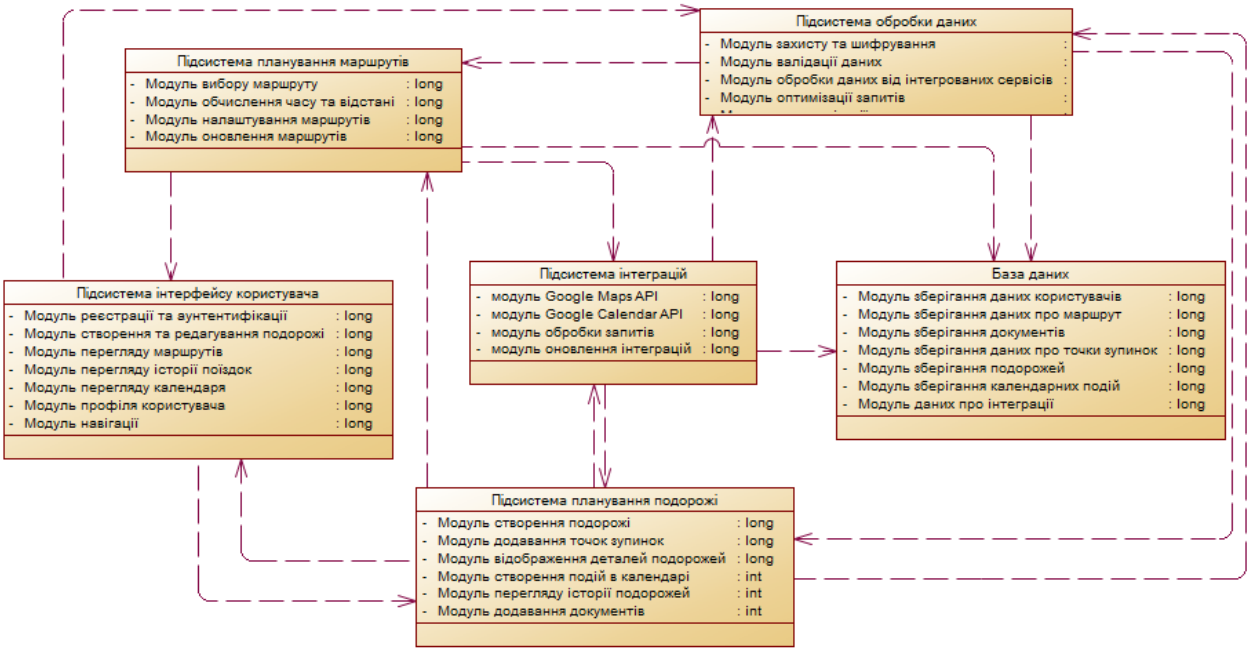

Рис. 2.13. Взаємодія підсистем

Відповідно до наведеної схеми користувач взаємодіє з підсистемою інтерфейсу, надаючи необхідні дані для планування майбутньої поїздки. Підсистема планування подорожі згідно введених даних передає їх до підсистеми планування подорожей, а також формує запит до системи інтеграцій, задля отримання доступу до функцій які містяться в Google Maps API. Система інтеграцій в свою чергу передає дані на обробку і тоді оброблені дані повертаються до системи планування маршрутів задля побудови коректного маршруту. Маршрут зберігається в базі даних.

Звісно, це не весь алгоритм роботи, а тільки опис необхідних дій задля отримання маршруту та інших даних поїздки.

Для кращого розуміння архітектури системи використовують DFD-діаграми, які візуалізують декілька рівнів основних процесів.

DFD-діаграма нульового рівня відображає загальну структуру системи (рис.2.14.) та потоки даних між елементами. Основні елементи: користувач, сервер, база даних, інтегровані сервіси.

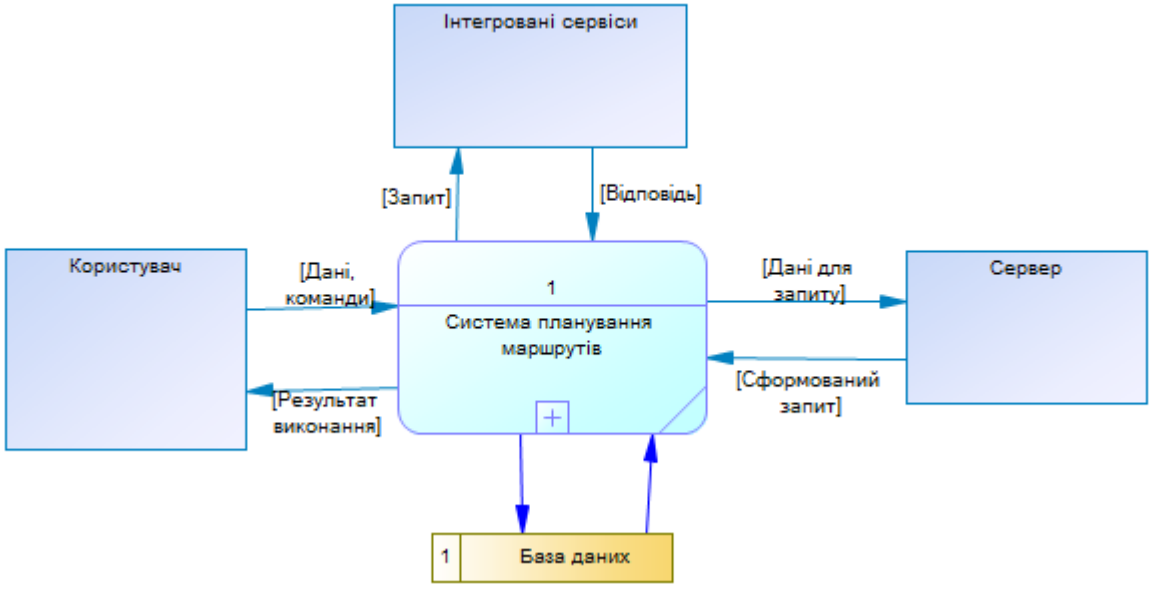

Рис.2.14 DFD-діаграма нульового рівня

DFD-діаграма першого рівня (рис.2.15.) створена для візуалізації основних процесів, зокрема це реєстрація, створення та перегляд подорожі, дії з акаунтом

користувача, можливість перегляду подій в календарі на основі створеної подорожі.

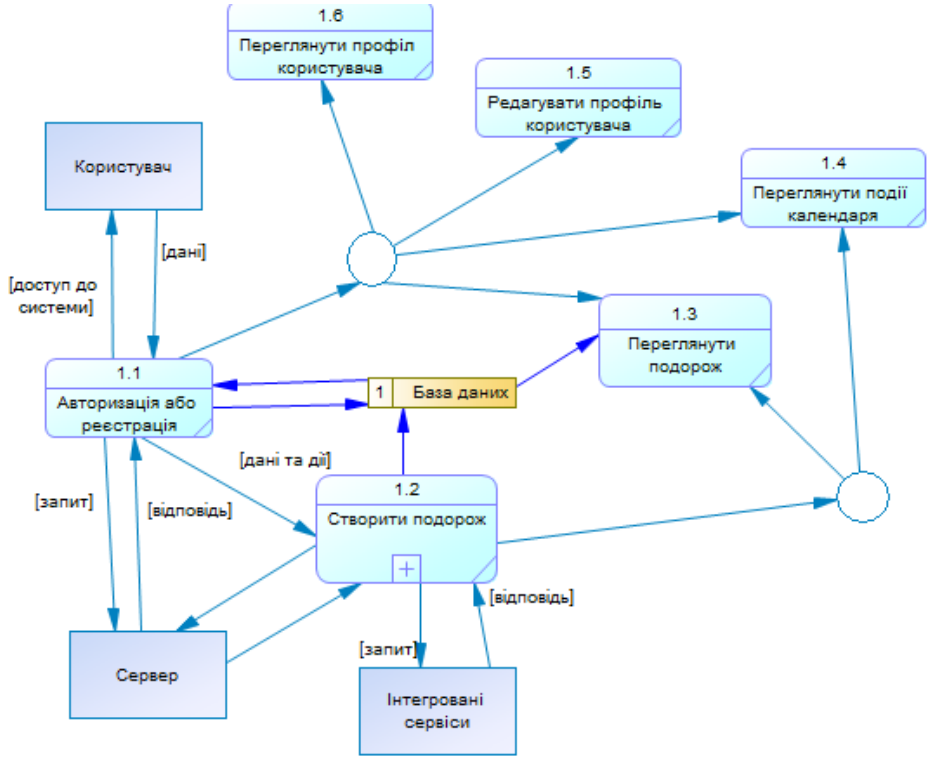

Рис. 2.15. DFD-діаграма першого рівня

DFD-діаграма другого рівня деталізує основні процеси, на рис. 2.16. показано деталізацію процесу створення подорожі.

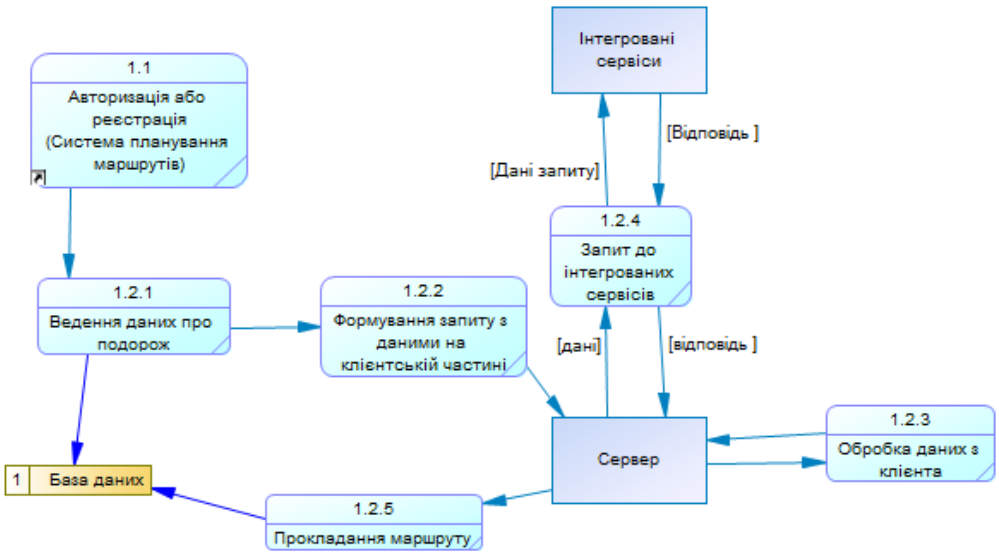

Рис.2.16. Деталізація процесу створення подорожі

2.4. Алгоритм роботи API

З інших пунктів цього розділу стало чітко зрозуміло, що для забезпечення функціоналу даної системи необхідно використовувати API.

API це інтерфейс, що використовується програмістами, задля взаємодії з сервером. Він є набором інструкцій та стандартів, які надають змогу взаємодії зі сторонніми додатками, системи, сервісами. API є простим у використанні, має багато плюсів у використанні, але також і мінусів.

Пропонується розглянути спрощену схему роботи API на рис. 2.17.

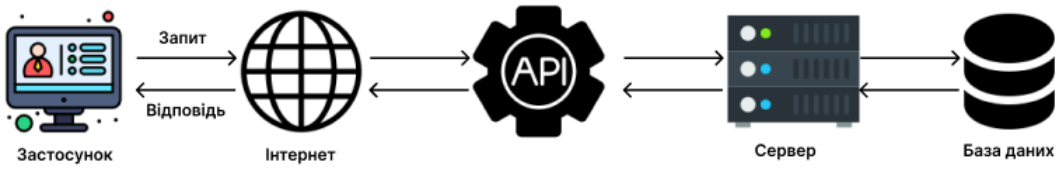

Рис. 2.17. Схема роботи API

Кроки впровадження API в проєкт:

1. Для початку потрібно проаналізувати вимоги проєкту, щоб обрати необхідні сервіси, які надають API ключ, для використання в проєкті.
2. Якщо проєкт потребує специфічних функцій, яких немає в загальнодоступних сервісах можна розробити власне API, це надасть змогу використовувати створені функції у інших проєктах
3. Налаштувати API ключ
4. Створити серверну логіку для обробки запитів до API
5. Розробити механізми обробки помилок та кешування
6. Розробити інтерфейс користувача для взаємодії з сервером з боку користувачів
7. Провести тестування API, тестування продуктивності та безпеки
8. Опис необхідної документації

9. Розгортання API в продакшн, для цього необхідно окрім серверної частини налаштувати також бази даних та інші ресурси. Після цих дій можна виконати розгортання на платформу.

З переваг використання цієї технології є економія ресурсів. Це стосується не тільки бюджету, а також часу розробки, тому що при використанні вже існуючих API не потрібно з нуля розробляти функціонал, робота вже з готовим кодом.

Також плюсом є те, що відразу відомий результат. Функції API вже є прописаними в документації і несуть передбачуваний характер.

Готовий код є очевидним плюсом, але це можна розглядати і з негативної сторони, так як сам розробник цього коду не бачить, це може бути проблемою для нових співробітників, якщо розробник API звільнився з компанії, або пішов з проєкту. Це стосується власного API, який не пов’язаний із загальнодоступними.

При використанні стороннього API на нього може стояти ліміт запитів, тому це безпосередньо треба враховувати при обговоренні бюджету проєкту.

Після розгляду плюсів та мінусів стає очевидним, що існує декілька видів API, які поділяються за типом доступу (схема наведена на рис. 2.18.) та підходом до розробки (або ж стилів).

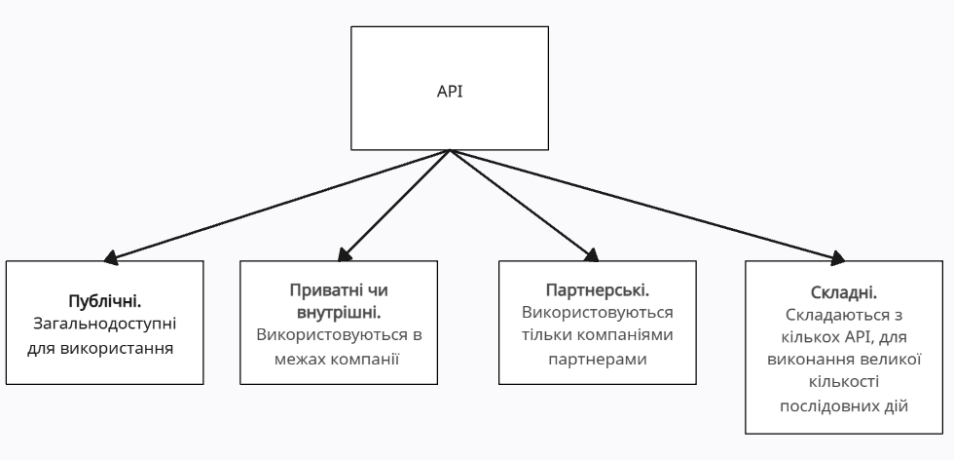

Рис. 2.18. Види API за типом доступу

Кожен розробник може створити своє API, але це не виключає того, що в будь-якому випадку потрібно притримуватись створених протоколів та правил, за якими працюватиме ця розробка.

Задля дотримання цих правил існують різні підходи розробки API, а саме RPC, REST та SOAP. Кожна з них відрізняється та застосовується для різних цілей.

RPC — використовується здебільшого для виклику дій чи процесів, без явного обміну даними та ресурсами, що відрізняє його від REST та SOAP. Підтримує формати кодування JSON та XML. Цей тип не використовується для приватних API.

SOAP — має жорстку структуру та підтримує лише формат XML, використовується задля підвищеної безпеки даних. [28]

REST — спрощений SOAP, використовує URL-адреси та підтримує різні формати даних JSON, XML, звичайний текст. Підходить для взаємодії з базами даних. REST використовується у веб-розробці, так як ідеально підходить для взаємодії клієнтської частини з серверною.

Всі відмінності підходів розробки API наведені в табл. 2.7.

Таблиця 2.7.

Характеристика стилів API

Параметр	RPC	SOAP	REST
Тип взаємодії	Виклик методів на віддаленому сервері	Виклик функцій з серверів через XML	Взаємодія через HTTP
Протокол	HTTP, UDP, TCP, тощо	HTTP, UDP, TCP, тощо	HTTP, HTTPS
Формат даних	Бінарні дані, JSON XML, Protocol Buffers	XML	JSON та XML
Архітектура	Процедурна	Повідомлення-орієнтована	Ресурсно-орієнтована
Використання	Для виклику віддалених функцій або методів	Для великих корпоративних інтеграцій з строгими умовами	Веб-служби, мобільні додатки, інтеграції
Обробка помилок	Немає стандартизованої обробки	Стандартизована обробка з повідомленням	Обробка помилок через HTTP статуси

2.5. Вибір та моделювання бази даних

Бази даних потрібні для зберігання даних з певною структурою та різним об’ємом. Вони не мають обмеженості на кількість інформації, дозволяють редагувати дані, видаляти, оновлювати. База даних — це організована колекція даних, яка зберігається і управляється за допомогою програмного забезпечення. Дані можуть зберігатись як на жорсткому диску комп’ютера або в хмарі, в залежності від вимог.

Спосіб взаємодії користувача з базою даних, яка зберігається на жорсткому диску, зображено на рис. 2.19.

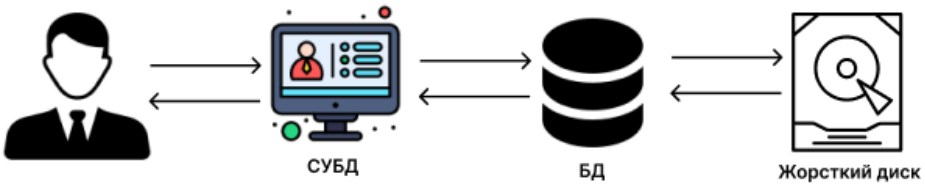

Рис. 2.19. Взаємодія з базою даних

Бази даних є різних моделей, кожна з яких підходить під певний перелік задач. Першими комп’ютерними системами для зберігання даних були ієрархічні та мережеві бази даних.

Ієрархічні бази даних мають деревоподібну структуру у вигляду «батьки-нащадки». Між головним елементом та нащадками є зв’язок «один до багатьох».

Схема моделі наведена на рис. 2.20.

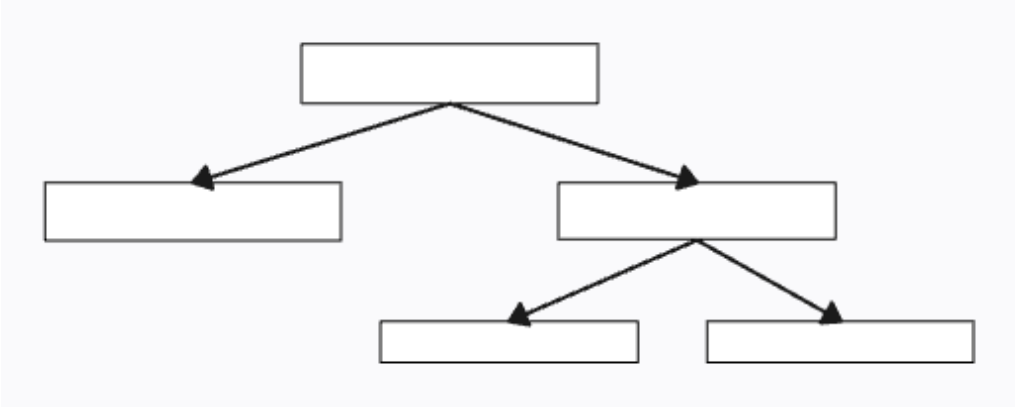

Рис. 2.20. Ієрархічна модель бази даних

Протилежністю до цієї моделі є мережна модель баз даних. Ця модель підтримує зв’язок між елементами «багато до багатьох». Вона не має «головного елемента», всі об’єкти рівні.

