МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-наукової програми «Інформаційні технології проектування»

на тему: «Веборієнтована інформаційна система планування туристичних подорожей»

Здобувачки групи ІТ.м-32 Подус Катерини Олександрівни
(шифр групи) (прізвище, ім'я, по батькові)

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

_____ Катерина ПОДУС
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник к.т.н., доц. Світлана ВАЩЕНКО _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-наукова програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО
«_____» _____ 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Подус Катерині Олександрівні
(прізвище, ім'я, по батькові)

1 Тема кваліфікаційної роботи «Веборієнтована інформаційна система планування туристичних подорожей»

затверджена наказом по університету від «11» жовтня 2024 р. № 1044-VI

2 Термін здачі студентом кваліфікаційної роботи «6» грудня 2024 р.

3 Вхідні дані до кваліфікаційної роботи документації бібліотек і фреймворків, матеріали дослідження та статистики, продукти-аналоги

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз і дослідження теми; дослідження продуктів-аналогів; визначення функціональних вимог; моделювання та проектування; розробка функціоналу.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) аналіз предметної області; постановка задачі; аналіз програмних продуктів-аналогів; порівняння продуктів-аналогів; методи кластеризації; математична модель модифікованого алгоритму; функціональні вимоги інформаційної системи; контекстна діаграма в нотації IDEF0 з точки зору користувача; перший рівень декомпозиції IDEF0-діаграми з точки зору користувача; другий рівень декомпозиції IDEF0-діаграми з точки зору користувача; діаграма варіантів використання; діаграма послідовності; діаграма діяльності; проєктування бази даних; засоби реалізації; архітектура вебдодатку веборієнтованої інформаційної системи планування подорожей; сторінка профілю та авторизації; сторінка всіх подорожей; сторінка конкретної подорожі; сторінка готелей, списку бажаних готелей та бронювання; сторінка списку речей,

розрахунку бюджету подорожі та приклад темного режиму вебдодатку; висновки.

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Проведення аналізу і дослідження даної теми	до 19.08.2024	
2	Проведення дослідження продуктів-аналогів	до 21.08.2024	
3	Визначення функціональних вимог	до 28.08.2024	
4	Виконання моделювання та проєктування	до 03.09.2024	
5	Розробка функціоналу	до 28.11.2024	

Магістрант _____ Катерина ПОДУС

Керівник роботи _____ к.т.н., доц. Світлана ВАЩЕНКО

АНОТАЦІЯ

Тема кваліфікаційної роботи магістра «Веборієнтована інформаційна система планування туристичних подорожей».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 33 найменувань, додатків. Загальний обсяг роботи − 209 сторінок, у тому числі 66 сторінок основного тексту, 4 сторінки списку використаних джерел, 143 сторінки додатків.

Актуальність роботи полягає в спрощенні планування подорожей, яке ускладнюється розподіленістю інформації. Централізоване рішення інтегрує розробку оптимальних маршрутів на кожен день, бронювання, розрахунок і аналіз витрат, а також створення переліку необхідних речей.

Мета роботи: розробка веборієнтованої інформаційної системи планування туристичних подорожей, спрямованої на автоматизацію процесів створення оптимізованих маршрутів, а також надання користувачам комплексного інструментарію для ефективної організації подорожей, включаючи функції навігації, розрахунку витрат та інформацію про бронювання.

Перший розділ аналізує сучасні рішення для автоматизації планування подорожей, зокрема огляд досліджень, аналіз програмних продуктів-аналогів і порівняння їх характеристик та дослідження API для бронювання готелів.

Другий розділ зосереджений на постановці задачі дослідження. У ньому сформульовано мету, визначено ключові задачі та обґрунтовано методи дослідження.

Третій розділ охоплює моделювання та проєктування системи, включаючи структурно-функціональне моделювання, створення UML-моделей варіантів використання та послідовності, а також розробку логічної моделі бази даних.

Четвертий розділ охоплює архітектуру та реалізацію системи, описуючи структуру програмного додатка, використані технології та приклад застосування.

Ключові слова: веборієнтована інформаційна система, планування туристичних подорожей, кластеризація, оптимальні маршрути, бронювання, список речей, бюджет.

ЗМІСТ

ВСТУП

Незважаючи на значний технологічний прогрес та доступність інформації в сучасному світі, процес планування туристичної подорожі людиною залишається складним завданням, яке важко охопити лише ментально. Він включає розробку маршруту, придбання квитків, бронювання житла та інші аспекти, що часто призводить до накопичення значного обсягу інформації, розподіленої між різними джерелами та додатками. Це ускладнює ефективне управління і використання даних, створюючи потребу в централізованому рішенні.

Саме тому розробка веборієнтованої інформаційної системи планування туристичних подорожей набуває особливої актуальності. Така система здатна забезпечити централізоване зберігання всіх необхідних даних з легким та зручним доступом до них, інтегруючи різноманітні функції, такі як: створення та управління маршрутами, заходження оптимальних маршрутів в залежності від кількості днів і часу, розрахунок бюджету, бронювання квитків, використання навігації безпосередньо під час подорожі.

Об’єкт дослідження – інформаційні технології, які застосовуються для планування маршрутів туристичних подорожей.

Предмет дослідження – програмні засоби реалізації планування подорожей з побудовою оптимальних маршрутів.

Метою магістерської роботи є розробка веборієнтованої інформаційної системи планування туристичних подорожей, спрямованої на автоматизацію процесів створення оптимізованих маршрутів, а також надання користувачам комплексного інструментарію для ефективної організації подорожей, включаючи функції навігації, розрахунку витрат та інформацію про бронювання.

Для досягнення мети необхідно виконати наступні поставлені задачі:

– провести аналіз існуючих веборієнтованих платформ для планування подорожей, щоб визначити їх переваги та недоліки; дослідити методи кластеризації для вирішення задачі розподілу локацій по дням подорожі;

– змоделювати роботу та розробити архітектуру системи з акцентом на масштабованість, безперебійну роботу та інтеграцію з картографічними платформами;

– реалізувати модуль кластеризації пам’яток за днями подорожі з автоматизованою оптимізацією маршрутів на основі географічного розташування;

– створити адаптивний інтерфейс із функціями створення, збереження та редагування подорожей і маршрутів в них, управління бронюванням, створення списків речей для подорожі та розрахунком бюджету за категоріями витрат;

Практична значущість дослідження полягає у розробці інноваційного програмного рішення, що дозволяє суттєво підвищити ефективність процесів планування туристичних подорожей як на індивідуальному рівні, так і в масштабах туристичних організацій. Запропонована система надає потужний інструментарій для автоматизованої побудови оптимальних маршрутів з урахуванням часових та бюджетних обмежень, що значно скорочує витрати ресурсів на етапі планування. Особлива увага приділяється персоналізації та адаптивності системи, що дозволяє задовольнити потреби як індивідуальних туристів, так і професійних туроператорів при формуванні унікальних туристичних пропозицій. Впровадження даної системи сприяє підвищенню якості туристичних послуг та забезпечує комплексний підхід до організації подорожей з максимальним урахуванням потреб кінцевого користувача.