Схема моделі наведена на рис. 2.21

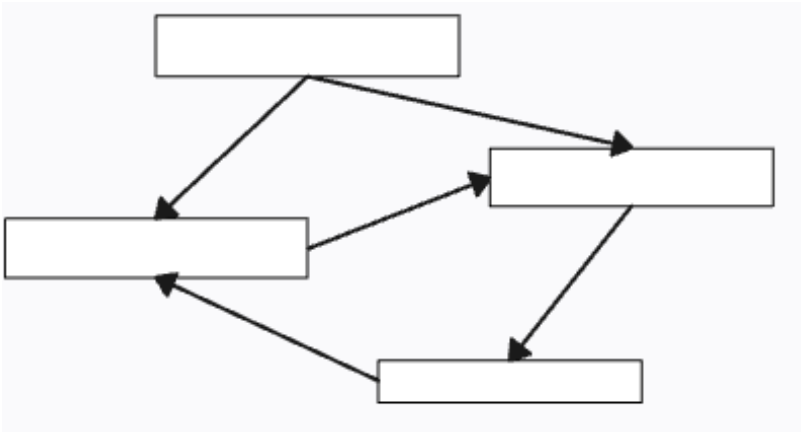

Рис. 2.21. Мережна модель

Через складність розробки даних моделей була винайдена реляційна база даних. БД зберігає дані у вигляді таблиці, зв’язки між ними встановлюються через ключі. Таблиця складається зі стовпців та рядів. Для взаємодії з цими таблицями була розроблена мова SQL, саме завдяки ній можна робити маніпуляції з даними. Реляційні БД базуються на відношеннях між даними у формі таблиць. Приклад таблиці наведено в табл. 2.8.

Таблиця 2.8.

Приклад даних в реляційній базі даних

ID	First_name	DateOfBirst	Class	Hight
1	Andrii	04.06.2010	9	178
2	Mary	06.03.2012	7	152

Кожна таблиця повинна відповідати переліку вимог, для коректної роботи з БД:

- Унікальне ім’я для стовпців
- Всі дані стовпця відносяться до одного типу даних
- Рядки повинні відрізнятись вмістом як мінімум одного стовпця

- Таблиця повинна містити стовпець-ключ

У реляційних БД таблиці — це відношення, стовпці — поля, а рядки — записи. Підтримуються різні типи зв’язків між таблицями, на відміну від ієрархічних та мережних БД [29].

Задля перекриття недоліків реляційних БД були розроблені нереляційні, або ж NoSQL БД. Ця модель зберігає неструктуровані дані. Вона є документно-орієнтованою, отже може зберігати дані у вигляді документів, наприклад, JSON.

Також існують об’єктно-орієнтовні БД, вони можуть зберігати не тільки дані, а і самі методи для їх обробки. Об’єкти в такій базі формують за класами та підтримують принципи ООП.

Порівняння моделей баз даних відображено в табл. 2.9.

Таблиця 2.9.

Порівняння моделей баз даних

Модель	Використання	Переваги	Недоліки
Реляційна	Для фінансових систем, управління запасами, даними клієнтів, тощо	Чітка структура	Проблеми з масштабованістю для великих обсягів даних
		Забезпечення цілісності даних	Висока складність налаштування для великих та складних систем
		SQL	Не підходить для неструктурованих даних
Мережна	Моделювання складних зв’язків між елементами. Логістика, транспортні, телефонні мережі	Підходить для складних структур даних	Важке управління у порівнянні з реляційними базами
Ієрархічна	Для зберігання структурованих даних, що мають чітку ієрархічну організацію.	Простота структури для зберігання даних з чіткими ієрархічним зв’язком	Не гнучка у змінах структури даних
		Висока продуктивність для обмеженого набору даних	Ускладнення при наявності складних зв’язків між даними

Продовження таблиці 2.9.

Модель	Використання	Переваги	Недоліки
NoSQL	Для веб-додатків, соціальних мереж, обробки великих даних, інтернет-магазинів	Підтримка різних моделей зберігання даних	Немає стандартів запитів
		Підходить для неструктурованих даних	Не завжди підтримує транзакції
		Легка в масштабуванні	Потребує специфічних знань та досвіду
Об’єктно-орієнтовна	CAD системи, медичні інформаційні системи, зберігання складних об’єктів	Інтеграція з ООП мовами	Не поширена та підтримувана в порівнянні з реляційними
		Зберігання складних типів даних	Проблеми з масштабованістю для великих обсягів даних
		Підтримка принципів ООП	Має складності в управлінні, потребує більше досвіду

Для веб-системи планування подорожі оптимальним варіантом буде реляційна модель бази даних, оскільки вона проста у використанні та маю чітку структуру, добре підходить для зберігання даних про подорожі.

Перед розробкою бази даних необхідно провести аналіз вимог до системи, що розробляється, щоб уникнути подальших проблем. Визначити які саме дані будуть зберігатись, типи даних та зв’язки між ними.

На основі проведеного вище аналізу, визначеного функціоналу була розроблена концептуальна модель бази даних (рис. 2.22.).

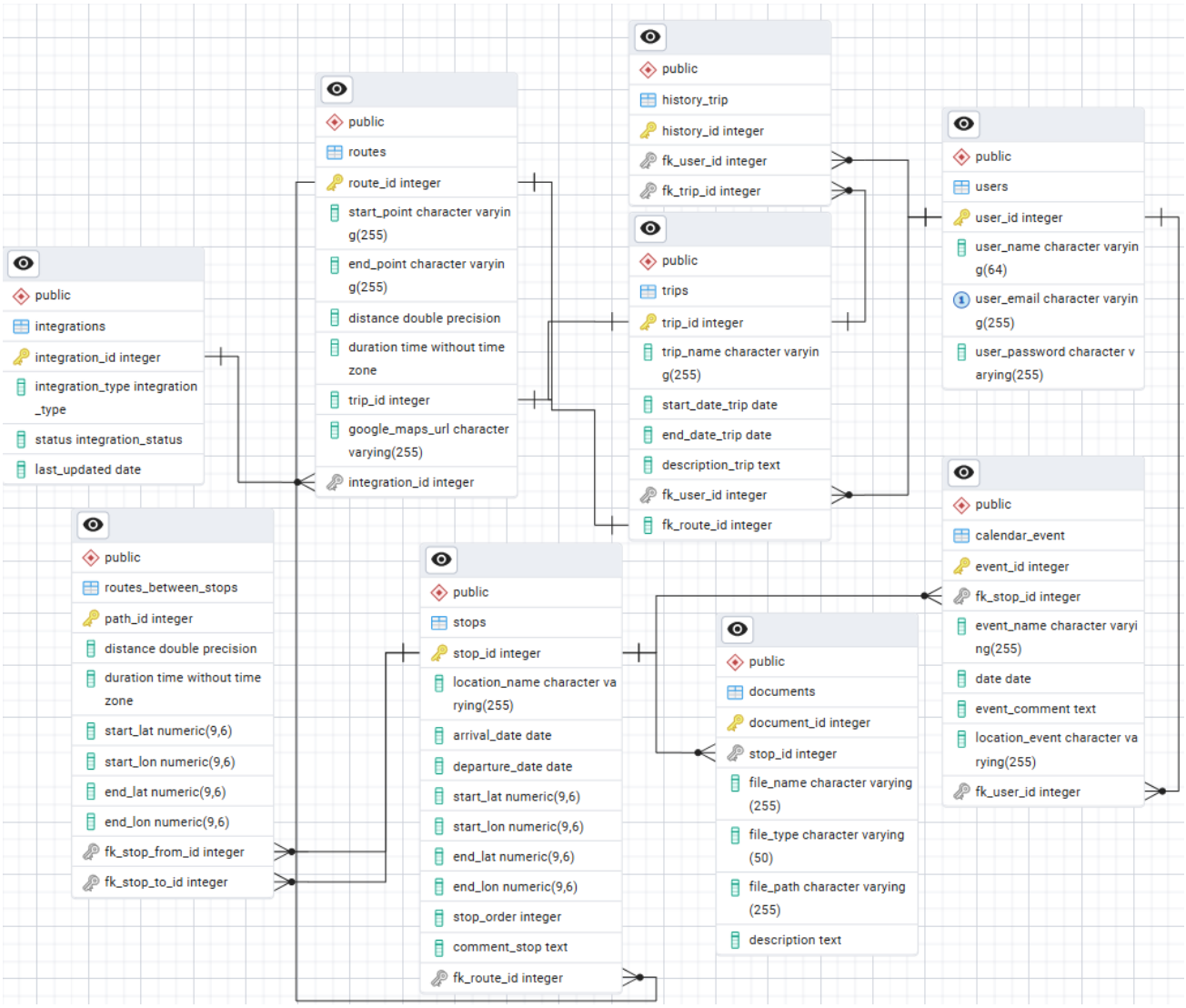

Рис. 2.22. Концептуальна модель бази даних

Таблиця даних про користувачів «users» містить поля з унікальним id для кожного користувача, ім’я, електронну адресу та зашифрований пароль. Типи даних для полів вказані в табл. 2.10.

Таблиця. 2.10.

Типи даних для «users»

Поля	Типи даних
user_id	integer внутрішній ключ
user_name	varchar(64)
user_email	varchar(255) унікальні значення
user_password	varchar(255)

«users» має зв’язки з таблицею «calendar_event» зі зв’язком «один до багатьох», що означає, що один користувач може мати багато подій в календарі. А також з таблицями «trips», «history_trip» зв’язок також «один до багатьох», так як користувач може мати багато поїздок відповідно кожна з них має історію поїздки (рис. 2.23.). Типи даних для цих таблиць описані в табл. 2.11.

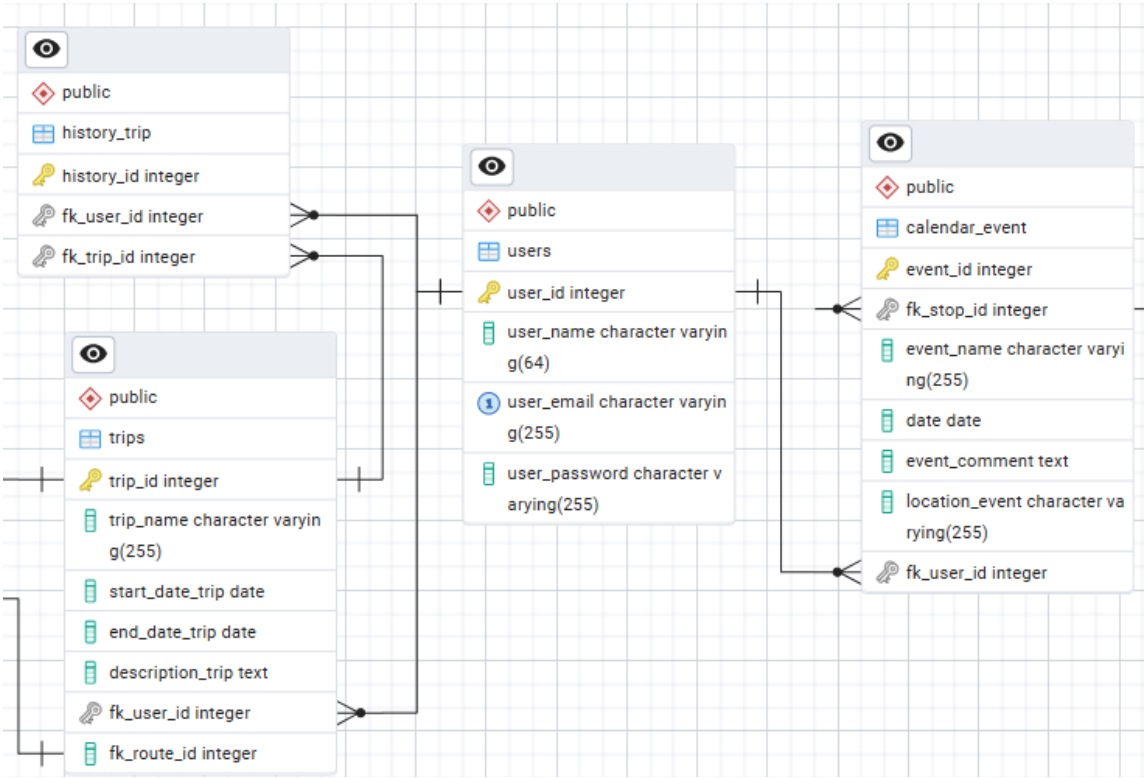

Рис. 2.23. З’єдання для «users»

Таблиця поїдки має зв’язок з «users», «history_trip» та «routes» (рис. 2.24.). Відношення до «routes» «один до одного». В межах однієї подорожі розрахований один маршрут, з урахуванням зупинок і підмаршрутів. Таблиці з’єднані унікальними id, а саме «trip_id» та «route_id»

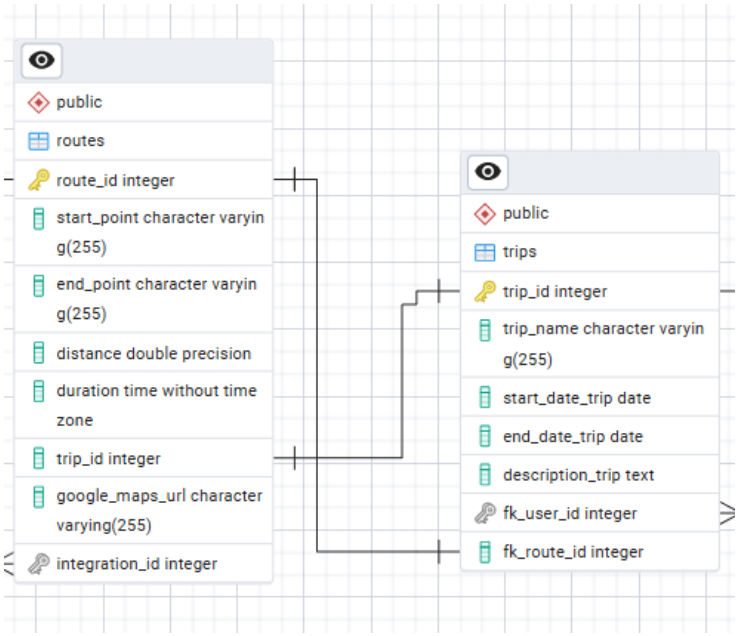

Рис. 2.24. З’єднання для «trips»

Таблиця маршруту має безпосереднє з’єднання з таблицею інтеграцій та зупинками, які входять до маршруту (рис. 2.25.). Зв’язки між «integrations» та «routes» «один до багатьох», між «routes» та «stops» також «один до багатьох».

Таблиця зупинок має з’єднання з «documents», «calendar_event» та «routes_betweeb_stops» з типом «один до багатьох» до кожної з таблиць (рис. 2.26.).

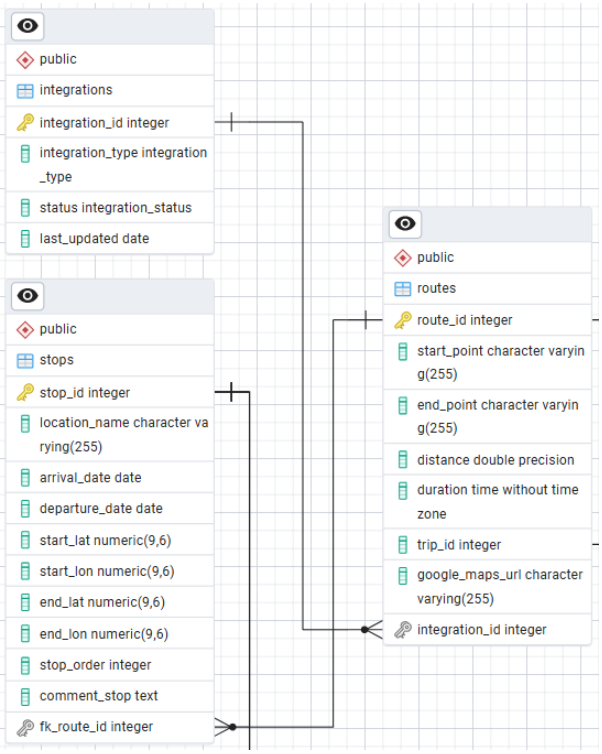

Рис. 2.25. Схема з’єднань для «routes»

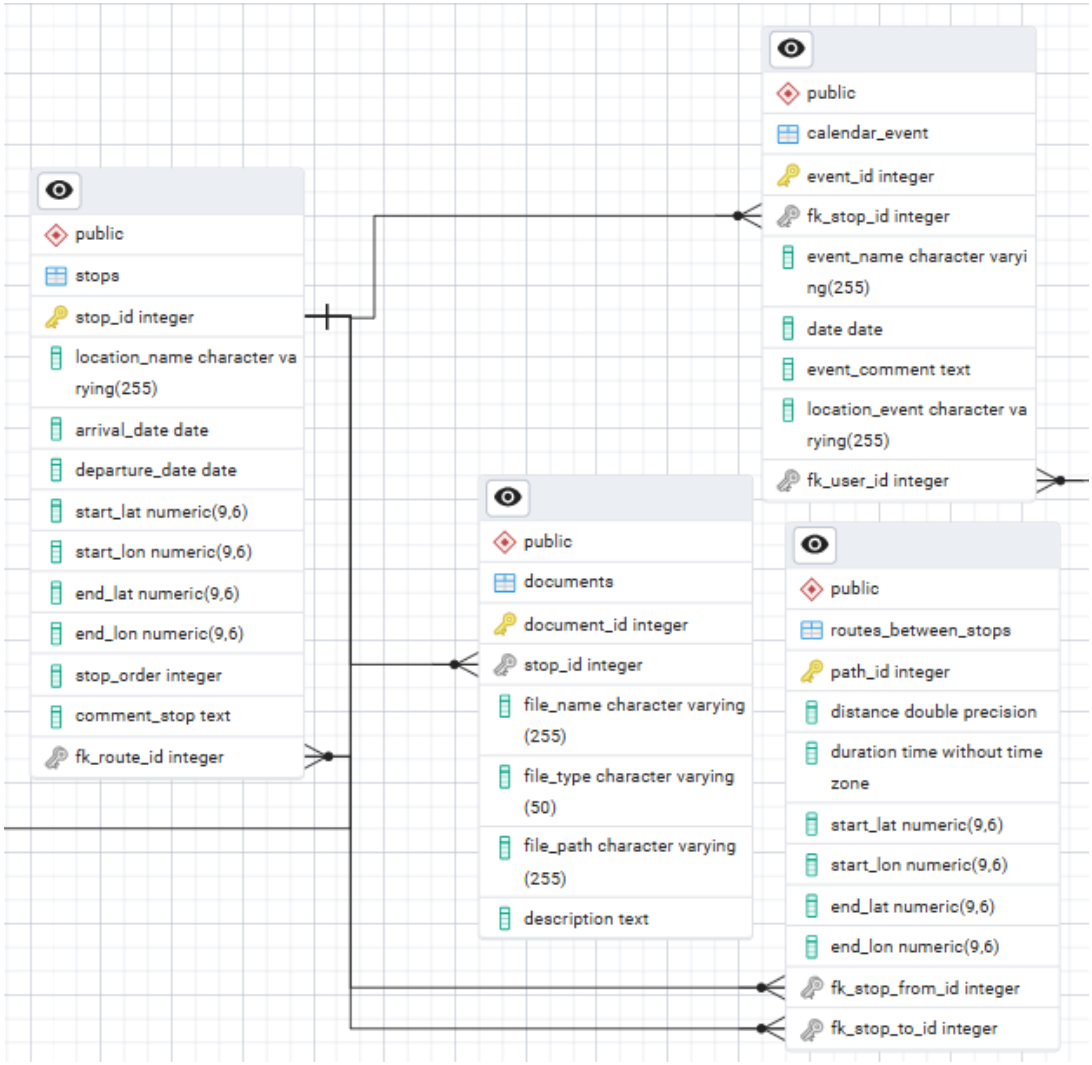

Рис. 2.26 Схема з’єднань для «stops»

Таблиця 2.11.

Таблиця типів даних

documents		trips	
document_id	integer внутрішній ключ	trip_id	integer внутрішній ключ
file_name	varchar (255)	trip_name	character varying(255
file_type	varchar (50)	start_date_trip, end_date_trip	date
file_path	varchar (255)	description_trip	text
description	text	fk_user_id, fk_route_id	integer зовнішній ключ

Продовження табл. 2.11

Routes		routes_between_stops	
route_id	integer внутрішній ключ	path_id	integer внутрішній ключ
start_point	character varying(255)	distance	double
end_point	character varying(255)	duration	time
distance	double	start_lat	numeric(9, 6)
duration	time	start_lon	numeric(9, 6)
trip_id	integer зовнішній ключ	end_lat	numeric(9, 6)
google_maps_url	character varying(255)	end_lon	numeric(9, 6)
integration_id	integer зовнішній ключ	fk_stop_from_id, fk_stop_to_id	integer зовнішній ключ

Продовження табл. 2.11

history_trip		integrations	
history_id	integer внутрішній ключ	integration_id	integer внутрішній ключ
fk_user_id	integer зовнішній ключ	integration_type	integration_type (власний тип даних)
fk_trip_id	integer зовнішній ключ	status	integration_status (власний тип даних)
		last_updated	date

Висновки до розділу 2

У розділі було розглянуто вимоги до веб-системи для планування подорожей, а також основні аспекти її архітектури. Основні функціональні вимоги:

- Забезпечити можливість створення маршрутів.
- Збереження подій в календар.
- Безпечна авторизація користувачів.
- Забезпечити інтеграцію з картографічними сервісами.

З огляду на архітектуру системи можна визначити, що архітектура є клієнт-серверною та має модульних підхід. Клієнтська частина це інтерфейс користувача для взаємодії з системою, серверна частина обробляє введенні дані та забезпечує інші функціональні частини системи.

Основні модулі системи:

- Підсистема планування маршрутів
- Підсистема інтеграцій
- Підсистема планування подорожі
- База даних
- Підсистема інтерфейсу користувача

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-СИСТЕМИ

3.1. Вибір стеку технологій

Сучасні технології налічують багато інструментів для реалізації веб-продуктів, так як на даний момент веб-системи є важливою частиною Інтернет простору.

Розробка веб-системи містить багато етапів, починаючи від аналізу вимог щодо майбутнього продукту. Після проектування інтерфейсу дизайнерами його потрібно програмно сформувати – це задача Front-end розробника. На разі існує 3 базові інструменти для розробки інтерфейсів, це HTML, JavaScript та CSS та допоміжні бібліотеки, фреймворки, плагіни. Розглянемо детально кожен із цих засобів.

HTML (HyperText Markup Language) – мова гіпертекстової розмітки, яка використовується для створення веб-сторінок, саме завдяки їй браузер відображає контент на сторінці. HTML містить багато тегів, що надають можливість додавання відео, фото, посилань та інших елементів на сторінку. Але для створення повноцінного продукту її звісно недостатньо, навіть для простого односторінкового веб-сайту потрібно разом з нею використовувати CSS.

CSS використовують для стилізації сторінки, це візуальна частина html документу. За допомогою цього інструменту можна додавати динамічності для сторінки, наприклад, анімаціями. Але використання тільки HTML і CSS може підійти для веб-сайтів, які хочуть продемонструвати якусь інформацію, без взаємодії з користувачем. Такі продукти не є функціональними, навіть для мінімальних функцій, наприклад, відкрити меню на веб-сайті, потрібно використовувати JavaScript.

JavaScript є єдиною мовою яка підтримується у всіх браузерах, що робить її незамінною. За допомогою неї можна створювати складні анімації, збирати дані, додавати різні інтерактивні елементи. Для полегшення та оптимізації роботи в складних проектах сучасні програмісти використовують фреймворки та

бібліотеки, найпопулярніші з них станом на 2024 рік за статистикою (рис. 3.1.) на GitHub стали Vue.js, React.js і Angular

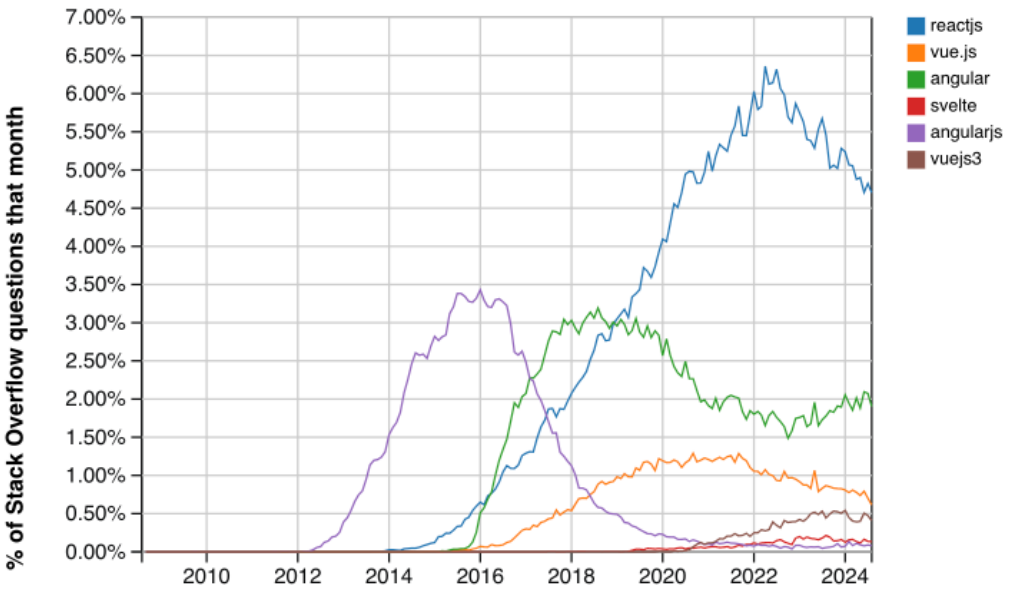

рис. 3.1. Статистика використання фреймворків

Першим, з вищезгаданих відомих продуктів, з'явився Angular. Цей фреймворк випустила компанія Google в 2010 році. На той момент це був AngularJS, який повністю переписали в 2016 році, на разі сучасний Angular, з оновленнями кожного пів року. На відміну від React.js та Vue.js, які базуються на мові JavaScript, Angular працює на основі TypeScript. А також даний фреймворк не працює з віртуальним DOM, що не підходить для створення інтерфейсів, але Angular чудово підходить для CRM систем, та в загалом роботи з даними, так як TypeScript має чітку структуризацію коду та статичний тип даних.

В 2013 році на світовому ринку з'явився продукт від компанії Meta (раніше Facebook) – React.js. React.js – бібліотека JavaScript, яка найчастіше використовується розробниками на даний момент, з поміж інших бібліотек. Спираючись на документацію React.js розроблений для побудови інтерфейсів користувача з компонентів . Особливістю даної бібліотеки є ефективне управління DOM елементами, а також вона має свій синтаксис під назвою JSX. JSX дозволяє

тісно взаємодіяти з HTML, тому це допомагає більш оптимізувати та мінімізувати код, а це в свою чергу позитивно впливає на майбутню продукцію.

В 2015 році Meta, випустила фреймворк React Native, який створено виключно для розробки мобільних додатків. Особливість даного фреймворка це кросплатформність, так як код написаний на React Native використовується як для iOS так і для Android.

Натхненний роботою з AngularJS, розробник Еван Ю у 2014 році випустив Vue.js – легкий фреймворк для створення інтерфейсів користувача та односторінкових веб-додатків. На відміну від Angular, Vue має простішу структуру та дозволяє швидше почати розробку. Як і React, Vue використовує віртуальний DOM для оптимізації змін у справжньому DOM, що покращує продуктивність. На відміну від React, Vue має вбудовану підтримку реактивності без потреби у зовнішніх бібліотеках. Завдяки своїй гнучкості та легкості Vue часто описують як сучасний фреймворк для створення інтерфейсу веб - застосунків.

Підсумувавши вище сказане можна скласти порівняльну таблицю для допоміжних засобів JavaScript (табл. 3.1.)

Таблиця 3.1.

Характеристики JavaScript фреймворків

Параметр	Vue.js	React	Angular
Тип	Фреймворк	Бібліотека	Фреймворк
Мова	JavaScript	JavaScript	TypeScript
Двостороння прив’язка даних	Так	Ні	Так
Віртуальний DOM	Так	Так	Ні
Гнучкість	Висока	Висока	Менша
Приклад використання	Легкі та швидкі застосунки, невеликі проєкти, PWA, інтерактивні UI	Динамічні UI, SPA, великі масштабовані застосунки, інтеграція у проєкти	Корпоративні системи, складні SPA, проєкти з чіткою архітектурою
Підтримка компаній	Open-source	Meta	Google

Звісно, існують також засоби, які використовують окрім JavaScript, такою мовою є TypeScript.

TypeScript (TS) є надмножиною JavaScript і був випущений Microsoft у 2012 році. Це означає що ця мова містить в собі всі функції JavaScript, але має додатковий функціонал та деякі відмінності. Основною особливістю є розширений набір типів даних. TypeScript використовують для покращення якості програмного забезпечення, через статичну типізацію TypeScript виявляє помилки ще до виконання коду, це є великим плюсом. Так, у JavaScript динамічна типізація і часто це призводить до помилок, особливо якщо працювати зі збором числових даних і операцій в майбутньому над ними. Але код написаний за допомогою TypeScript все одно компілюється у JavaScript-код.

На разі TypeScript стрімко розвивається. За версією, DOU найпопулярнішою мовою програмування на українському ринку станом на 2025 стала TypeScript, але ще в минулому 2024 році JavaScript займав перше місце серед мов. Відображення статистики мов програмування за популярністю за 2024 рік – рис.3.2., за 2025 рік– рис.3.3.

рис. 3.2. Статистика популярності мов програмування на DOU за 2024 рік

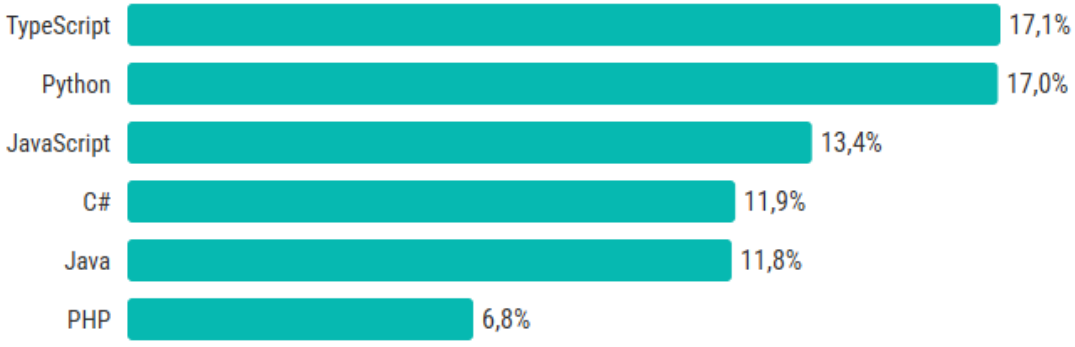

рис. 3.3. Статистика популярності мов програмування на DOU за 2025 рік

Враховуючи проведений вище аналіз використання засобів програмування для проекту були обрані наступні технології та бібліотеки:

JavaScript: основна мова програмування. Без JavaScript неможливо створити сучасну веб-систему, в незалежності від виду цієї систем, окрім статичних веб-сторінок. Мова містить багато бібліотек, надає всі необхідні інструменти для створення інтерфейсу користувача.

Node.js: використовується для створення серверної частини додатку. В системі планування подорожей буде забезпечувати інтеграцію з зовнішніми сервісами, обробку даних від користувачів, створювати зв’язок з базою даних.

React.js: забезпечує взаємодію користувача з серверною частиною додатку. Використовується для побудови інтерфейсів, завдяки компонентному підходу зручний при розробці.

Axios: бібліотека для виконання HTTP запитів. Так як в проекті використовується REST API ця бібліотека необхідна для обробки запитів та відповідей від серверу

React Google Maps Api (@react-google-maps/api): бібліотека, яка забезпечує інтеграцію з Google Maps по створеному API ключу в системі Google.

Стандарт OAuth 2.0: дозволяє авторизацію через акаунт Google в додатку. В проєкті необхідний для реалізації запису подій в Google Calendar

JSON Web Tokens: для авторизації користувачів недостатньо лише стандарту OAuth 2.0, для цієї задачі потрібно створювати токен.

Googleapis: бібліотека дозволяє доступ до API Google, а також потрібна для механізму авторизації.

Express: фреймворк який спрощує створення серверної частини додатку.

Dotenv: бібліотека для інтеграції даних з конфігураційних файлів.

Axios: бібліотека що дозволяє роботу з HTTP запитами з клієнтської та серверної частини.

@react-oauth/google: бібліотека для інтеграції OAuth 2.0 Google в додаток. Забезпечує один зі способів авторизації в додатку.

Cors: механізм, який дозволяє серверам визначати з якими доменами допущена їх взаємодія.

Css: мова стилів, потрібна для зовнішнього вигляду додатку.

HTML: мова гіпертекстової розмітки.

3.2. Реалізовані алгоритми та компоненти

У ході розробки системи був використаний певний набір алгоритмів для забезпечення коректної роботи відповідно стандартів.

3.2.1. Алгоритм авторизації користувачів

Один з важливих алгоритмів є алгоритм авторизації користувачів. У сучасних системах для авторизації використовується стандарт OAut 2. Це протокол авторизації, який дозволяє стороннім додаткам отримати дані користувачів. OAut 2 містить декілька грантів завдяки яким дані користувачів залишаються в безпеці при авторизації на сторонніх серверах.

Гранти OAut 2 [30]:

1. Authorization Code Grant: авторизація за допомогою одноразового коду, створеного сервером, який в подальшому обмінюється на токен. Однак, даний грант сам по собі не має певних заходів безпеки і з точки зору кібербезпеки він повинен бути доповнений PKCE
2. Implicit Grant: токен безпосередньо повертається клієнту. Застарілий варіант, який зараз не використовується
3. Authorization Code Grant with Proof Key: містить надання коду авторизації, як у першому гранті, але містить додаткові коди, що робить його більш безпечний.
4. Resource Owner Credential Grant: вимогає від користувача облікових даних, працює без перенаправлення на сторонні сервіси, надає access token і refresh token. Вимагає додаткових заходів безпеки.

5. Client Credential Grant: не потребує участі користувача, додаток самостійно аунтифікується на сервері авторизації, використовуючи ClientId та ClientSecret. Має певні заходи безпеки, оскільки містить ClientSecret, зловмисник зможе дістатись до даних користувачів тільки при його наявності.
6. Device Autorization Flow: використовується для пристроїв, на яких є обмеження введення. Для авторизації потрібне підтвердження на іншому пристрої користувача. Використовує user_code та device_code.
7. Refresh Token Grant: включає обмін access token на refresh token при виявленні підозрілих дій збоку клієнта.

Для побудови алгоритму авторизації потрібно розглянути поняття токену, його структуру, види, та інші деталі.

Токен – це набір даних, використовується для авторизації та аунтифікації користувачів. На заміну введення пароля кожного разу користувачем, він отримує спеціальний токен при першому успішному вході в систему.

Токен може бути різного формату та типу даних, наприклад, JSON формат або електронний ключ.

Найбезпечнішим типом токенів є JWT (JSON Web Token) – стандарт, який визначає безпечний спосіб передачі інформації. Інформація, що передається, повинна мати цифровий підпис, використовується алгоритм HMAC, або ж пара відкритого та приватного ключа.

JWT складається з трьох частин header, payload та signature. Header містить інформацію про токен, його тип, алгоритм шифрування. Payload це основні дані, які передаються. Signature це підпис даних, забезпечує цілісність та безпеку даних.

Приклад структури JWT на рис.3.4.

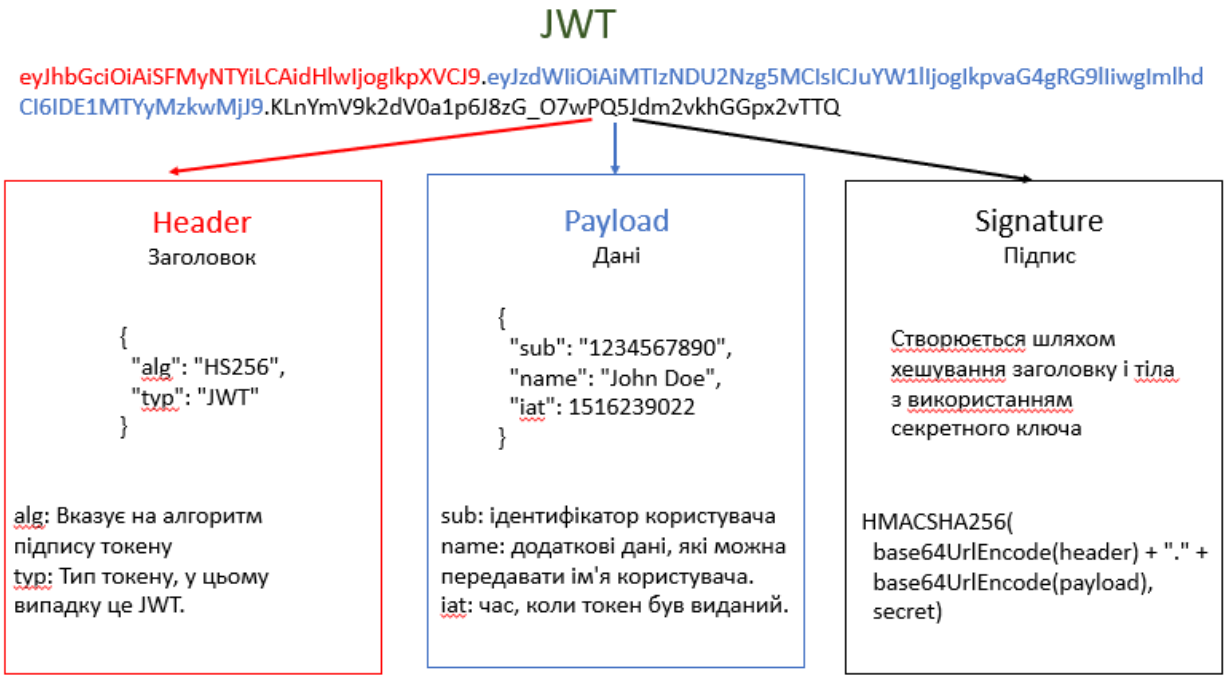

Рис.3.4. Структура JWT

Токени також поділяються на access token та refresh token. В сучасних алгоритмах авторизації вони використовуються парі для безпеки користувачів.