Створення такої системи не лише відповідає на поточні виклики туристичної галузі, але й готує ґрунт для майбутнього зростання та інновацій у сфері планування подорожей, сприяючи відновленню та розвитку туризму в постпандемічну епоху.

1 АНАЛІЗ АКТУАЛЬНИХ РІШЕНЬ ДЛЯ АВТОМАТИЗАЦІЇ ПЛАНУВАННЯ ТУРИСТИЧНИХ ПОДОРОЖЕЙ

1.1 Огляд останніх досліджень і публікацій

У сучасному світі автоматизація процесів планування туристичних подорожей набуває все більшої актуальності. Незважаючи на широкий доступ до інформації в мережі Інтернет, самостійна організація подорожі залишається складним завданням, особливо щодо оптимізації маршрутів та врахування індивідуальних побажань. Інформаційні системи для сфери туризму відіграють ключову роль у вирішенні цих проблем, забезпечуючи ефективне ранжування, вибір та маршрутизацію визначних місць в умовах обмеженого часу. Впровадження сучасних технологій дозволяє створювати індивідуальні оптимальні маршрути, автоматизувати процеси бронювання та розрахунку витрат, підвищуючи гнучкість та якість індивідуального туризму [1].

За даними Всесвітньої туристичної організації (UNWTO), у 2019 році кількість міжнародних туристичних прибуттів досягла близько 1,5 мільярда, що на 4%, або на 54 мільйони перевищує показники 2018 року [2]. Однак, пандемія COVID-19 та війна в Україні призвели до різкого спаду у туристичній галузі. Незважаючи на це, згідно з останніми даними UNWTO, світовий туризм демонструє ознаки відновлення: у першому кварталі 2024 року кількість міжнародних туристів досягла 97% від допандемічного рівня. За даними ООН з туризму, у січні-березні понад 285 мільйонів туристів здійснили міжнародні подорожі, приблизно на 20% більше, ніж у першому кварталі 2023 року, що підкреслює майже повне відновлення після наслідків пандемії [3].

Позитивна динаміка відновлення туристичної галузі акцентує увагу на нагальній потребі розробки інноваційних інструментів планування подорожей, здатних не лише задовольнити зростаючий попит, але й адаптуватися до трансформованих реалій індустрії. Історичний аналіз демонструє, що еволюція

засобів пересування завжди корелювала з удосконаленням методів планування подорожей. Яскравим прикладом цього взаємозв'язку слугує епоха Великих географічних відкриттів, яка стала можливою завдяки синергії розвитку мореплавства та інновацій у картографії, навігації та плануванні експедицій. Ця тенденція знаходить своє продовження і в сучасному світі: впровадження цифрових технологій надає мандрівникам доступ до безпрецедентно потужного інструментарію, який суттєво оптимізує та вдосконалює процес організації подорожей.

Незважаючи на те, що інформаційні технології мають великий вплив на розвиток сфери туризму, вони містять свої переваги та недоліки, які добре описані авторами Паньків, Н., та Гуменяк В. у своїй статті «Діджиталізація туристичних маршрутів в Україні: сучасний стан та тенденції розвитку» [4].

Отже, до переваг впровадження таких інформаційних систем належать:

–стимулювання інновацій;

–розширення ринку збуту;

–зниження витрат;

–підвищення безпеки, якості операцій, ефективності діяльності підприємств, прибутковості, конкурентоспроможності;

–формування споживчої цінності.

До недоліків впровадження інформаційних систем належать:

–залежність від технологій;

–приватність і безпека;

–кіберзлочини;

–вартість інфраструктури [4].

Таким чином, розробка та впровадження інноваційних інформаційних систем для планування подорожей є одним з ключових факторів модернізації туристичної галузі. Проте, для досягнення максимальної ефективності та мінімізації потенційних ризиків, необхідно постійно вдосконалювати ці системи, враховуючи як переваги, так і виклики цифровізації туризму.

1.2 Аналіз програмних продуктів – аналогів

У контексті розробки веборієнтованої інформаційної системи планування туристичних подорожей було проведено компаративний аналіз існуючих рішень на світовому ринку туристичних сервісів. Дослідження спрямоване на виявлення оптимальних підходів до реалізації функціональних можливостей та архітектурних рішень, що забезпечують максимальну ефективність процесу планування подорожей.

В рамках дослідження було проаналізовано провідні платформи туристичної індустрії:

–TripAdvisor [5];

–Roadtrippers [6];

–Wanderlog [7];

Комплексний аналіз функціональних можливостей, архітектурних особливостей та користувацького досвіду даних платформ дозволив сформувати чітке розуміння критичних компонентів, що мають бути імплементовані у розроблюваній системі. Особлива увага приділялася дослідженню механізмів оптимізації маршрутів, систем управління користувацьким контентом та методів інтеграції різноманітних туристичних сервісів.

Результати проведеного дослідження становлять теоретичну та практичну базу для формування функціональних вимог до веборієнтованої інформаційної системи планування туристичних подорожей, що розробляється. Це забезпечить створення конкурентоспроможного продукту, який відповідає сучасним вимогам ринку та задовольняє потреби цільової аудиторії у контексті ефективної організації подорожей.

1.2.1 TripAdvisor

Одним з найпопулярніших вебсайтів для планування туристичних подорожей наразі є TripAdvisor, який щомісяця обслуговує 315 мільйонів унікальних відвідувачів та має понад 70 мільйонів зареєстрованих

користувачів [5]. До основного функціоналу даної платформи входить пошук і перегляд різних туристичних місць, готелів, ресторанів, з наданням великої кількості відгуків від людей. Завдяки TripAdvisor, можна порівняти ціни на різні послуги, виконати бронювання будь-яких білетів, безпосередньо через сайт і спланувати власну подорож з можливістю її редагування з іншими людьми. Також доступний функціонал планування, пошуку та збереження локацій на кожен день подорожі.

На головній сторінці сайту можна побачити пошук за різними категоріями, такими як: готелі, розваги, ресторани, рейси літаків, оренда помешкання (рис. 1.1). Крім цього, пропонуються найкращі подорожі цього року та можливість створення подорожі завдяки штучному інтелекту.

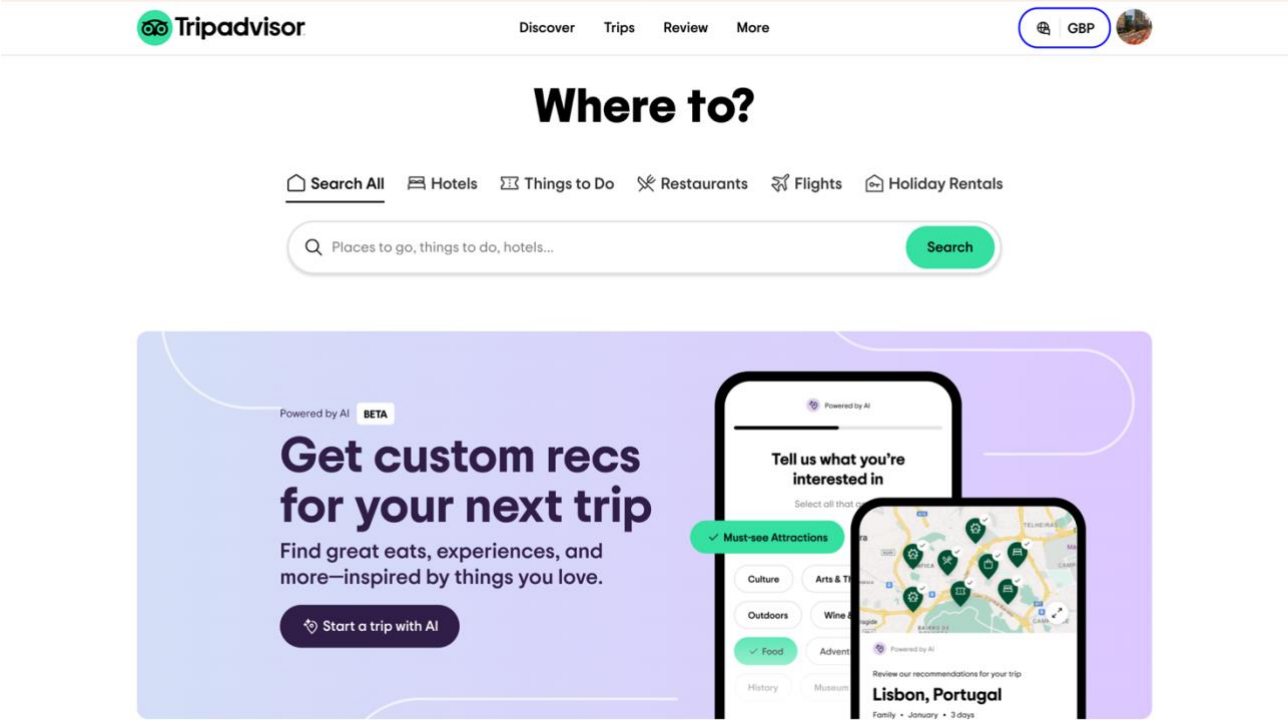


Рисунок 1.1 – Головна сторінка вебсайту «TripAdvisor»

Джерело:[5]

Перейшовши на сторінку «Trips», можна переглянути всі наявні подорожі та основну інформацію про них (рис. 1.2). Також тут є можливість створення нової подорожі самому, або за допомогою AI асистента. Одним з найвагоміших

мінусів цього сайту для мешканців України є те, що немає можливості обрати наш регіон та валюту, що ускладнює процес планування.

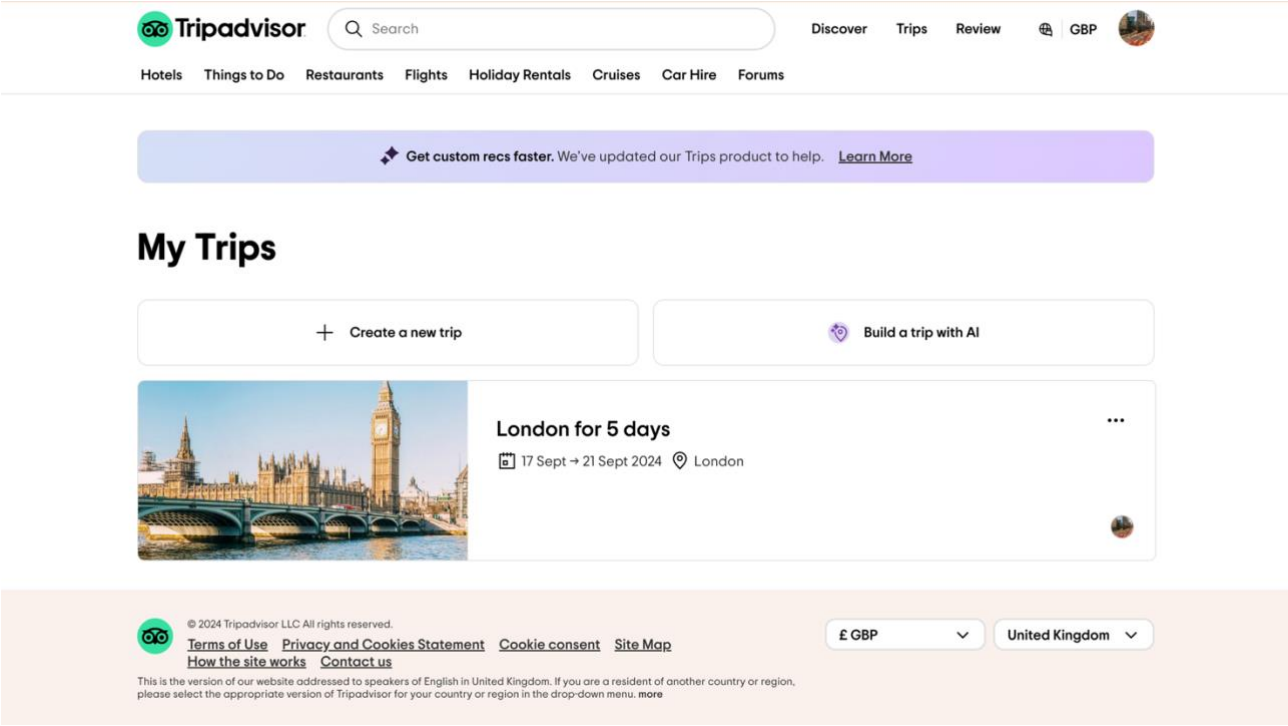


Рисунок 1.2 – Сторінка всіх подорожей «TripAdvisor»

Джерело:[5]

Після створення нової подорожі, виконується перехід на сторінку з усіма її деталями планування (рис. 1.3). На цій сторінці у вкладці «Itinenary» можна створити список всіх локацій на кожен день, з урахуванням готелей, ресторанів, заходів тощо. Також у вкладці «For you» користувачам надаються рекомендації різних місць у межах районів вашої подорожі, найпривабливіші з яких можна зберегти, щоб потім переглянути у вкладці «Save». Ця сторінка допомагає зручно спланувати свій графік на кожен день, проте не надає можливості для створення оптимального маршруту та не відображає його на карті, а також немає функціоналу для підрахунку бюджету подорожі. Тобто у користувача немає можливості скористатися картою та навігацією безпосередньо під час самої прогулянки. Крім цього, під час додавання нових місць до списку локацій на день, дуже важко знайти майже кожне місцезнаходження, бо пошук

здійснюється лише завдяки транслітерації з української на англійську, але навіть за таких умов, багато визначних пам’яток повністю відсутні.