Access token використовується для авторизації користувачів, має обмежений термін використання (15-20 хв), зберігається в LocalStorage та сам по собі не є захищеним.

Refresh token використовується для оновлення access token, щоб користувач не здійснював вхід в систему кожного разу коли access token вийшов з терміну придатності. Цей тип токену має великий термін придатності визначений розробником та зберігається в Cookie(+httpOnly), що є більш захищено.

У веб-системі планування подорожей за визначеними вимогами передбачається авторизація через Google акаунт. Для цієї задачі в консолі для розробників від Google майбутня система була зареєстрована як веб-додаток, для отримання ClientId та client_secret. Таким чином сервер авторизації буде розуміти, що даний додаток є безпечним для користувачів. Це є одним із заходів безпеки в розробленому алгоритмі авторизації . Ще одним заходом безпеки є використання гранту «Refresh Token Grant» разом з «Authorization Code Grant».

Розроблений алгоритм авторизації в системі показано на рис.3.5.

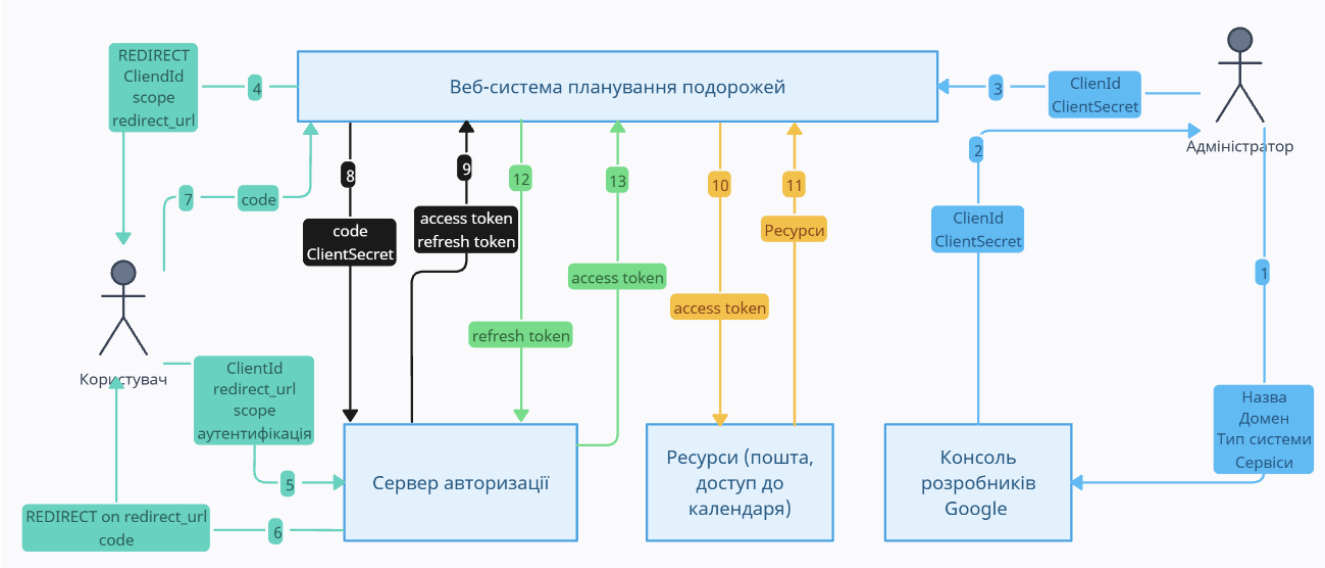

Рис.3.5. Алгоритм авторизації користувачів

Цифри на рис.3.5. є кроками виконання алгоритму. Кроки 1-3 є частиною «Client Credential Grant», а саме:

1. Систему потрібно зареєструвати в консолі розробників, в даному випадку, це консоль розробників Google. Для цього надати їй назву системи, домен, на якому розміщено (не обов'язковий пункт на стадії розробки), тип системи та з якими сервісами буде взаємодія (наприклад, Google Calendar)
2. Після надання необхідних даних для реєстрації системи формується ClientId та ClientSecret. ClientSecret є ключом до даних, тому при збережені потрібно застосовувати методи шифрування.
3. ClientId та ClientSecret потрібно інтегрувати в систему для використання

Кроки 4-7, це кроки взаємодії системи з користувачем, користувача з сервером авторизації:

4. При спробі входу система перенаправляє користувача на сервер авторизації, при цьому передає ClientId, scope та посилання перенаправлення. Scope описують на які ресурси потрібен доступ та дії з ними, наприклад, читання фотографій.

5. Користувач проходить стандартну аунтифікацію на сервері авторизації, на який було перенаправлення. Разом з цим передаються дані, які були отримані від системи.
6. Користувач отримує code, що є частиною «Authorization Code Grant» та перенаправлення до системи, в яку здійснюється вхід.
7. Система отримує code.

Кроки 8-9 описують частину отримання access token та refresh token. Крок 8 – система передає серверу авторизації отриманий code. Крок 9 – генерація токенів від серверу.

Кроки 10-11 це отримання доступу до даних користувача та дій з ними зі сторонніх ресурсів.

Кроки 12-13 це оновлення токенів. Оновлення токенів потрібне в ситуаціях:

- Термін дії access token скінчився
- Помічені дії, які можуть загрожувати безпеці даних користувача.
- Перевищення ліміту використання токену.
- Оновлення після відновлення сесії користувача.
- Зміна налаштувань в системі

Програмно процес отриманні токену реалізується за допомогою HTTP-запитів з боку клієнтської частини та серверної. За допомогою використання необхідних бібліотек код для клієнтської частини отримання токену виглядає наступним чином:

```
const googleLogin = useGoogleLogin({
    onSuccess: async ({ code }) => {
        try {
            const { data } = await axios
            .post('http://localhost:3001/auth/google', { code })
        } catch (error) {
            console.error('Error exchanging code:', error);
        }
    },
    onError: (error) => {
        console.error('Google Login Error:', error);
    },
    flow: 'auth-code',
    scope: 'openid email profile https://www.googleapis.com/auth/calendar',
```

```
});
```

На серверній частині отримання коду авторизації від клієнта та обмін його на токен відбувається наступним чином:

```
app.post('/auth/google', async (req, res) => {
  try {
    const { tokens } = await oAuth2Client.getToken(req.body.code);
    if (tokens.refresh_token) {
      storedRefreshToken = tokens.refresh_token;    }
    res.json(tokens);
  } catch (error) {
    res.status(400).json({ error: 'Failed to exchange code' });
  }
});
```

Клієнт надсилає свій код авторизації попередньо сформований в oAuth2Client. Використовую метод getToken() ці дані обробляються і надсилається запит до Google на отримання пари токенів access та refresh. Після отримання пари токенів перевіряється чи дійсно наявний refresh_token, за позитивної відповіді він зберігається в окрему змінну для подальших операцій. Токени повертаються до клієнту за допомогою json(tokens).

Результатом авторизації користувачів є повернення refresh токену з можливістю подальшого використання.

При тестуванні алгоритму отриманий очікуваний результат (рис.3.6.), що свідчить про те, що алгоритм готовий до використання.

Рис.3.6. Результат тестування алгоритму авторизації

3.2.2. Алгоритм запису подій в календар

Даний алгоритм запису подій в Google Calendar розрахований на авторизацію через Google акаунт, для отримання доступу до календаря. Тож, перше, що необхідно реалізувати це авторизацію за попереднім алгоритмом.

Для реалізації також необхідний компонент форми для заповнення даних. Форма повинна містити наступні поля: місце події, коментарі (за необхідності), дату початку та дату кінця. Місце події користувач може ввести вручну, або поставити маркер на карті.

Додавання подій відбувається за сформованим запитом до Google Calendar API. Calendar API треба підключити до зареєстрованого проєкту в консолі розробників.

Програмна реалізація цього коду містить в собі серверну та клієнтську частину. Формування та відправка запиту відбувається в клієнтській частині та має наступний вигляд:

```
axios.post('http://localhost:3001/auth/calendar',{
            location,
            description,
            startDateTime,
            endDateTime})
        .then(response=>console.log(response.data))
        .catch(error=>console.log(error.message))
```

Відповідно до елементів форми запит містить місце, коментар, дату початку та кінця. Обробка цього запиту відбувається на сервері. Першим кроком є вилучення об’єкту даних з клієнтської сторони , який має наступний вигляд:

```
const {
      location,
      description,
      startDateTime,
      endDateTime
    }=req.body
```

Наступним кроком є встановлення особи користувача, отримання токену з авторизації:

```
oAuth2Client.setCredentials({refresh_token:storedRefreshToken})
```

Формування запиту до календаря:

```
const calendar=google.calendar('v3')
    const response=await calendar.events.insert({
      auth:oAuth2Client,
      calendarId:"primary",
      requestBody:{
        location:location,
        description: description,
        colorId:'3',
        start: {
          dateTime:new Date(startDateTime),
        },
        end: {
          dateTime:new Date(endDateTime),
        }
      }
    })
    res.send(response)
  }
```

Спрощена схема алгоритму запису подій в календар показана на рис.3.7.

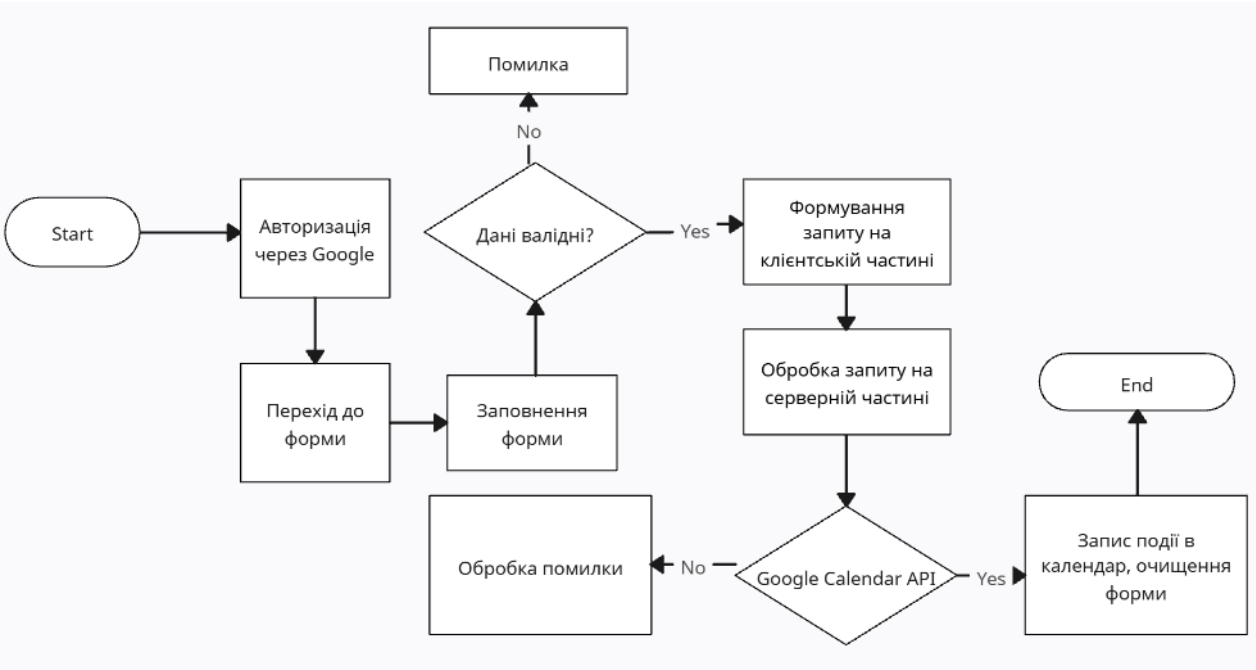

Рис.3.7. Схема алгоритму запису подій в календар

Форма для введення даних відображена на рис.3.8.

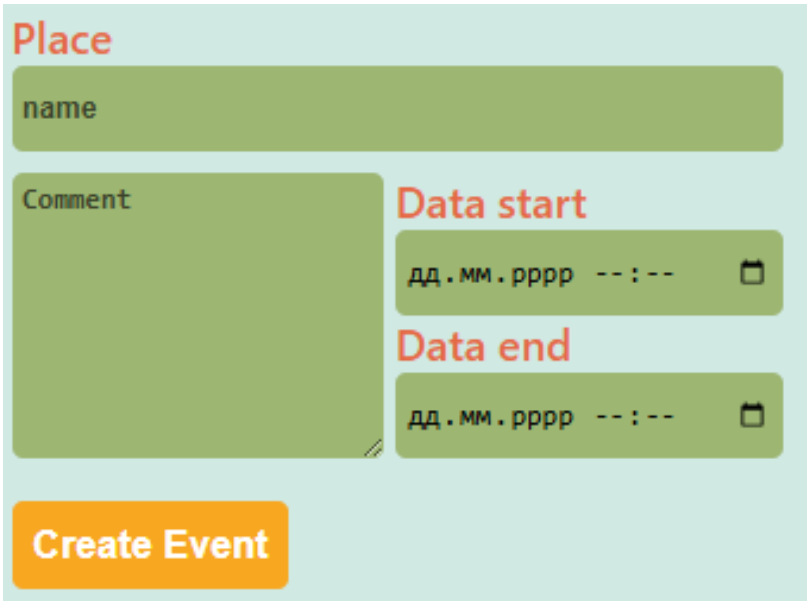

Рис.3.8. Форма для запису подій в календар

Поле для введення місця є компонентом Autocomlete для зручного пошуку місць. Введення тестових даних для перевірки алгоритму представлено на рис.3.9.

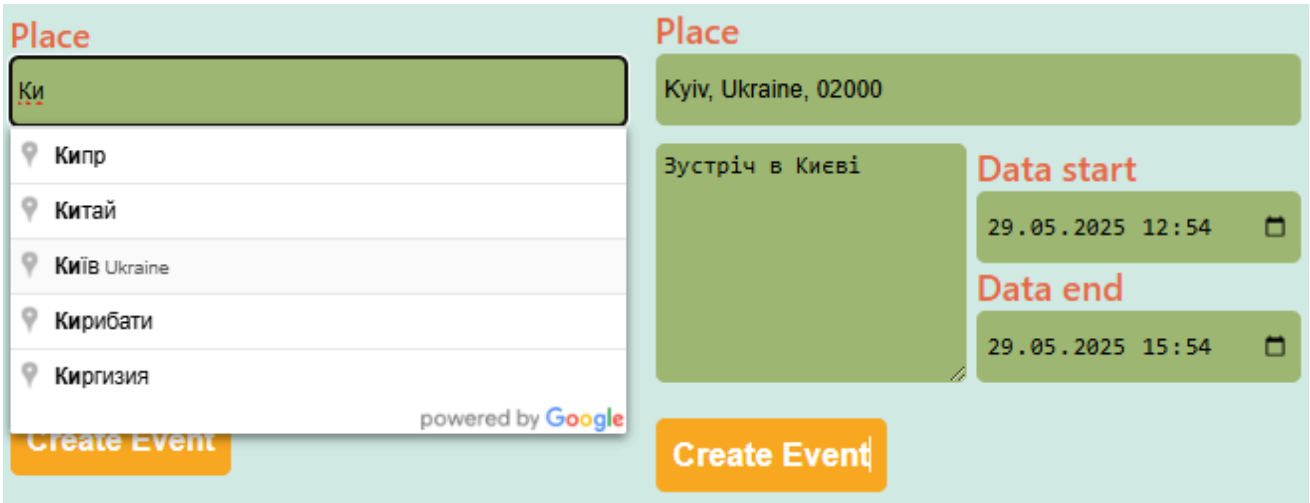

Рис.3.9. Заповнена форма для подій

Результат запису тестових даних в календар показано на рис.3.10.

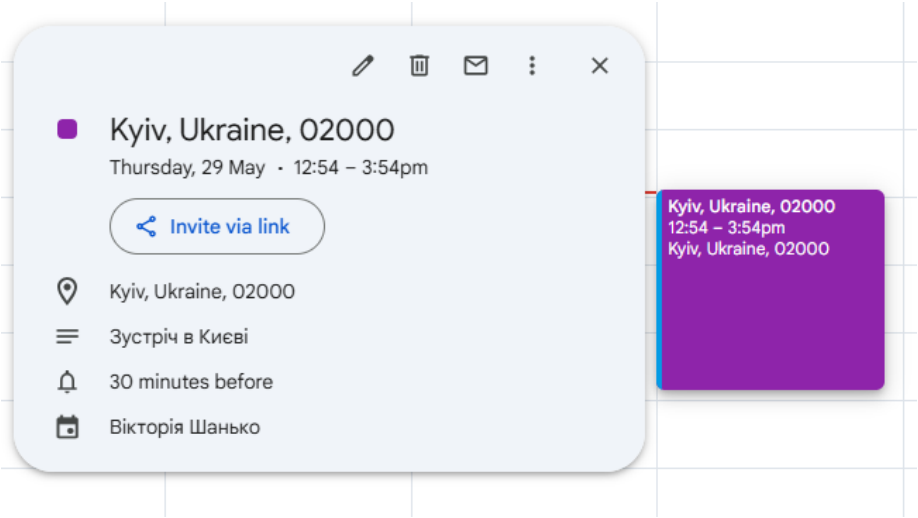

Рис.3.10. Результат тестування запису подій в календар

З огляду на тестовий приклад запису подій можна вважати, що алгоритм робочий та готовий до використання.

3.2.3. Алгоритм побудови маршруту

За визначеними вимогами в системі необхідно реалізувати побудову маршруту двома способами: за введеними даними вручну і за виставленими маркерами на мапі.

Для реалізації і першого і другого способу необхідно забезпечити відображення карти в інтерфейсі. Ця задача потребує взаємодію з Google Maps API та використання бібліотеки @react-google-maps/api.

Перед початком роботи потрібно, як в алгоритмі авторизації, зареєструвати систему в Google Cloud Console, але вже для отримання API Key. Отриманий ключ повинен містити необхідний набір API, а саме:

- Directions API – використовується, щоб обчислювати маршрути між точками, також налаштування виду транспорту.
- Places API – використовується, для отримання інформації про місця на карті.
- Geolocation API – для отримання геолокації користувача.

- Maps JavaScript API – API яке дозволяє розробникам інтегрувати елементи та функції Google Maps в додаток.
- Geocoding API – використовується для перетворення адрес на координати.

Цей набір API містить всі необхідні функції для реалізації прокладання маршрутів на карті.

Побудова маршруту за введеними вручну даними потребує реалізації декількох компонентів, окрім самої карти. Цими компонентами є: поля для введення точок, кнопка для запуску функції, кнопка для очистки маршруту. Необов'язковими елементами є поля для відображення часу та дистанції, але вони є корисними для покращенням розуміння користувачами загальної картини поїздки.