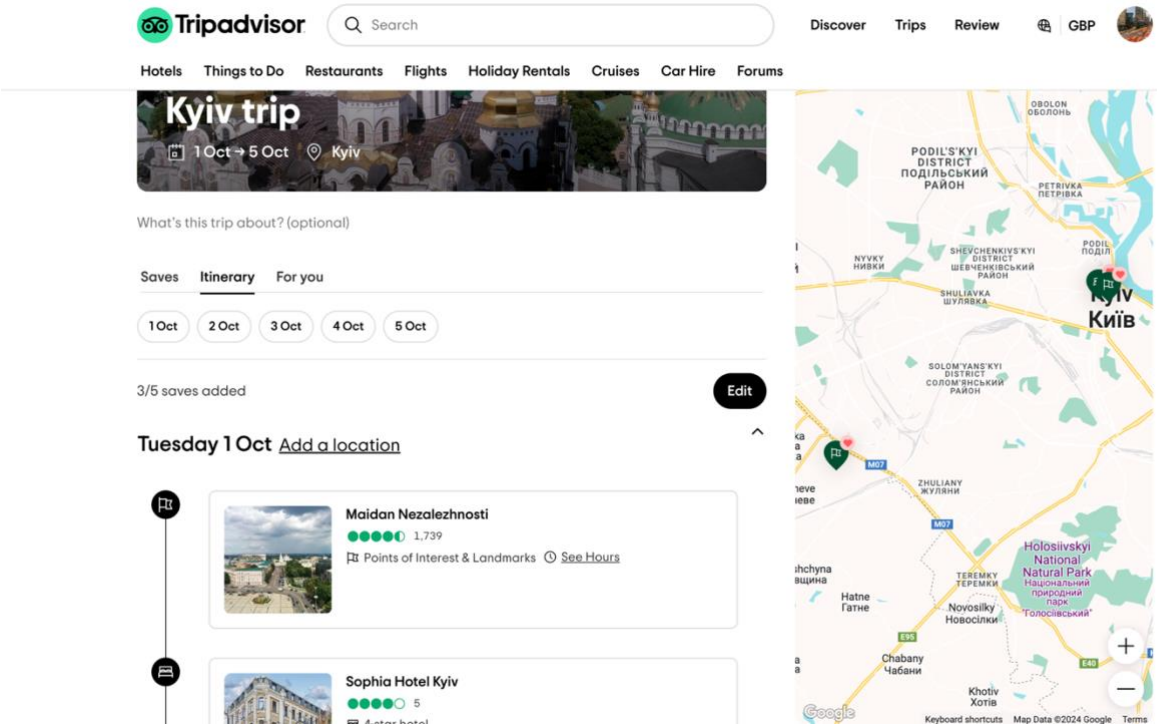


Рисунок 1.3 – Сторінка подорожі «TripAdvisor»

Джерело:[5]

1.2.2 Roadtrippers

Ще одним досить популярним інструментом для планування подорожі є Roadtrippers. Цей вебдодаток набуває особливої уваги в Сполучених Штатах Америки і Канаді [6]. Основний функціонал Roadtrippers схожий на той, що має TripAdvisor, до нього відносяться створення подорожі, планування маршруту, співпраця з іншими людьми, отримання рекомендацій на базі вподобань. Додатково доступні функції розрахунку бюджету під час подорожі, використання додатку в дорозі навіть офлайн та оптимізація маршруту.

На головній сторінці сайту Roadtrippers відразу можна почати будувати свій маршрут, обираючи початкову та кінцеву точку подорожі (рис. 1.4). Після цього треба надати уточнюючі дані для більш детального планування. Для полегшення цього процесу пропонується використати штучний інтелект, який

допоможе налаштувати подорож, але він недоступний для безкоштовної версії продукту.

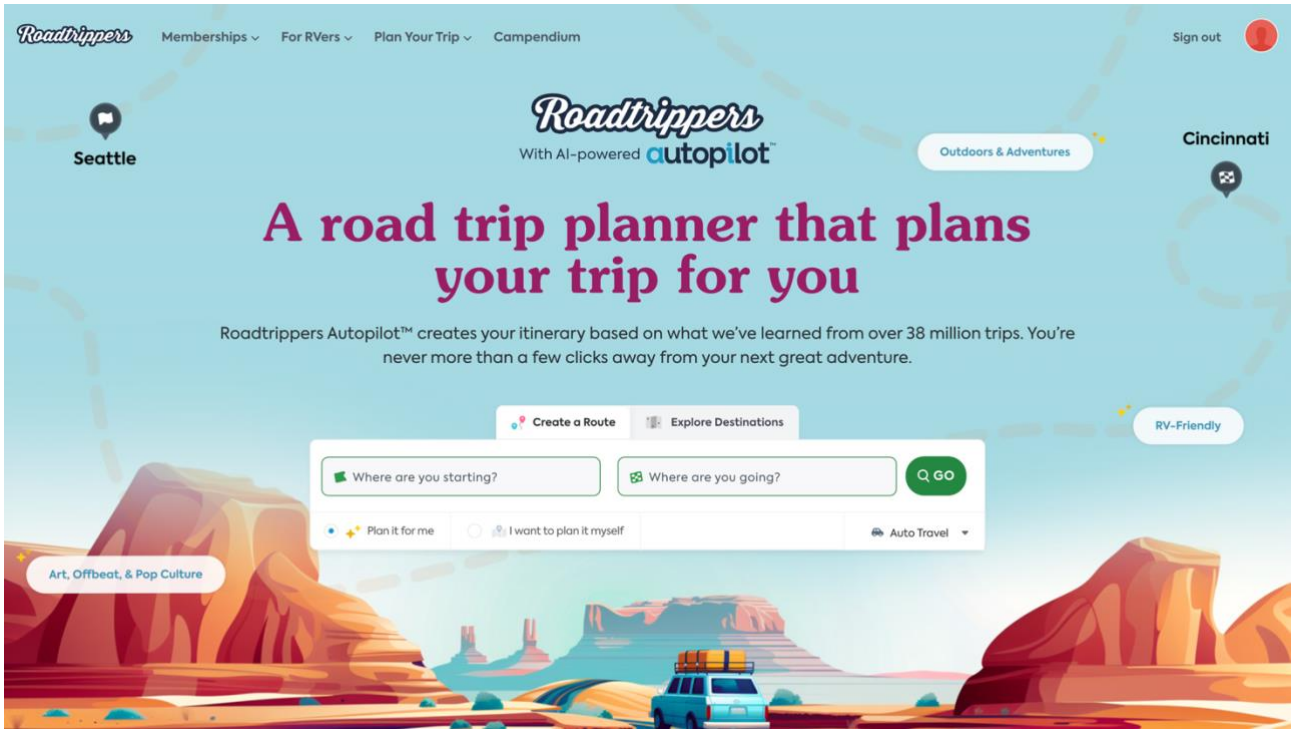


Рисунок 1.4 – Головна сторінка «Roadtrippers»

Джерело:[6]

Наступна сторінка «My Trips» містить вже створений набір подорожей з короткою інформацією про них і можливістю редагування (рис. 1.5).

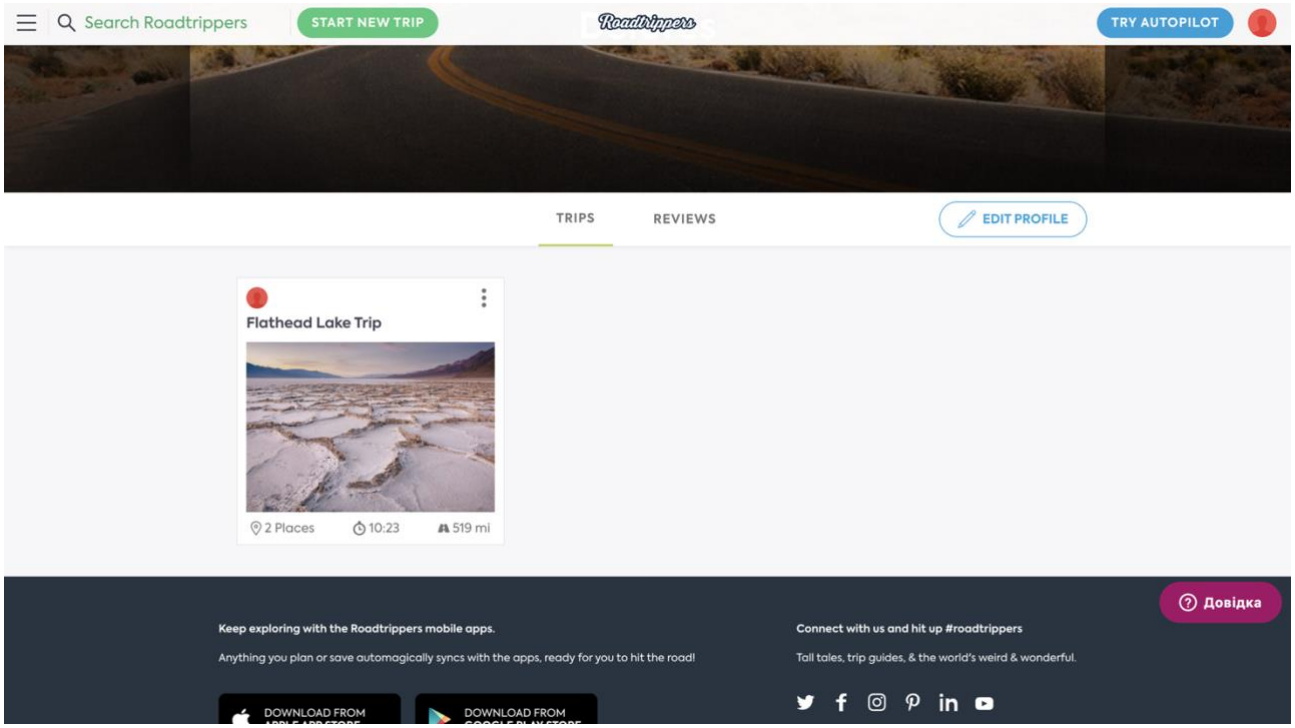


Рисунок 1.5 – Сторінка всіх подорожей «Roadtrippers»

Джерело:[6]

Далі, перейшовши до самої подорожі, відкривається великий список можливостей для її планування, серед якого можна на карті або в пошуку обирати локації та заносити їх як точки маршруту (рис. 1.6). Для кожної локації можна визначити її власні параметри, такі як дата, бюджет, бронювання та примітки. Також на панелі зліва міститься список з різними рекомендаціями щодо готелів, ресторанів тощо. Там можна знайти більшість необхідних локацій для додавання у власний маршрут.

Отже, серед переваг даного вебдодатку можна зазначити легкий для розуміння інтерфейс, інтерактивну карту, інтеграцію з навігаційними системами, зручне налаштування маршруту та його деталей, рекомендаційні системи. Проте, у користувача немає можливості оптимізувати свій шлях за певну кількість днів, підрахувати бюджет, а також фокус цього сайту більше цілеспрямований на США та Канаду, бо в інших регіонах набагато менше доступної інформації про визначні місця, що унеможливлює їх додавання до маршруту напряму через карту.