Форма для введення вхідних даних містить поля з використанням компонента «Autocomplete». Цей компонент використовується в додатку для автозаповнення введеного тексту, а саме, пошук адрес або місць на карті в базі Google Map. Програмна реалізація цього компоненту має наступний вигляд:

```
export const Autocomplete = ({isLoaded, onSelect}) => {
    const {
        ready,
        value,
        suggestions: { status, data },
        setValue,
        init,
        clearSuggestions,
    } = usePlacesAutocomplete({
        debounce: 300
    });
}
```

Ініціалізація компонента відбувається за допомогою usePlacesAutocomplete, з параметром затримки відправлення запиту, для зменшення випадкових запитів і зайвого навантаження на сервер. Параметри для автозаповнення ready – готовність компонента до використання; value – введене значення; suggestions (status,data) – набір пропозиції; що повертаються з API; setValue – оновлення значення введення; init – ініціалізація автозаповнення.

Також компонент Autocomplete має значення не тільки для зручного заповнення даних, а головною функцією є отримання координат.

```
const handleSelect =
        ({ description }) =>
            () => {
                getGeocode({ address: description })
                .then((results) => {
                    const { lat, lng } = getLatLng(results[0]);
                    onSelect({ lat, lng })
                });
            };
```

Створений об’єкт handleSelect приймає description, що є введеним текстом в поле Autocomplete. За допомогою методу getGeocode() передається запит до Google Geocoding API, який у відповідь повертає масив даних results , який містить координати місця. Для обробки даних масив застосовується метод getLatLng(), завдяки якому в результаті будуть отримані координаті місця з найбільшою точністю. Метод onSelect() передає ці дані безпосередньо в батьківський компонент. Тепер ці дані можна використовувати в інших компонентах та функціях.

Схема алгоритму роботи Autocomplete показана на рис.3.11.

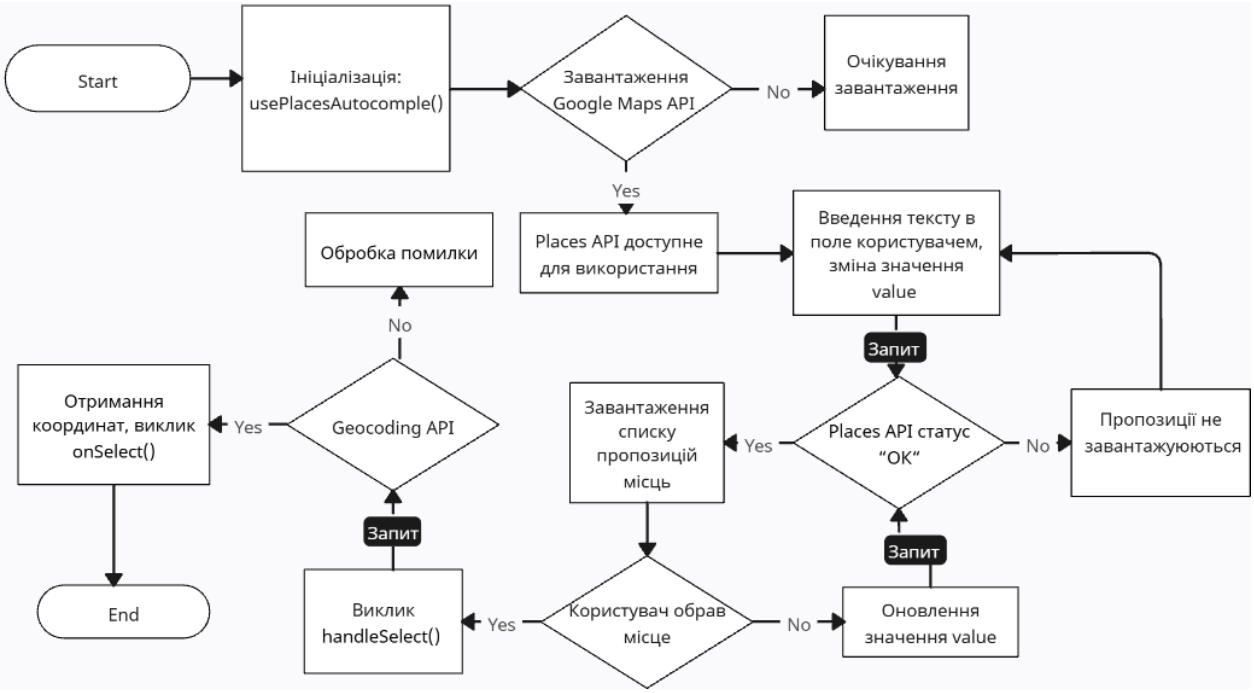

Рис.3.11. Алгоритм роботи Autocomplete

Після отримання даних з компоненту Autocomplete необхідно реалізувати функцію побудови маршруту між точками. Ключовим моментом для цієї функції є взаємодія з сервісом Google Maps Directions API. Цей сервіс розраховує маршрути між точками, також надає можливість обрати тип транспорту, розрахунок дистанції та часу. В системі має наступну програмну реалізацію:

```
async function calculateRoute() {
  if (originRef.current.value === '' || destiantionRef.current.value === '') {
    return
  }
  const directionsService = new google.maps.DirectionsService()
  const results = await directionsService.route({
    origin: originRef.current.value,
    destination: destiantionRef.current.value,
    travelMode: google.maps.TravelMode.DRIVING,
  })
  setDirectionsResponse(results)
  setDistance(results.routes[0].legs[0].distance.text)
  setDuration(results.routes[0].legs[0].duration.text)
}
```

Функція calculateRoute() це асинхронна функція для надсилання запиту до Directions API і обробки відповіді. Для уникання більшості помилок перед надсиланням запиту проводить перевірку чи заповненні поля вводу місць. Після отримання true створюється константа directionsService екземпляру сервісу API, а також константа results. В запиті використовуються наступні параметри:

- origin – початкова точка.
- destination – кінцева точка.
- travelMode – містить інформацію про тип транспорту (автомобіль).

Отримана відповідь є масивом results, він містить набір маршрутів, їх час та дистанцію. Зазвичай використовується перший маршрут, він вважається найоптимальнішим.

Схема алгоритму функції calculateRoute() показана на рис.3.12.

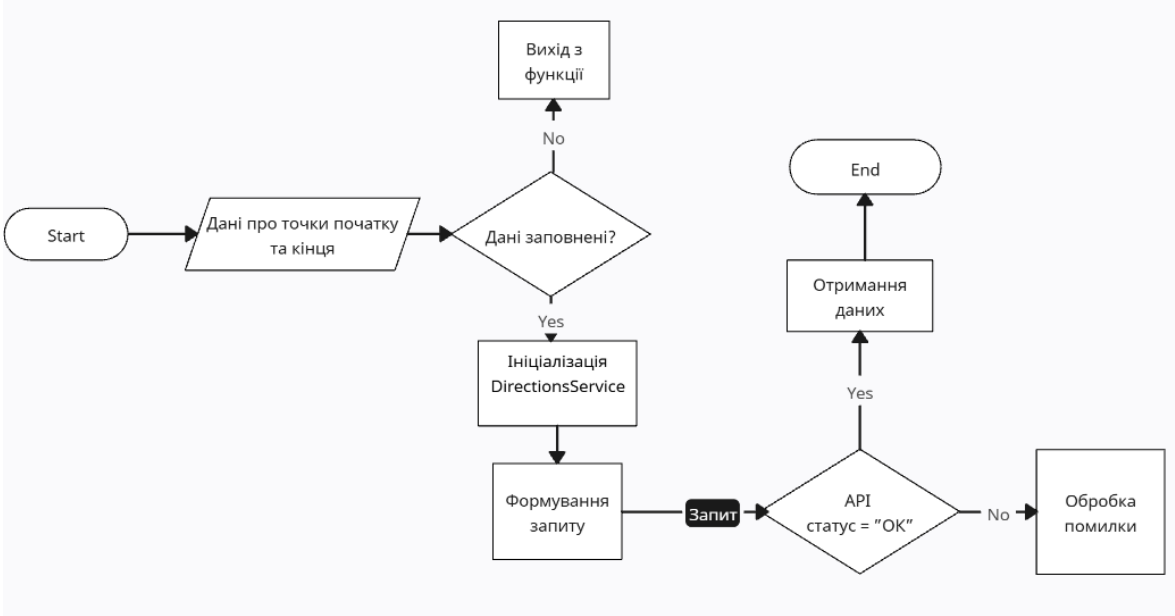

Рис.3.12. Алгоритм роботи функції побудови маршрутів

Після створення необхідних функцій та компонентів результатом використання алгоритму є відображення маршруту на карті (рис.3.13.), обчислення його часу та дистанції. Отримані дані готові до використання в інших частинах коду та зберіганні в базі даних.

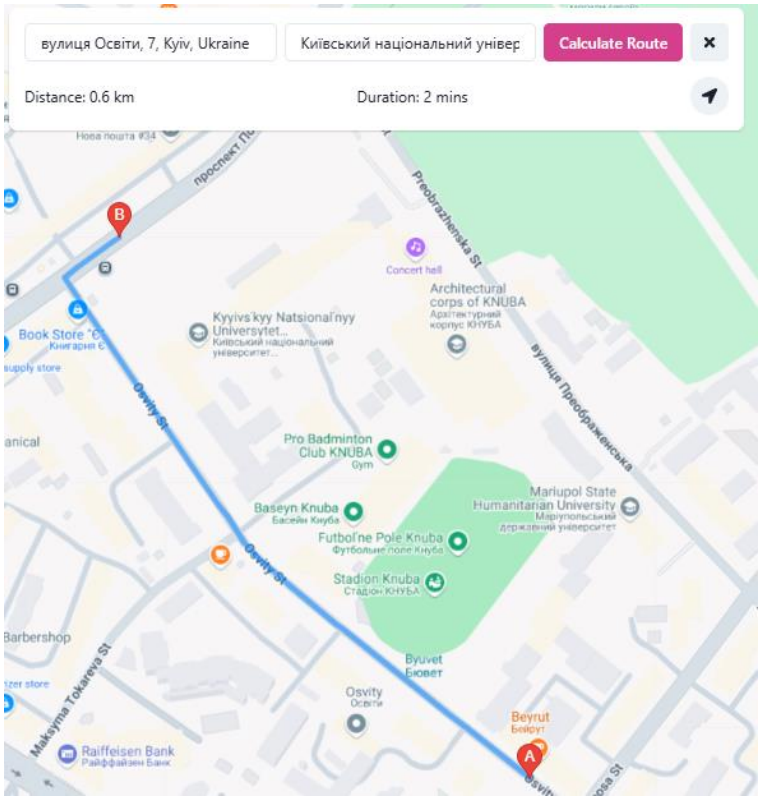

Рис.3.13. Результат тестування побудови маршруту за введеними даними

Для реалізації другого способу побудови маршруту, а саме за виставленими маркерами, до імпортованої карти в проєкт необхідно додати можливість ставити маркери і з них отримувати дані широти та довготи. Компонент карти повинен приймати наступні параметри:

- Початкові дані, зазвичай, це геолокація користувача.
- Режими взаємодії з картою.
- Маркери.
- Функція додавання маркерів.

Цей спосіб розрахунку маршруту відрізняється від попереднього тільки способом отриманні даних. Ця дія відбувається наступним чином:

1. При зміні режиму взаємодії з картою на «додавання маркерів» викликається відповідна функція, яка формує масив з координат.
2. Маркер – це об’єкт з координатами місця, який передається в компонент карти, як параметр.
3. Також необхідна обробка проміжних точок, щоб обчислювати маршрути з 3 і більше маркерами.

Підготовка проміжних точок потребує створення нового об’єкта в функції побудови маршруту

```
const waypoints = markers.slice(1, markers.length - 1).map((marker) => ({
            location: new google.maps.LatLng(marker.lat, marker.lng),
            stopover: true,
        }))
```

Об’єкт waypoints містить обробку даних. Метод markers.slice() обирає всі точки, окрім першої та останньої, таким чином це і є «зупинками» на маршруті. Метод map() перетворює ці точки на окремі об’єкти для API. Цей об’єкт передає координати (location) та вказує на те, що це саме зупинки (stopover). Потім ці проміжні точки додаються в запит до Google Maps Directions API, як окремий параметр до вже існуючих параметрів.

Блок-схема алгоритму функції побудови маршрутів показана на рис.3.14., тестовий результат використання алгоритму (рис.3.15.)

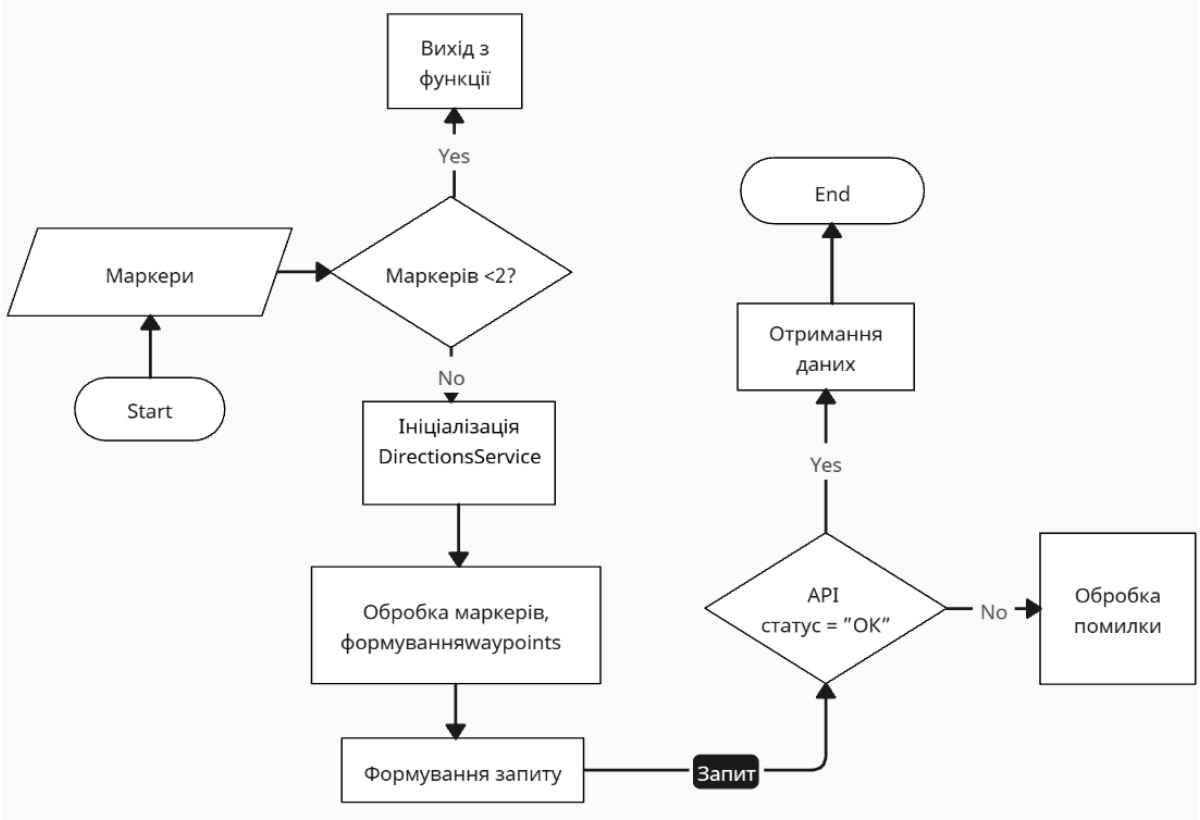

Рис.3.14. Алгоритм функції побудови маршрутів за маркерами

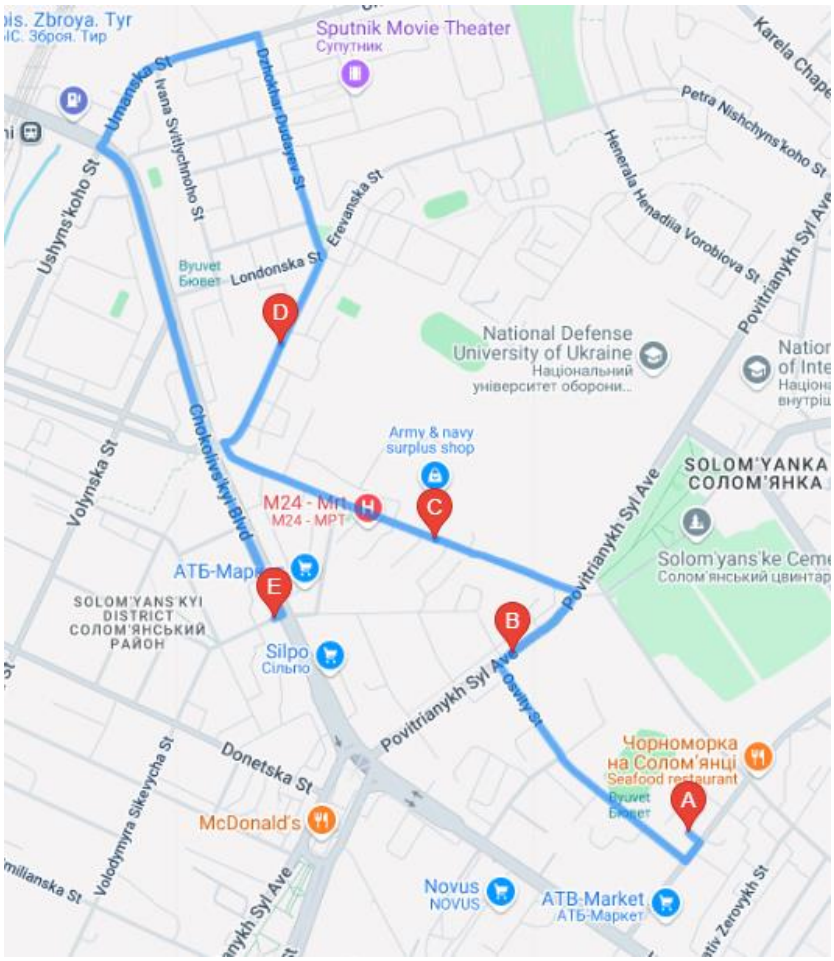

Рис.3.15. Результат тестування побудови маршруту за маркерами

Висновок до розділу 3

В розділі було визначено основні програмні рішення для реалізації веб-системи для планування подорожей. Обрано стек технологій, основними технологія стали JavaScript, React, Express, також допоміжні бібліотеки для забезпечення функцій інтегрованих сервісів. Визначено необхідний набір API, що використовуються. Розглянуті ключові алгоритми, а саме:

- Алгоритм авторизації користувачів через OAuth 2.0
- Алгоритм побудови маршруту за допомогою Google Maps API двома способами
- Алгоритм запису подій у Google Calendar.

Тестування кожного алгоритму показало їх ефективність та готовність до використання у реальних умовах. Всі основні функціональні частини системо працюють стабільно і виконують поставлену задачу, в побудові маршрутів і синхронізації з Google Calendar помилок не виявлено.

4. ВПРОВАДЖЕННЯ СИСТЕМИ

4.1. Інтерфейс програми

Розробка інтерфейсу містить кілька важливих етапів для отримання хорошого користувацького досвіду від системи.

Перший етапом розробки є аналіз вимог. Цей етап є аналізом бізнес-вимог та потреб цільової аудиторії. Такими вимогами є забезпечення зручності використання, багатомовності, функціонального планування.

Другим етапом є створення прототипу. В цей етап входить збір необхідних функцій в системі. Після збору функціоналу потрібно не забувати ураховувати принципи UI/UX. Ці принципи допомагають спроектувати правильний, приємний для користувачів інтерфейс, який буде зручним у використанні.