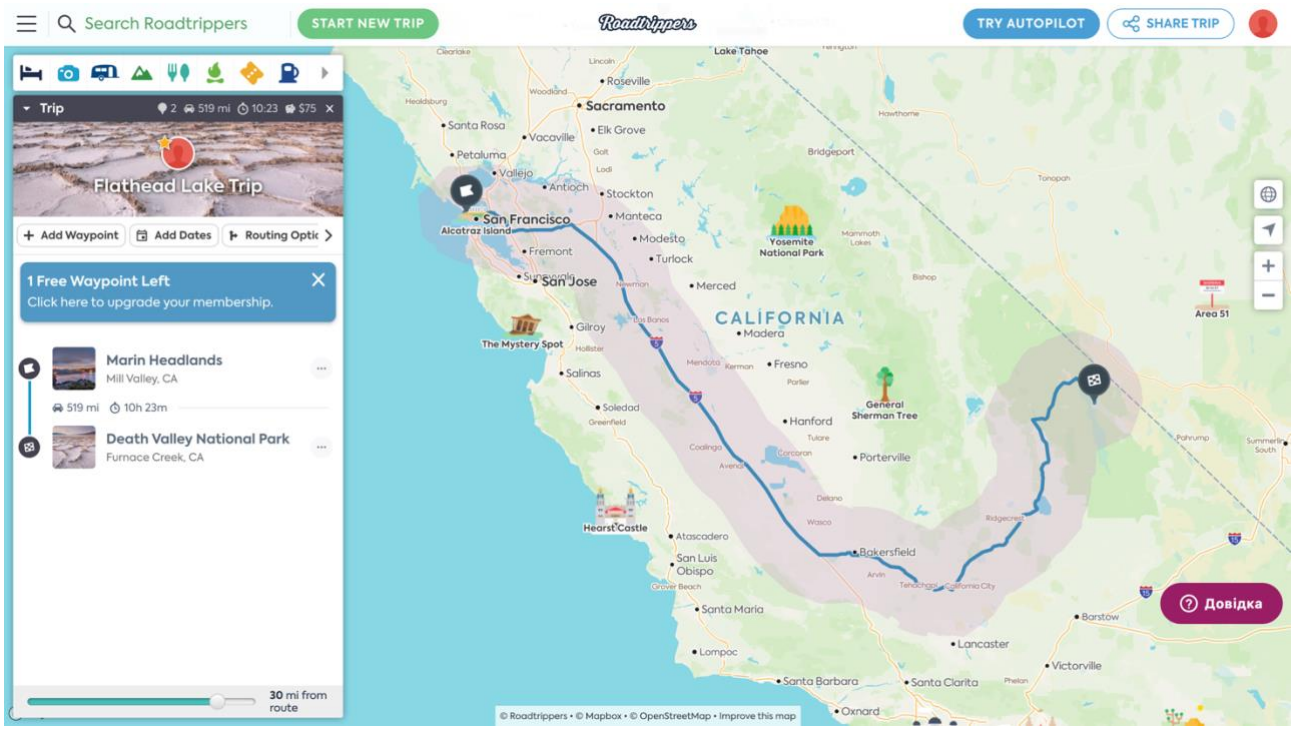


Рисунок 1.6 – Сторінка подорожі «Roadtrippers»

Джерело:[6]

1.2.3 Wanderlog

Останнім вебсайтом для планування подорожей розглянемо Wanderlog – Travel Planner. Цей вебдодаток створений для планування будь-яких подорожей, у тому числі автомобільних і групових. Серед його можливостей є створення маршруту поїздки, планування витрат, бронювання квитків і готелів, перегляд місць на карті, які варто відвідати та співпраця з іншими людьми [7].

Нижче, на рисунку 1.7 представлена головна сторінка Wanderlog, на якій можна переглянути свої останні сплановані подорожі та створити нові.

На сторінці планування подорожі користувачі можуть додавати локації маршруту та його деталі у вигляді часу, коштів і примітки. Також, окрім місць на карті, доступні для додавання у планування дня списки та нотатки. Wanderlog також пропонує рекомендації щодо популярних пам’яток та закладів у вибраному місці призначення. На карті можна переглянути маршрути на кожен день, для зручності також можна їх відкрити у Google Maps та інших програмах.

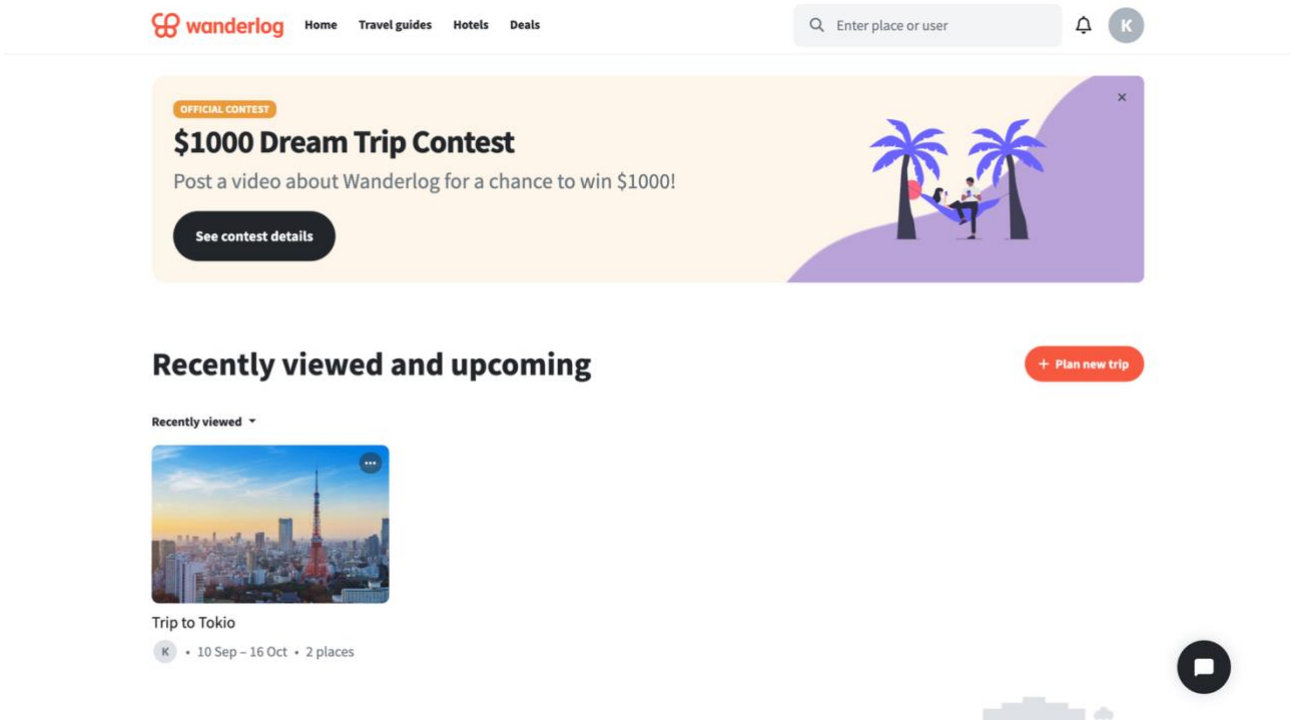


Рисунок 1.7 – Головна сторінка «Wanderlog»

Джерело:[7]

При оформленні передплати за повний функціонал продукту, можна також отримати доступ для оптимізації маршрутів. Крім цього, додаючи інформацію про точки маршрутів, можна вказувати запланований бюджет, який в результаті підрахунків за всі дні складається в єдине число.

Підсумовуючи, Wanderlog надає великий функціонал для зручного та інтуїтивного планування подорожі, створюючи акцент на візуалізації маршруту на карті та її використанні, безпосередньо, під час прогулянки. Незважаючи на широкий потенціал цього додатку, він містить свої негативні сторони, які висвітлюються в складності використання додатку новими користувачами та обмеженням для оптимізації маршрутів через передплату.