За зібраними вимогами можна зіставити список необхідних елементів в інтерфейсі для взаємодії з системою:

- Відображення карти. Це є головним елементом для побудови маршрутів. Карта має містити кнопки збільшення/зменшення, а також кнопку додавання маркерів.
- Відображення маршруту на карті.
- Панель навігації. Очікується, що додаток буде мати декілька компонентів.
- Форма створення подорожі. В цю форму обов’язковими полями будуть: назва поїздки, дата, кнопка додати зупинку, форма для зупинки, кнопка збереження
- Форма для зупинок. Містить в собі назву зупинки, час зупинки, коментарі до зупинки, додавання/видалення документів, опис документів. Кнопка видалення зупинки.
- Сторінки додатка. Необхідними сторінками є створення поїздки, історія поїздок, перегляд профілю.

- Історія поїздок: має містити історію всіх поїздок, історія кожної поїздки має назву подорожі, маршрут в загальному вигляді, дату початку та кінця, кнопку для розгорнутого перегляду.
- Вікно розгорнутого перегляду історії поїздки. Має містити всі дані про створену поїздку внесені користувачем.

Це загальний огляд компонентів, що необхідні для забезпечення реалізації визначеного функціоналу. Після створення прототипу можна безпосередньо переходити до дизайну.

Для демонстрації дизайну необхідно створити його макет. Для виконання цієї задачі існує багато різноманітних інструментів, таких як Adobe Photoshop, Canva, Figma, інші.

Інтерфейс додатку складається з 3 вікон. Перше – створення подорожі, друге – перегляд історії подорожей, третє – перегляд календаря. На кожному з вікон є панель навігації (header). Header виконує роль навігаційної панелі та має: логотип, перехід між сторінками та перехід на особистий акаунт користувача. Дизайн відображено на рис.4.1.

Рис.4.1. Header

З рис.4.1. можна помітити що в панелі навігації елементи різного кольору, це зроблено задля позначення активної вкладки: сіра – не активна, помаранчева – активна.

Після авторизації в системі користувач потрапляє на головний екран додатку (рис.4.2.). Цей екран є інтерфейсом для створення подорожі та має наступні основні елементи:

- Карта та кнопки взаємодії.
- Форма для заповнення і відправки даних подорожі з підформами.

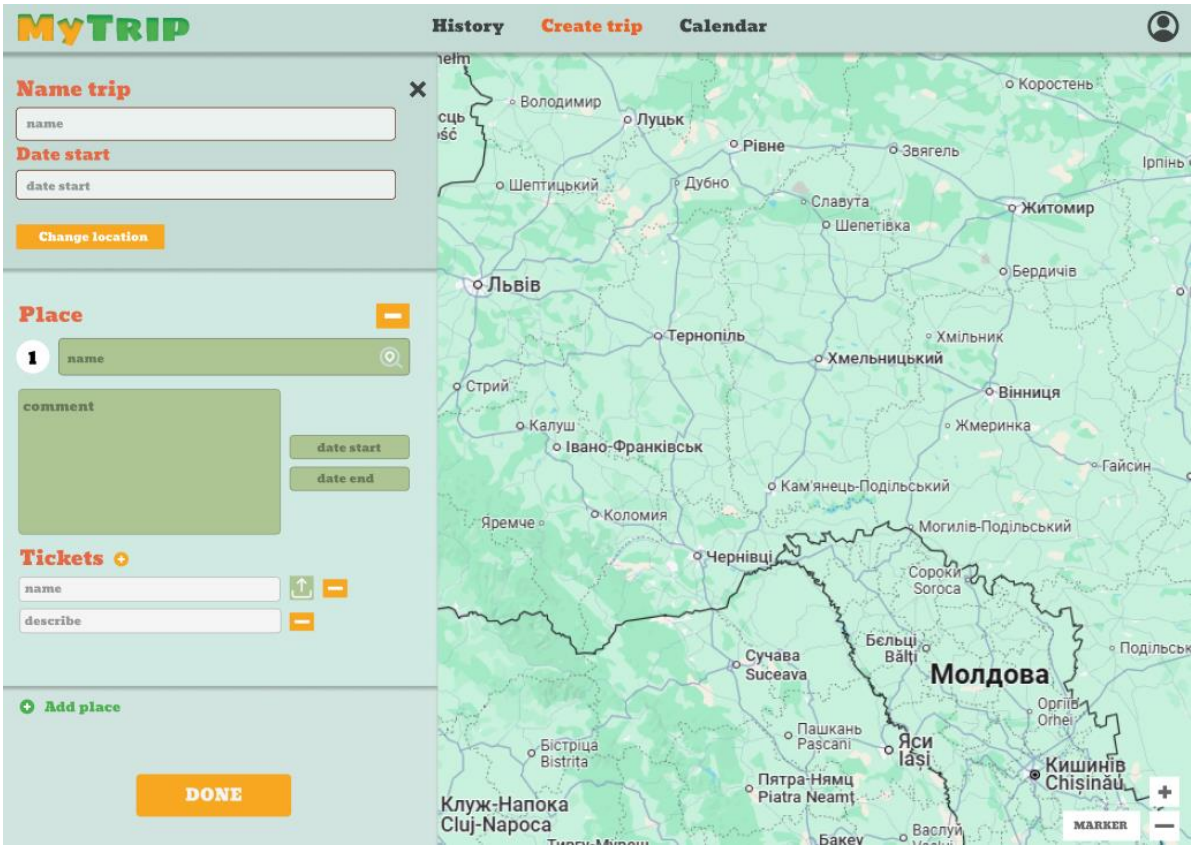

Рис.4.2. Вікно створення подорожі

Інтерфейс карти є повністю імпортованим з Google карт, тому містить всі назви місць, вулиць, країн, тощо. Компонент карти також має декілька кнопок (рис.4.3.). Кнопка «Marker» змінює стан взаємодії з картою, за вхідними налаштуваннями при натисканні на карту можна здійснювати переміщення по карті, при активації кнопки «Marker» дія натиску на карту має результат – точка на карті. При взаємодії з кнопкою її дизайн змінюється (рис.4.4.) для чіткого розуміння користувачами, що кнопка активна. Кнопки «+» та «-» відповідальні за збільшення та зменшення масштабу карти відповідно.

Рис.4.3. Кнопки компоненту карти

Рис.4.4. Дизайн активних кнопок

Форма для заповнення даних подорожі є компонентом, який містить в собі інші елементи, а саме:

- Загальні дані про подорож (рис.4.6.)
- Компонент зупинки (рис.4.7.)
- Кнопка додавання зупинок. (рис.4.5.)
- Кнопка збереження.
- Кнопка для закриття панелі.

Рис.4.5. Кнопка додавання зупинок.

Загальними даними для подорожі є її назва та дата початку. За початковими налаштуваннями першою точкою для побудови маршруту є геолокація користувача, але це можна змінити обравши іншу точку відправлення за допомогою кнопки «change location».

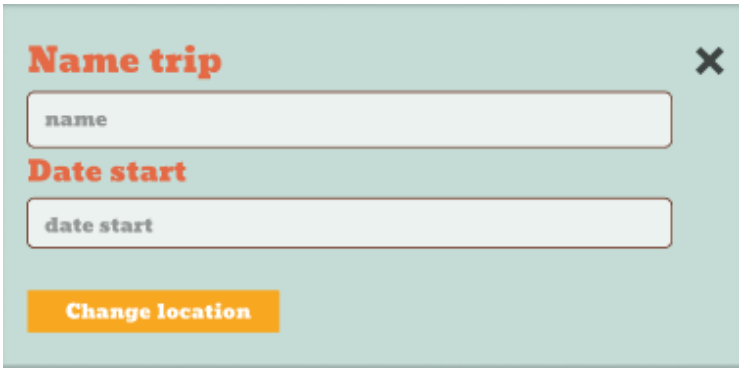

Рис.4.6. Форма загальних даних про поїздку

Наступним компонентом є компонент заповнення даних про точки зупинок. Функціонал компоненту включає в себе: відповідні поля для заповнення даних (локація, коментарі до зупинки, дати перебування), видалення зупинки, додавання або видалення білетів, додавання/видалення опису до білету. Дизайн форми відображено на рис.

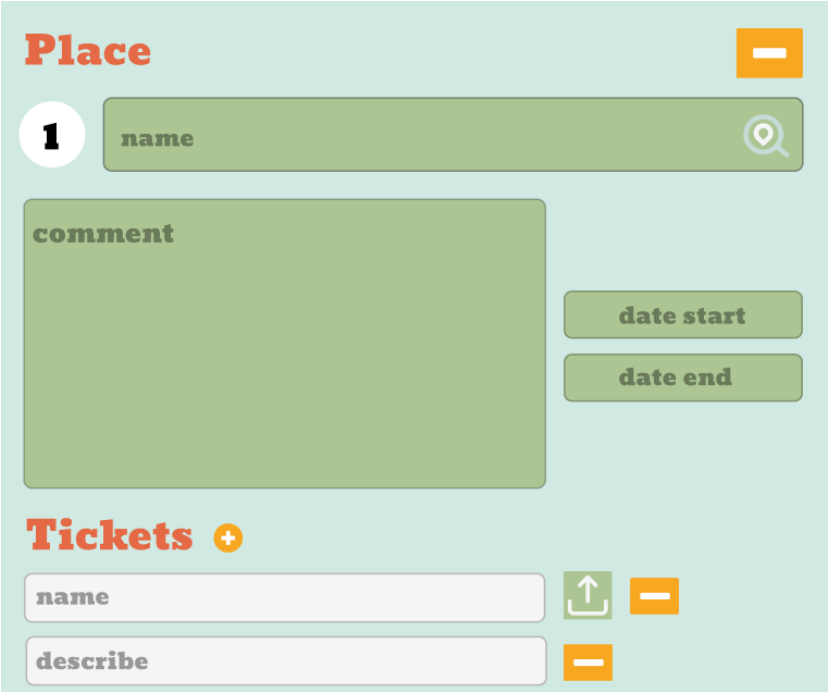

Рис.4.7. Форма даних для зупинок.

Кнопка збереження на формі має два стани відображення (рис.4.8.) помаранчевий фон – кнопка не використовувалась, зелений – дані успішно збережено. При натисканні на кнопку будується маршрут на карті з обраних зупинок, а всі дані про подорож зберігаються в історію подорожей і базу даних автоматично.

Рис.4.8. Стани кнопки збереження поїздки

На кожну додану зупинку при створені маршруту за введеними даними користувача створюється окрема подія в календарі на обрану дату та час з вказаними локаціями та коментарями.

Дизайн із заповненими даними вкладки «Create trip» проілюстровано на рис.4.9. Так виглядає вікно створення подорожі після опрацювання всіх введених даних.

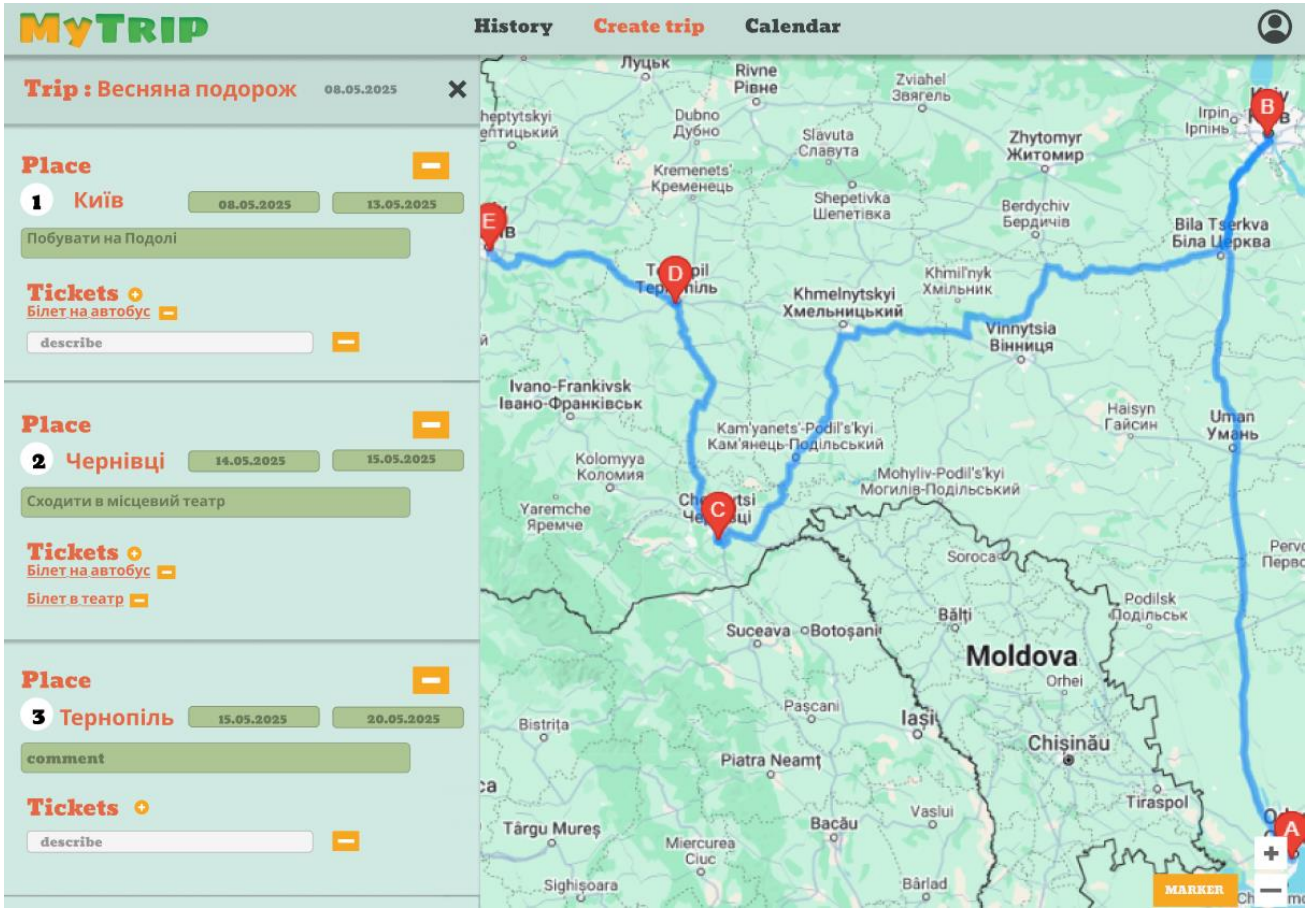

Рис.4.9. Вікно створення подорожі після обробки даних

Якщо після заповнення даних користувачу необхідно внести зміни при взаємодії з панеллю даних або картою стан кнопки «done» повертається до «не використана», що дозволяє зберегти дані та побудувати маршрут знову.

Вкладка «History» відображає список створених поїздок, кожен запис є окремим компонентом, який можна «розгорнути» та отримати всі подробиці про обрану подорож. Дизайн вікна показано на рис.4.10., дані є демонстраційними.

Рис.4.10. Вікно історії подорожей

Пропонується розглянути компонент кожної поїздки більш детально на рис.4.11. Кожна запис містить назву поїздки, назви локацій на маршруті, дати початку та кінця та кнопку «Details». При використанні кнопки «Details» відображається повна історія поїздки (рис.4.12.).

При використанні вкладки «Calendar» відкривається інтерфейс Google Calendar.

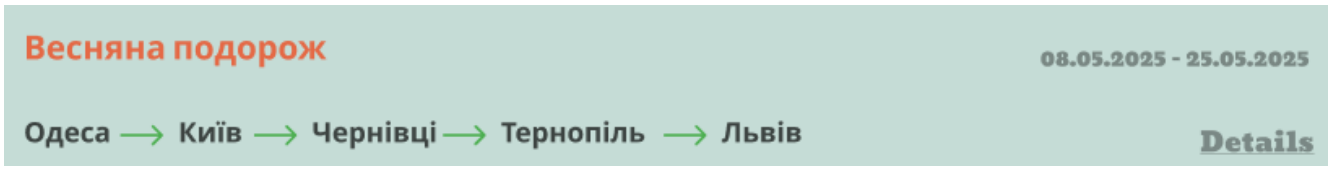

Рис.4.11 Компонент загальної історії подорожі

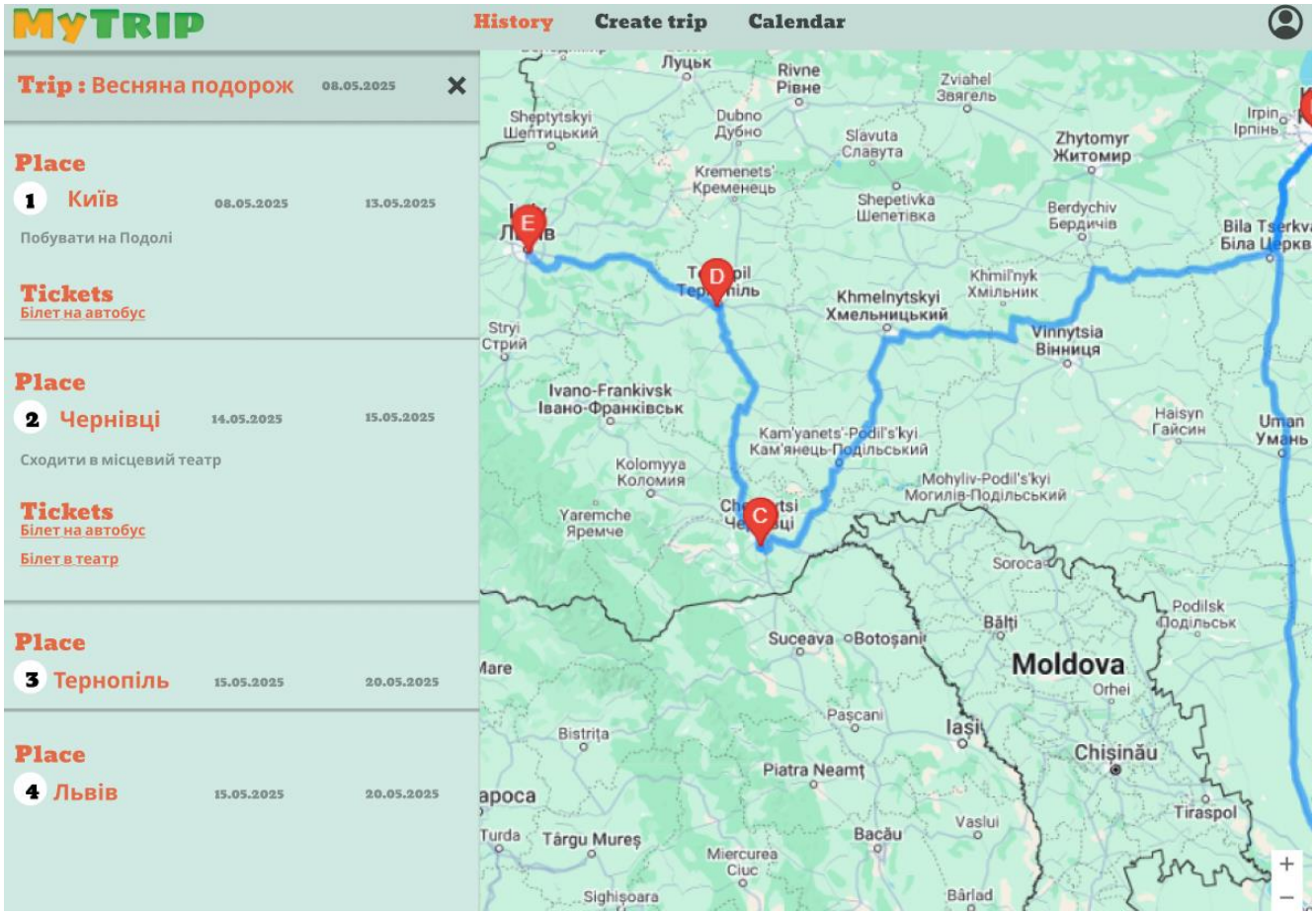

Рис.4.12. Повна історія поїздки

Задля забезпечення гарного користувацького досвіду при програмній реалізації необхідно реалізувати адаптивність сторінок, врахувати різні розширення пристроїв. Вище наведені макети розраховані на десктопну версію додатку. Також необхідно реалізувати багатомовність.