Вигляд сторінки для налаштувань власної подорожу наведений на рисунку 1.8.

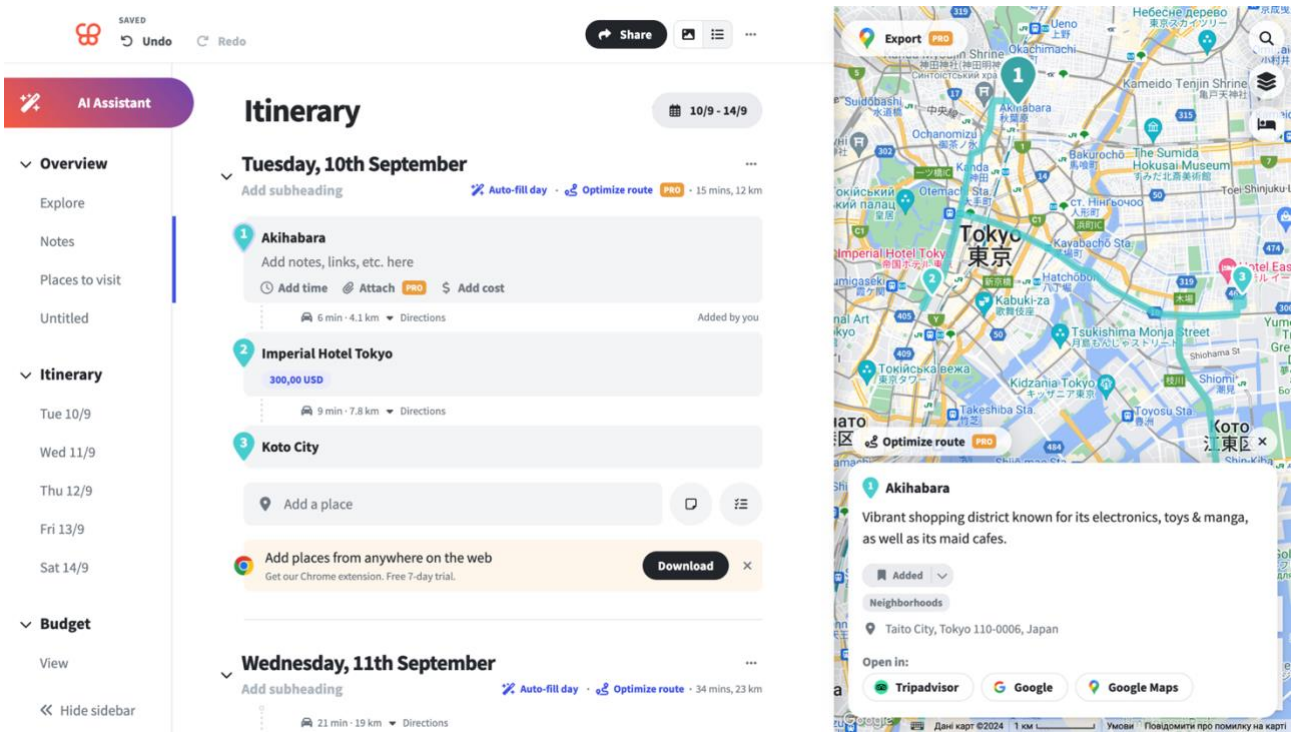


Рисунок 1.8 – Сторінка подорожі «Wanderlog»

Джерело:[7]

1.3Аналіз API ресурсів бронювання готелів

Під час проведення аналізу інструментів для створення бронювання в системі було розглянуто декілька відомих API, які можуть допомогти в інтеграції процесу бронювання в додатку. До найпопулярніших рішень належать:

–Booking.com API – цей API надає можливість інтеграції з одним із найбільших сервісів для бронювання готелів та іншого житла. Він надає доступ до широкого каталогу об'єктів для проживання та дозволяє відображати ціни, наявність та здійснювати бронювання. Проте цей API не є безкоштовним; використання передбачає підписання партнерської угоди та надання певного обсягу бронювань [8].

–Hotels.com API – цей сервіс також надає доступ до бронювання житла, проте більшість доступу через цей API доступна лише за партнерською програмою або за платною основою. Детальна інформація про умови

використання вимагає додаткового запиту до представників сервісу або партнерської угоди. Якщо потрібна додаткова інформація, можу знайти джерела для глибшого аналізу [9].

–TripAdvisor Content API – цей API забезпечує доступ до широкого спектра контенту, такого як відгуки, рейтинги готелів, ресторанів та інших туристичних об'єктів, фотографії подорожей, списки кращих готелів у місцевостях, нагороди Travelers' Choice тощо. Однак цей API вимагає реєстрації в партнерській програмі та погодження умов використання [10].

–Travel System API – даний API пропонує широкий вибір готелів для туристичних агентств, просту інтеграцію та програму лояльності для B2B партнерів. Зазвичай цей API платний або має вимоги до участі у партнерській програмі [11].

Таким чином, майже всі популярні API для бронювання наразі є платними або вимагають участі у партнерських програмах, що також передбачає певні зобов'язання та фінансові вкладення.

З урахуванням вищесказаного, прийнято рішення, що буде створено тимчасовий модуль для використання замість API Booking, який взаємодіятиме з власною базою даних, що міститиме основну інформацію, необхідну для бронювання. Структура бази даних включатиме такі основні таблиці, як: готелі (з інформацією про назву, місце розташування, категорію, опис тощо), номери (тип, кількість ліжок, вартість за ніч, статус доступності), бронювання (інформація про клієнта, дати, статус бронювання), а також відгуки (рейтинг, коментарі, дати).

1.4 Порівняння характеристик продуктів-аналогів

В рамках дослідження було проведено комплексний аналіз функціональних можливостей та технічних характеристик провідних

веборієнтованих систем планування подорожей: TripAdvisor, Roadtrippers та Wanderlog. Дослідження базувалося на оцінці ключових параметрів, що визначають ефективність та зручність використання подібних платформ в контексті сучасних вимог користувачів. Результати оцінки можна наглядно побачити у таблиці 1.1.

Аналіз інтерфейсних рішень досліджуваних платформ показав, що всі вони характеризуються сучасним дизайном та інтуїтивно зрозумілою навігацією, при цьому TripAdvisor та Roadtrippers демонструють найвищий рівень зручності використання. Wanderlog, маючи дещо складнішу структуру навігації, компенсує це розширеним функціоналом для досвідчених користувачів. Важливою характеристикою всіх платформ є їх адаптивність та кросплатформність, що забезпечує зручний доступ з різних пристроїв та операційних систем.