За даним дизайном користувачу потрібно виконати мінімум дій для отримання маршруту, а з використанням автозаповнення локацій ця задача ще спрощується.

Інтерфейси стандартні та зрозумілі, без зайвих модифікацій та не відволікають від основної мети користувача, обрана кольорова гама має спокійні поєднувані між собою кольори.

4.2. Розрахунок вартості розміщення системи

Система потребує розміщення як веб-додаток для загального користування у хмарних сервісах. Це спрощує використання системи як розробникам так і користувачам, з очевидних плюсів:

- Безперервна робота системи.
- Легке масштабування.
- Інтеграція системи з іншими сервісами.
- Обробка даних на віддалених серверах.

На разі існує багато хмарних сервісів, але найпопулярніші з них це: Amazon Web Services, Google Cloud Platform, Microsoft Azure. Ці сервіси надають різні послуги для розробників, мають різний бюджет та різні потужності використання.

З точки зору бюджету систему можна розміщувати на кожному з цих сервісів в залежності від кількості ресурсів. Порівняння вартості послуг розміщення системи за місяць на кожному з вище згаданих сервісів описана в табл.4.1.

Таблиця 4.1.

Порівняння цін в хмарних сервісах

Послуга / платформа	Веб-хостинг	Зберігання файлів (1 ГБ)	Реєстрація домену / рік
AWS	$8.47	$0.023	від $12
Google Cloud Platform	$4.69	$0.02	від $12
Microsoft Azure	$13.39	$0.0208	Не надає прямої реєстрації домену

Ціни в табл.4.1. вказані за найменшим тарифом для невеликих систем. Порівнявши вартість послуг платформ можна зробити висновок, що з фінансової точки зору найвигідніше буде розміщувати систему в Google Cloud Platform. Але, якщо не враховувати бюджет, а враховувати функції системи вибір може бути

інакший. Однак, так як в системі використовується інтеграція з сервісами Google то і з цієї точки зору найкраще буде розміщувати систему в Google Cloud Platform.

Сервіс Google Cloud Platform зручний не тільки для розміщення самої системи, а і бази даних. Базу даних також краще не зберігати на локальному пристрої, тому що:

- Локальний пристрій має обмежену потужність, тому скоріш за все не зможе обробляти велику кількість запитів і даних.
- Хмарні сервіси надають більшу захищеність.
- Резервне копіювання даних потрібно робити вручну.
- Підтримка локального пристрою буде нести більше витрат, аніж зберігання в хмарі.
- Обмежений доступ до бази, тільки через локальну мережу.

Google Cloud Platform надає можливість підтримувати різні типи баз даних, але для розробленої системи найкращим вибором є сервіс Cloud SQL. Cloud SQL це керований сервіс для таких баз даних як MySQL, PostgreSQL. Сервіс містить різні тарифи та екземпляри в залежності від цілей використання, має автоматичне масштабування та високу продуктивність. Порівняння екземплярів Cloud SQL для PostgreSQL наведено в табл.4.2.

Таблиця 4.2.

Порівняння екземплярів Cloud SQL

	Опис	Кількість процесорів (шт.)	Оперативна пам’ять (ГБ)	SSD (ГБ)	Можливе відновлення бази даних
SandBox	Використовується для тестування	2	8	10	7 днів
Development	Можна використовувати для реальних продуктів, але має низьку доступність	4	32	250	35 днів
Production	Оптимізовано для великих систем	8	64	250	35 днів

Cloud SQL надає можливість самостійно змінювати конфігурацію даних екземплярів, наприклад, кількість процесорів (до 128 шт.) та кількість оперативної пам’яті (до 864 ГБ). Але так як спроектована база даних на даному етапі не передбачає занадто великого обсягу даних та складних розрахунків то екземпляру Development цілком буде достатньо для роботи. З подальшими оновленнями продукту конфігурацію та екземпляр можна змінити, враховуючи потрібне навантаження та кількість ресурсів.

Розрахунок вартості підтримки бази даних в регіоні europe-west1(Бельгія) та кількома зонами доступності наведена в табл.4.3.

Таблиця 4.3.

Вартість підтримки бази даних з кількома зонами

Елемент	Вартість/годину
4 віртуальних процесора	$0,21
32 ГБ оперативної пам’яті	$0,29
SSD-накопичувач 250 ГБ	$0,06
Кешування даних 375 ГБ	$0,08
Резервна віртуальна машина	$0,65
Загально	$1,29

Вартість в табл. 4.3. наведена за годину роботу. Якщо розраховувати вартість за місяць, то сума буде складати:

$$\begin{cases} 24\ \text{години} \times 30\ \text{днів} \times \$1{,}29 \approx \$928{,}8 \\ 24\ \text{години} \times 31\ \text{днів} \times \$1{,}29 \approx \$959{,}76 \\ 24\ \text{години} \times 28\ \text{днів} \times \$1{,}29 \approx \$866{,}88 \\ 24\ \text{години} \times 29\ \text{днів} \times \$1{,}29 \approx \$897{,}84 \end{cases}$$

Сума залежить від місяця та року, але зрозуміло, що для початкового етапу впровадження додатку вона занадто велика. Зменшити вартість можна змінивши регіон розташування бази даних та обрати одну, а не кілька зон. Але така зміна зменшить доступність бази даних та сповільнить швидкість роботи з нею. Одна зона доступності значно зменшить витрати на підтримку бази даних, тому що не

передбачає використання резервної віртуальної машини. Розрахунок вартості з використанням однієї зони наведено в табл.4.4.

Таблиця 4.4.

Вартість підтримки бази даних з однією зоною

Елемент	Вартість/годину
4 віртуальних процесора	$0,21
32 ГБ оперативної пам’яті	$0,29
SSD-накопичувач 250 ГБ	$0,06
Кешування даних 375 ГБ	$0,08
Загально	$0,65

Тоді місячна вартість становить:

$$\begin{cases} 24 \text{ години} \times 30 \text{ днів} \times \$0{,}65 \approx \$468 \\ 24 \text{ години} \times 31 \text{ днів} \times \$0{,}65 \approx \$483{,}6 \\ 24 \text{ години} \times 28 \text{ днів} \times \$0{,}65 \approx \$436{,}8 \\ 24 \text{ години} \times 29 \text{ днів} \times \$0{,}65 \approx \$452{,}4 \end{cases}$$

Таким чином вартість обслуговування зменшилась, але використовуючи цей варіант потрібно враховувати важливий ризик: при технічних проблемах в даній зоні база даних може бути недоступному деякий час.

Якщо стартовий капітал проєкту передбачає зовсім невеликий бюджет можна змінити тип екземпляру на SandBox, база буде розташована в europe-west1(Бельгія) з однією зоною доступності. SandBox дозволить підтримувати роботу системи з малою кількістю користувачів. Розрахунок вартості підтримки бази даних наведена в табл.4.5

Таблиця 4.5.

Вартість підтримки бази даних SandBox

Елемент	Вартість/годину
2 віртуальних процесора	$0,08
8 ГБ оперативної пам’яті	$0,06
SSD-накопичувач 10 ГБ	$0,002
Загально	$0,142

Окрім, відсутності резервної віртуальної машини в цьому варіанті також відсутній кешування даних, що є очевидним мінусом. За використання SandBox з такою конфігурацією місячна вартість буде наступною:

$$\begin{cases} 24 \text{ години} \times 30 \text{ днів} \times \$0{,}14 \approx \$100{,}8 \\ 24 \text{ години} \times 31 \text{ днів} \times \$0{,}14 \approx \$104{,}16 \\ 24 \text{ години} \times 28 \text{ днів} \times \$0{,}14 \approx \$94{,}08 \\ 24 \text{ години} \times 29 \text{ днів} \times \$0{,}14 \approx \$97{,}44 \end{cases}$$

Цей варіант керування базою даних є значно дешевшим, ніж два попередніх, але не рекомендується до використання в системі, що розглядається.

Звичайно, наведено порівняння цін недостатньо для обрахування вартості обслуговування системи, тому що потрібно ще врахувати, як мінімум, кількість запитів до домену, а також кількість запитів до цього домену, кількість запитів до API.

Обрахування вартості підтримки API залежить від кількості інтегрованих API та кількості запитів до нього. Тарифний план API, що використовуються в системі, наведено в табл.4.6.

Таблиця 4.6.

Тарифний план API

Назва	Кількість безкоштовних запитів	Вартість 1000 платних запитів
Directions API	2500/день	$5
Places API	2500/день	$2.83
Geolocation API	50000/день	$5
Maps JavaScript API	28000/місяць	$7
Geocoding API	40000/місяць	$5
Google Calendar API	1000000/день	$0.006

На початкових етапах роботи безкоштовного ліміту запитів буде вистачати для функціонування системи. Для обрахування витрат на API потрібен глибокий

аналіз, дані про кількість активних користувачів, статистику використання додатка.

Для більш точного складання фінансового плану потрібно розуміти кількість та тип доменів на який буде розміщена система. У даному випадку найбільш підходящими типами доменів є .com та .travel. Домен .com стандартизований та має широку підтримку, підходить для великої кількості аудиторії. Домен .travel спеціально призначений для туристичної індустрії, є менш впізнаваним, але широко використовується для залучення певної цільової аудиторії.

Домени можна реєструвати через різні сервіси, це не обов'язкову обраний хмарний сервіс, але такий підхід містить певні мінуси:

- Додаткові кроки: окрім реєстрації домена на сторонньому сервісі потрібно також окрема налаштувати Google Cloud DNS. Це зайві дії, так як набагато зручніше управління доменом через одну платформу.
- Додаткові витрати: для керування стороннім доменом за допомогою Google Cloud DNS стягується окрема плата.
- Незручне керування: у випадку створення домену не в сервісі Google Cloud керування цим доменом здійснюється на окремому інтерфейсі вибраного сервісу, а керування DNS – через Google Cloud Console.
- Безпека: обраний сервіс повинен мати достатній рівень захисту даних.

Плюси реєстрації домену на сторонньому сервісі:

- Можливість більшого вибору типів домену. Google Cloud підтримує домени тільки верхнього рівня, такі як .com, .org, .net, .co, .dev, інші.
- Можливість обирати цінову політику.

Дізнавшись про плюси і мінуси реєстрації домену на сторонньому сервісі можна зробити висновок, що кращим рішенням буде реєстрація домену через Google Cloud. Це забезпечить зручне управління доменом та його захищеність. Оскільки GCP підтримує лише поширені домени, то для системи оптимальним рішенням був би домен .com.

Отже, підсумувавши вище сказані аспекти можна скласти таблицю вартості розміщення й підтримки системи на місяць використання на ранніх етапах просування. Розрахунок вартості наведений в табл.4.7.

Таблиця 4.7.

Вартість впровадження та підтримки системи

Послуга	Вартість
Веб-хостинг	$4.69
Розміщення та підтримка бази даних	В середньому $452,4
Зберігання даних (1 ГБ)	$0.02
Реєстрація домену	$12
Підтримка API	Безкоштовно до використання лімітів
DNS	$0.20 за кожну DNS-зону, запити безкоштовні до використання лімітів (1000000 на місяць)
Загальна сума на місяць	Від $470

В табл.4.7. обрахована мінімальна сума впровадження та підтримки системи в хмарному сервісі Google Cloud Platform на місяць з рекомендованим розміщенням бази даних. В цю суму не входить використання лімітів API та DNS.

4.3. Стратегія просування

Стратегія просування є комплексним інструментом для будь-якого бізнесу, включаючи IT-продукти. Це план дій спрямований на просування продукту з метою залучення нових користувачів.

Першою необхідною складовою плану є аналіз цільової аудиторії. Основною аудиторією продукту будуть мандрівники, які активно використовують технологію задля планування своїх поїздок. Другорядна аудиторія це бізнеси, які займаються організацією подорожей.

Унікальні пропозиції продукту для користувачів:

- Інтеграція з Google сервісами для зручного планування.

- Персоналізовані маршрути: забезпечення можливості створення індивідуальних маршрутів з урахуванням вподобань користувача.
- Офлайн-доступ: з подальшими оновленнями продукту забезпечити офлайн-доступ до додатку.
- Багатомовність: забезпечення підтримки різних мов інтерфейсу.

Після аналізу цільової аудиторії та визначення сильних сторін продукту потрібно визначити напрямки просування. Основні рішення просування в інтернеті:

- Ведення соціальних мереж: Facebook та Instargam слугують для запуску таргетованої реклами, створення корисного контенту (огляди оновлень в додатку, поширення цікавих маршрутів). YouTube та TikTok для відео-оглядів, інструкцій з використання, демонстрації можливостей функціоналу.
- Пошукова оптимізація SEO: за допомогою правильно побудованого SEO веб-сайт буде добре взаємодіяти з пошуковими системами, тому буде більш доступним для користувачів. Покращить трафік залученості користувачів без витрат на рекламу.
- Реклама в пошукових система, наприклад, реклама в Google та на партнерських сайтах.
- Розсилка новин на пошту користувачів.
- Співпраця з блогерами та інфлюєнсерами: наразі блогери є лідерами думок серед молодого покоління, тому необхідно щоб про продукт дізнавались не тільки з таргетованої реклами, а і від реальних людей.
- Пропозиції знижок, винагороди за привернення нових користувачів по промокодам.

Просування не в Інтернет-просторі задля демонстрації продукту:

- Участь у туристичних виставках
- Участь в конференціях

- Розповсюдження брошур у місцях популярних серед туристів, це можуть бути вокзали, аеропорти, та інше.
- Організація заходів від компанії. Створення подій, де окрім демонстрації продукту для гостей будуть різні бонуси.

Такими є основні аспекти залучення нових користувачів. Для всіх заходів, таргетованої реклами, ведення сторінок в соціальних мережах необхідно зіставити контент-план. В контент-плані повинні бути прописані всі статті, відео, програми, задля успішного просування.

Також стратегія потребує постійної аналітики. Необхідно відслідковувати вподобання користувачів, статистику соціальних мереж, які фото та відео більше цікавлять аудиторію. Прораховувати слабкі місця та змінювати старі формати подачі інформації на нові в разі низьких охоплень.

4.4. Оцінка ризиків

При просуванні продукту в маси загального користування важливо розуміти ряд факторів, які можуть вплинути на якість продукту. Ці ризики необхідно вміти усувати або намагатись частково знизити їх вплив.

Найбільш розповсюдженою категорією ризиків є «технічні ризики»:

- Проблеми з хостингом. Інколи виникають проблеми з впровадженням системи, це має значний вплив, повністю перешкоджає роботі системи тому потребує негайного вирішення, а саме: обрати надійний хостинг для уникнення хакерських атак, перевірка кількості використаних запитів.
- Помилки в коді. Часта проблема з високим впливом, зниженню кількості помилок в коді сприяє використання код-рев’ю
- Проблеми з продуктивністю. Впливає на швидкість завантаження додатка, рішенням є оптимізація великих файлів.

- Проблеми з безпекою даних. Потребує негайного вирішення при виникненні, для уникнення потрібно використовувати надійні сервіси, алгоритми шифрування.

Фінансові ризики:

- Неправильний розподіл бюджету. Уникненням цього ризику є створення детальних планів просування, обрахування затрат на розробку, впровадження, збирання статистики.
- Перевищення бюджету. Навіть з детально складеними планами бувають ситуації коли в системі потрібні зміни, які потребують додаткових витрат.

Також є категорії ризиків на які складно уникнути без додаткового використання бюджету. Такими ризиками є ринкові та юридичні. Часто ринок змінюється, користувачі потребують впровадження нових функцій або ж на ринку з’явився конкурентний продукт і потрібно встигати за новими тенденціями. Юридичні ризики також непередбачувані так як законодавство змінюється.

Не менш важливими є ризики спричинені людським фактором, це можуть бути помилки в тестуванні або зміни у вимогах під час розробки продукту. Але кожна категорія ризиків веде до можливих фінансових труднощів для компанії.

Висновки до розділу 4

У розділі було проведене моделювання інтерфейсу згідно принципів ергономіки і зібраним вимогам. Визначено оптимальні варіанти для розміщення системи на хмарних платформах, зокрема Google Cloud Platform. Оцінено ризики та переваги використання хмарних сервісів, а також розглянуто аспекти безпеки та захисту даних, що мають критичне значення для стабільної роботи системи.

ВИСНОВКИ

У результаті виконання дипломної роботи була розроблена веб-система для планування подорожей, яка інтегрується з сервісами Google Maps та Google Calendar. Основною проблемою, яка стала предметом дослідження, є відсутність універсальної веб-системи, що дозволяє мандрівникам ефективно планувати свої подорожі, використовуючи можливості сучасних онлайн-сервісів. У роботі було поставлено мету створити веб-систему, що об'єднує функціональність популярних інструментів у зручному інтерфейсі, який враховує сучасні потреби користувачів.

Для досягнення мети виконання завдання був проведений аналіз предметної області. Аналіз існуючих додатків виявив переваги та недоліки кожного застосунку з розглянутих. На основі цих недоліків було визначено головні вимоги до системи та необхідний функціонал. Також обрано тип веб-системи, а саме веб-додаток виду MPA.

Реалізовано алгоритм безпечної авторизації користувачів за допомогою стандарту OAuth 2.0, алгоритм додавання подій в Google Calendar, алгоритми побудови маршруту за введеними даними та маркерами на карті. Тестування алгоритмів підтвердило їх готовність до застосування в реальних умовах.

Веб-система побудована з використанням Google Maps API для візуалізації маршрутів і Google Calendar API для створення подій. Для обробки даних використовуються сучасні серверні технології, база даних та інтерактивний інтерфейс, розроблений відповідно до принципів ергономіки — зокрема, адаптивності, зручності навігації та підтримки різних мов. Архітектура системи клієн-сервер та структурована на підсистеми, що спрощує її масштабування та подальший розвиток.

Переваги розробленої системи:

- Інтеграція з Google Maps та Google Calendar: поїздки синхронізуються з подіями календаря, що значно спрощує планування.
- Гнучка архітектура: модульна структура дозволяє легке масштабування та вдосконалення функціональності.

- Безпека: розміщення системи в захищеному хмарному сервісі, а також дотримання стандартів безпечної авторизації користувачів.
- Зручний інтерфейс: інтуїтивно зрозумілий адаптивний інтерфейс дозволяє легке користування з будь-якого пристрою.

Виявлені можливі ризики:

- Залежність від сторонніх сервісів: неможливість розвитку системи як самостійного елементу.
- Збільшення витрат: у разі збільшення користувачів треба оптимізація баз даних та серверних ресурсів.
- Залежність від інтернет-з’єднання.

Розроблена веб-система для планування подорожей є ефективним інструментом для користувачів, що дозволяє зручно прокладати маршрути, зберігати події та отримувати доступ до інформації про подорожі. Усі основні функціональні вимоги були успішно реалізовані, а тестування показало стабільну роботу системи. Незважаючи на переваги, система має деякі ризики, пов'язані з залежністю від сторонніх сервісів і безпекою даних, які необхідно враховувати при подальшій експлуатації і розвитку системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Tripit Navigator [Електронний ресурс] // Help. Tripit. [сайт]. – URL: https://help.tripit.com/en/support/solutions/articles/103000063354-navigator (дата звернення: 11.02.2025)
2. Tripit Mobile Calendar [Електронний ресурс] // Help. Tripit. [сайт]. – URL: https://help.tripit.com/en/support/solutions/articles/103000063344-tripit-mobile-calendar (дата звернення: 11.02.2025)
3. Інтеграція з Apple Watch Tripit [Електронний ресурс] // Help. Tripit. [сайт]. – URL: https://help.tripit.com/en/support/solutions/articles/103000063424-apple-watch (дата звернення: 11.02.2025)
4. Рейтинг Tripit - Travel Planner [Електронний ресурс] // Google Play. [сайт]. – URL: https://play.google.com/store/apps/details?hl=en_US&id=com.tripit (дата звернення: 11.02.2025)
5. Рейтинг Tripit - Travel Planner [Електронний ресурс] // App Store. [сайт]. – URL: https://apps.apple.com/us/app/tripit-travel-planner/id311035142 (дата звернення: 11.02.2025)
6. Tripit Web Pro [Електронний ресурс] // Tripit. [сайт]. – URL: https://www.tripit.com/web/pro (дата звернення: 11.02.2025)
7. Відгуки про додаток Tripit [Електронний ресурс] // Going.com. [сайт]. – URL: https://www.going.com/guides/tripit-review (дата звернення: 11.02.2025)
8. Про додаток Roadtrippers [Електронний ресурс] // Wikipedia. [сайт]. – URL: https://en.wikipedia.org/wiki/Roadtrippers (дата звернення: 11.02.2025)
9. Рейтинг Roadtrippers Trip Planner [Електронний ресурс] // App Store. [сайт]. – URL: https://apps.apple.com/us/app/roadtrippers-trip-planner/id944060491 (дата звернення: 11.02.2025)
10. Рейтинг Roadtrippers [Електронний ресурс] // Google Play. [сайт]. – URL: https://play.google.com/store/apps/details?id=com.roadtrippers (дата звернення: 11.02.2025)

11. TripAdvisor – офіційний сайт [Електронний ресурс] // TripAdvisor. [сайт]. – URL: https://www.tripadvisor.com/ (дата звернення: 11.02.2025)
12. Про додаток TripAdvisor [Електронний ресурс] // Wikipedia. [сайт]. – URL: https://en.wikipedia.org/wiki/Tripadvisor (дата звернення: 11.02.2025)
13. Документація React – офіційний сайт [Електронний ресурс] // React Dev. [сайт]. – URL: https://react.dev/ (дата звернення: 12.02.2025)
14. Історія заснування Google Maps [Електронний ресурс] // Medium. [сайт]. – URL:https://medium.com/@lewgus/the-untold-story-about-the-founding-of-google-maps-e4a5430aec92 (дата звернення: 12.02.2025)
15. Карти Google [Електронний ресурс] // Wikipedia. [сайт]. – URL: https://uk.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D1%82%D0%B8_Google (дата звернення: 12.02.2025)
16. Стаття з оглядом алгоритмів побудови маршрутів Google Maps. International Journal of Computer Applications [Електронний ресурс] // Academia. [сайт]. – URL: https://www.academia.edu/56111037/Google_Maps (дата звернення: 13.02.2025)
17. Бабич, В., Костенко, А., Плеша, В., Плеша М., Хмілярчук, Л. (2023). Задача пошуку найкоротшого шляху: порівняльний аналіз основних алгоритмів [Електронний ресурс] // Politehnica.dp.ua. – URL: https://journals.politehnica.dp.ua/index.php/it/article/download/306/274/482 (дата звернення: 13.01.2025)
18. Принципи роботи алгоритму Дейкстри JavaRush Python [Електронний ресурс] // JavaRush. [сайт]. – URL: https://javarush.com/ua/quests/lectures/ua.javarush.python.core.lecture.level17.lecture08 (дата звернення: 13.02.2025)
19. Туристичний погляд 2023: тенденції та прогнози [Електронний ресурс] // НВ Бізнес. [сайт]. – URL:https://biz.nv.ua/ukr/markets/turistichniy-poglyad-2023-tendenciji-ta-prognozi-50391247.html (дата звернення: 14.02.2025)

20. Підсумки літнього сезону 2023 туроператор Join UP [Електронний ресурс] // Wing.com.ua. [сайт]. – URL: https://www.wing.com.ua/content/view/34233/81/ (дата звернення: 14.02.2025)
21. Показники міжнародного туризму UN Tourism Tracker [Електронний ресурс] // UNWTO. [сайт]. – URL: https://www.unwto.org/tourism-data/un-tourism-tracker (дата звернення: 14.02.2025)
22. Статистика використання цифрових технологій мандрівниками Consumer Trends [Електронний ресурс] // Phocuswright. [сайт]. – URL: https://www.phocuswright.com/Travel-Research/ConsumerTrends (дата звернення: 14.02.2025)
23. Туризм в Україні: новітні підходи [Електронний ресурс] // Donnu.edu.ua. [сайт]. – URL: https://jktod.donnu.edu.ua/article/view/16287/16182 (дата звернення: 14.02.2025)
24. Веб-додатки: чим сайт відрізняється від веб-додатку [Електронний ресурс] // Outsourcing.team. [сайт]. – URL: https://outsourcing.team/ua/blog/development/shho-take-veb-dodatok-chim-sajt-vidriznyayetsya-vid-veb-dodatku/ (дата звернення: 15.02.2025)
25. Як створити веб-додаток [Електронний ресурс] // Wezom. [сайт]. – URL: https://wezom.com.ua/ua/blog/kak-sozdat-veb-prilozhenie (дата звернення: 15.02.2025)
26. The Agile Journey: A Scrum Overview [Електронний ресурс] // PM Partners. [сайт]. – URL: https://www.pm-partners.com.au/insights/the-agile-journey-a-scrum-overview/ (дата звернення: 27.04.2025)
27. Козак О.Л. Опорний конспект лекцій з курсу — Аналіз вимог до програмного забезпечення для студентів напрямку підготовки — Програмна інженерія / О.Л. Козак. – Тернопіль, 2011. – 56 с.
28. Що таке API? [Електронний ресурс] // Hostiq.ua. [сайт]. – URL: https://hostiq.ua/blog/ukr/what-is-api/ (дата звернення: 04.05.2025)
29. Руденко В. Д. Інформатика: бази даних (модуль для учнів 10-11 класів, рівень стандарту) / В. Д. Руденко. – Харків: Вид-во «Ранок», 2019. – 112 с.

30. Що таке OAuth 2.0? [Електронний ресурс] // Auth0. [сайт]. – URL: https://auth0.com/intro-to-iam/what-is-oauth-2 (дата звернення: 14.05.2025)

ДОДАТКИ

ДОДАТОК А – Map.jsx

```
import React, { useState, useCallback, useRef, useEffect } from 'react';
import { GoogleMap } from '@react-google-maps/api';
import s from './Map.module.css';
import { CurrentLocationMarker } from '../CurrentLocationMarker';
const containerStyle = {
    width: '100%',
    height: '100%',
};
const defaultOptions = {
    panControl: true,
    zoomControl: true,
    mapTypeControl: false,
    streetViewControl: false,
    rotateControl: false,
    clickableIcons: false,
    scrollwheel: true,
    keyboardShortcuts: false,
    disableDoubleClickZoom: true,
    fullscreenControl: false
};
export const MODES = {
    MOVE: 0,
    SET_MARKER: 1,
};
const Map = ({ center, mode, markers, onMarkerAdd }) => {
    const [directionsResponse, setDirectionsResponse] = useState(null);
    const [markersInstances, setMarkersInstances] = useState([]);
    const mapRef = useRef(undefined);
    const directionsRendererRef = useRef(null);
    const onLoad = useCallback(function callback(map) {
        mapRef.current = map;
        if (directionsResponse) {
            directionsRendererRef.current.setMap(map);
        }
    }, [directionsResponse]);
    const onUnmount = useCallback(function callback() {
        markersInstances.forEach(marker => {
            if (marker) {
                marker.setMap(null);
            }
        });
        mapRef.current = undefined;
    }, [markersInstances]);

    const onClick = useCallback((loc) => {
```

```
        if (mode === MODES.SET_MARKER && mapRef.current) {
            const lat = loc.latLng.lat();
            const lng = loc.latLng.lng();

            const newMarker = new window.google.maps.Marker({
                position: { lat, lng },
                map: mapRef.current,
                icon: undefined
            });
            // Додаємо маркер до стану
            setMarkersInstances(prev => [...prev, newMarker]);
            // Повідомляємо батьківський компонент про нові координати
            onMarkerAdd({ lat, lng });
        }
    }, [mode, onMarkerAdd]);

    // Оновлюємо маркери при зміні пропсу markers
    useEffect(() => {
        if (!mapRef.current || !window.google) return;

        // Очищаємо попередні маркери
        markersInstances.forEach(marker => {
            if (marker) {
                marker.setMap(null);
            }
        });

        // Створюємо нові маркери
        const newMarkers = markers.map(marker => {
            return new window.google.maps.Marker({
                position: { lat: marker.lat, lng: marker.lng },
                map: mapRef.current
            });
        });

        setMarkersInstances(newMarkers);
    }, [markers]);

    // Функція для обчислення маршруту
    const calculateRoute = () => {
        if (markers.length < 2 || !window.google) return;

        const directionsService = new
window.google.maps.DirectionsService();
        const waypoints = markers.slice(1, markers.length - 1).map((marker)
=> ({
            location: new window.google.maps.LatLng(marker.lat,
marker.lng),
            stopover: true,
        }));
```

```
        const request = {
            origin: new window.google.maps.LatLng(markers[0].lat,
markers[0].lng),
            destination: new
window.google.maps.LatLng(markers[markers.length - 1].lat,
markers[markers.length - 1].lng),
            waypoints: waypoints,
            travelMode: window.google.maps.TravelMode.DRIVING,
        };

        directionsService.route(request, (result, status) => {
            if (status === window.google.maps.DirectionsStatus.OK) {
                setDirectionsResponse(result);
            } else {
                console.error('Directions request failed due to ' +
status);
            }
        });
    };
    // Відображення маршруту при зміні directionsResponse
    useEffect(() => {
        if (directionsResponse && mapRef.current && window.google) {
            const directionsRenderer = new
window.google.maps.DirectionsRenderer();
            directionsRenderer.setDirections(directionsResponse);
            directionsRenderer.setMap(mapRef.current);
            directionsRendererRef.current = directionsRenderer;
        }
    }, [directionsResponse]);

    return (
        <div className={s.container}>
            <GoogleMap
                mapContainerStyle={containerStyle}
                center={center}
                zoom={10}
                onLoad={onLoad}
                onUnmount={onUnmount}
                onClick={onClick}
                options={defaultOptions}
            >
                <CurrentLocationMarker position={center} />
            </GoogleMap>
            <button onClick={calculateRoute}>Calculate Route</button>
        </div>
    );
};

export { Map };
```

ДОДАТОК Б – Autocomplete.jsx

```
import React from "react";
import usePlacesAutocomplete, {
    getGeocode,
    getLatLng,
} from "use-places-autocomplete";
import useOnclickOutside from "react-cool-onclickoutside";
import s from './Autocomplete.module.css'

export const Autocomplete = ({isLoaded, onSelect}) => {
    const {
        ready,
        value,
        suggestions: { status, data },
        setValue,
        init,
        clearSuggestions,
    } = usePlacesAutocomplete({
        initOnMount: false,
        debounce: 300,
    });

    const ref = useOnclickOutside(() => {
        clearSuggestions();
    });

    const handleInput = (e) => {
        setValue(e.target.value);
    };

    const handleSelect =
        ({ description }) =>
            () => {
                setValue(description, false);
                clearSuggestions();
                getGeocode({ address: description })
                .then((results) => {
                    const { lat, lng } = getLatLng(results[0]);
                    console.log("📍 Coordinates: ", { lat, lng });
                    onSelect({ lat, lng })
                });
            };

    const renderSuggestions = () =>
        data.map((suggestion) => {
            const {
                place_id,
                structured_formatting: { main_text, secondary_text },
```

```
            } = suggestion;
            return (
                <li className={s.listItem} key={place_id}
onClick={handleSelect(suggestion)}>
                    <strong>{main_text}</strong>
<small>{secondary_text}</small>
                </li>
            );
        });

    React.useEffect(()=>{
        if(isLoaded) {
            init()
        }
    },[isLoaded, init])

    return <div className={s.root} ref={ref}>
        <input
            type="text"
            className={s.input}
            value={value}
            onChange={handleInput}
            disabled={!ready}
            placeholder="Where are you going?"
        />
        {status === "OK" && <ul
className={s.suggestions}>{renderSuggestions()}</ul>}
    </div>
}
```

ДОДАТОК В – Calendar.jsx

```
import React, { useRef, useState } from "react";
import axios from 'axios';
import '../index.css';
import { useJsApiLoader, Autocomplete } from '@react-google-maps/api';

export default function Calendar() {
    const { isLoaded } = useJsApiLoader({
        googleMapsApiKey: process.env.REACT_APP_GOOGLE_MAPS_API_KEY,
        libraries: ['places'],
    });
    const [description, setDescription] = useState('');
    const [location, setLocation] = useState('');
    const [startDateTime, setStartDateTime] = useState('');
    const [endDateTime, setEndDateTime] = useState('');

    /** @type React.MutableRefObject<HTMLInputElement> */
    const originRef = useRef();

    if (!isLoaded) {
        return <div>Loading...</div>;
    }

    function handleSubmit(e) {
        e.preventDefault();
        console.log(location, description, location, startDateTime,
endDateTime);
        axios.post('http://localhost:3001/auth/calendar', {
            location,
            description,
            startDateTime,
            endDateTime
        })
            .then(response => console.log(response.data))
            .catch(error => console.log(error.message));
    }

    const handlePlaceChanged = () => {
        const place = originRef.current.getPlace();
        if (place && place.formatted_address) {
            setLocation(place.formatted_address); // Оновлюємо стан локації
з вибраного місця
        }
    };

    return (
        <>
            <form onSubmit={handleSubmit}>
```

```
                <label htmlFor="location"
className="placelabel">Place</label>
                <Autocomplete
                    onLoad={(autocomplete) => {
                        originRef.current = autocomplete; // Прив'язуємо
реф до компонента Autocomplete
                    }}
                    onPlaceChanged={handlePlaceChanged}
                >
                    <input
                        type="text"
                        className="placeinput"
                        placeholder="name"
                        id="location"
                        value={location}
                        onChange={e => setLocation(e.target.value)} // Це
дозволяє вводити текст вручну
                    />
                </Autocomplete>
                <div className="menu">
                <textarea id="description" value={description} onChange={e
=> setDescription(e.target.value)} placeholder="Comment"/>
                <div className="datemenu">
                <label htmlFor="startDateTime" className="placelabel">Data
start</label>
                <input type="datetime-local" className="date"
id="startDateTime" placeholder="date start" value={startDateTime}
onChange={e => setStartDateTime(e.target.value)} />
                <label htmlFor="endDateTime" className="placelabel">Data
end</label>
                <input type="datetime-local" className="date"
id="endDateTime" placeholder="date end" value={endDateTime} onChange={e =>
setEndDateTime(e.target.value)} />
                </div>
                </div>
                <button type="submit" className="buttoncreate">Create
Event</button>

            </form>
        </>
    );
}
```

ДОДАТОК Г – Autorization.jsx

```
import { useGoogleLogin } from '@react-oauth/google';
import axios from 'axios';

function GoogleAuthButton({setSignedId}) {
    const googleLogin = useGoogleLogin({
        onSuccess: async ({ code }) => {
            try {
                // Відправляємо код на ваш сервер для обміну на токени
                const { data } = await axios
                .post('http://localhost:3001/auth/google', { code })

                localStorage.setItem('access_token', data.access_token);
                localStorage.setItem('refresh_token', data.refresh_token);
                setSignedId(true);
            } catch (error) {
                console.error('Error exchanging code:', error);
                setSignedId(false);
            }
        },
        onError: (error) => {
            console.error('Google Login Error:', error);
        },
        flow: 'auth-code',
        scope: 'openid email profile
https://www.googleapis.com/auth/calendar',
    });

    return (
        <button onClick={() => googleLogin()}>
            Sign in with Google 🚀
        </button>
    );
}

export default GoogleAuthButton;
```

ДОДАТОК Д – app.js

```
require('dotenv').config();
const express = require('express');
const { OAuth2Client, UserRefreshClient } = require('google-auth-library');
const cors = require('cors');
const { google } = require('googleapis');
const app = express();

// Оголошення змінної для зберігання refreshToken
let storedRefreshToken = null;

app.use(cors());
app.use(express.json());

const oAuth2Client = new OAuth2Client(
  process.env.CLIENT_ID,
  process.env.CLIENT_SECRET,
  'postmessage'
);

app.post('/auth/google', async (req, res) => {
  try {
    const { tokens } = await oAuth2Client.getToken(req.body.code);
    console.log('Received tokens:', tokens);

    // Зберігаємо refreshToken
    if (tokens.refresh_token) {
      storedRefreshToken = tokens.refresh_token;
      console.log('Refresh token stored:', storedRefreshToken);
    }

    res.json(tokens);
  } catch (error) {
    console.error('Error exchanging code:', error);
    res.status(400).json({ error: 'Failed to exchange code' });
  }
});

app.post('/auth/google/refresh-token', async (req, res) => {
  try {
    const refreshToken = storedRefreshToken || req.body.refreshToken;

    if (!refreshToken) {
      return res.status(400).json({ error: 'No refresh token available' });
    }

    const user = new UserRefreshClient(
      process.env.CLIENT_ID,
```

```
      process.env.CLIENT_SECRET,
      refreshToken
    );

    const { credentials } = await user.refreshAccessToken();
    res.json(credentials);
  } catch (error) {
    console.error('Error refreshing token:', error);
    res.status(400).json({ error: 'Failed to refresh token' });
  }
});

app.get('/auth/current-token', (req, res) => {
  // Ендпоінт для перевірки збереженого токена (для дебагу)
  res.json({ storedRefreshToken });
});

app.post('/auth/calendar', async (req,res,next)=>{
  try{
    const {
      location,
      description,
      startDateTime,
      endDateTime
    }=req.body
    oAuth2Client.setCredentials({refresh_token:storedRefreshToken})
    const calendar=google.calendar('v3')
    const response=await calendar.events.insert({
      auth:oAuth2Client,
      calendarId:"primary",
      requestBody:{
        summary:location,
        location:location,
        description: description,
        colorId:'3',
        start: {
          dateTime:new Date(startDateTime),
        },
        end: {
          dateTime:new Date(endDateTime),
        }
      }
    })
    res.send(response)
  }catch(error){
    next(error)
  }
})
app.listen(3001, () => console.log('Server is running on port 3001'));
```