Функціональний аналіз виявив, що всі досліджувані системи надають базові можливості для створення та планування подорожей, включаючи роботу з точками інтересу та інформацією про бронювання. Особливої уваги заслуговує функціонал Wanderlog щодо оптимізації маршрутів з урахуванням часових обмежень та можливості розрахунку загального бюджету подорожі, що відсутній у конкурентних рішеннях.

Істотною перевагою Roadtrippers та Wanderlog є інтеграція навігаційних можливостей, що дозволяє користувачам відслідковувати свій маршрут у режимі реального часу. При цьому TripAdvisor та Wanderlog демонструють більш широку географічну орієнтованість, на відміну від Roadtrippers, який переважно фокусується на північноамериканському ринку.

Результати проведеного аналізу дозволили сформувати комплексне бачення критично важливих компонентів, що мають бути імплементовані у розроблюваній системі. Ключовими аспектами є: інтуїтивно зрозумілий інтерфейс, функціонал створення та оптимізації маршрутів, інтегрована навігація та підтримка користувачів на всіх етапах планування подорожі. Врахування виявлених характеристик та переваг існуючих рішень забезпечить

створення конкурентоспроможного продукту, що відповідає сучасним вимогам ринку туристичних послуг.

Таблиця 1.1 – Порівняльна таблиця характеристик аналогів платформ

Характеристика веборієнтованих інформаційних систем планування туристичних подорожей	**TripAdvisor**	**Roadtrippers**	**Wanderlog**	**Власна розробка**
Сучасний дизайн	+	+	+	+
Легкий інтерфейс і навігація	+	+	-	+
Відображення з різних пристроїв	+	+	+	+
Створення подорожі	+	+	+	+
Інформація про бронювання	+	+	+	+
Оптимізація маршруту по дням	-	-	+	+
Розрахунок загального бюджету	-	-	+	+
Створення списку речей	-	-	-	+
Можливість користуватися навігацією на карті під час прогулянки	-	+	+	+
Широка орієнтованість додатку на різні країни	+	-	+	+

Джерело: зроблено автором

Аналізуючи всі загальні характеристики вище описаних додатків для планування подорожей, буде розроблено власну веборієнтовану інформаційну систему, яка враховуватиме переваги та усуватиме недоліки існуючих рішень.

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ ВЕБОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ТУРИСТИЧНИХ ПОДОРОЖЕЙ

2.1Мета та задачі дослідження

В умовах постійного зростання популярності самостійних подорожей та збільшення кількості туристів, що планують поїздки за допомогою веборієнтованих сервісів, виникає необхідність розробки нових технологій, які б забезпечували швидке та зручне планування подорожей. Аналіз предметної області показав, що більшість існуючих інформаційних систем мають певні обмеження у функціональності, зокрема щодо оптимізації маршрутів, розрахунку бюджету та інтеграції з картографічними сервісами.

Метою магістерської роботи є розробка веборієнтованої інформаційної системи планування туристичних подорожей, спрямованої на автоматизацію процесів створення оптимізованих маршрутів, а також надання користувачам комплексного інструментарію для ефективної організації подорожей, включаючи функції навігації, розрахунку витрат та інформацію про бронювання.

Система орієнтована на єдину категорію користувачів – користувача. Це людина, яка реєструється та авторизується у системі, створює власні подорожі і маршрути, використовує функціонал планування та керування бюджетом, а також здійснює інші дії в рамках функціональності системи.

Для кращого розуміння логіки роботи системи та її структури було розроблено деталізовану навігаційну карту, що допомагає краще зрозуміти послідовність користувацьких дій та основні сторінки сайту.

На першому етапі користувач проходить реєстрацію у вебдодатку, після чого отримує токени доступу та переходить на персональну сторінку. Надалі, для доступу до персоналізованого функціоналу, необхідна авторизація на окремій сторінці. У лівій частині вебсайту буде розташоване меню, що відображає всі

основні сторінки системи, а у верхній частині профілю – шапка із заголовком та логотипом. У своєму акаунті користувач зможе додавати, редагувати та видаляти особисті дані. Перехід через меню до сторінки планування подорожі дозволяє створити нову подорож, яка відображатиметься у їх загальному списку. Додатково можливе редагування чи видалення створених подорожей. Перейшовши на сторінку окремої подорожі, користувачу пропонується заповнити всі бажані локації для відвідування, та відведений запланований і фактичний бюджет на кожну з цих локацій за необхідності. Після внесення даних можна автоматично розрахувати оптимальні маршрути на кожен день, які відображатимуться у вигляді списку днів із локаціями. На правій панелі відображатиметься інтерактивна карта з можливістю розгорнути її на весь екран. Сторінка бронювання дає можливість переглянути готелі та здійснити бронювання, яке в результаті буде відображене у створеній подорожі. На сторінці списку речей можна додавати елементи, позначати їх статус і додавати примітки. Остання сторінка системи – це розрахунок загального бюджету з відображенням витрат за категоріями, запланованих і фактичних витрат. Також передбачено можливість налаштування інтерфейсу, включаючи зміну теми оформлення.

Отже, розроблювана веборієнтована інформаційна система планування туристичних подорожей повинна відповідати наступним функціональним вимогам:

–можливість зареєструватися та авторизуватися в системі;

–можливість змінити та видалити персональні дані в системі;

–можливість створення нової подорожі з основною інформацією у вигляді дат початку та кінця подорожі, її назви та опису, а також з урахуванням функціоналу редагування та видалення даних;

–можливість додавання, редагування та видалення додаткової інформації про подорож у вигляді основних локацій та запланованого на них бюджету;

–можливість побудови оптимальних маршрутів на кожен день подорожі, враховуючи час;

–можливість створення, додавання та видалення інформації про бронювання до подорожі;

–можливість створення списку речей подорожі з функціоналом редагування та видалення як самого списку, так і його елементів;

–можливість розрахунку бюджету всієї подорожі, з урахуванням аналізу категорій запланованих і фактичних витрат;

Крім цього, система повинна бути доступна через веббраузер та оптимізована для роботи на різних пристроях (комп’ютери, планшети, смартфони). Інтерфейс має бути інтуїтивно зрозумілим та відповідати сучасним вимогам до вебдизайну. Для використання системи користувачам достатньо мати базові навички роботи з веббраузером та інтернетом.

Розроблювана система має забезпечувати високу продуктивність та надійність роботи, а також відповідати сучасним вимогам до безпеки вебдодатків. Особлива увага приділяється захисту персональних даних користувачів.

2.2Методи дослідження

Для забезпечення ефективного планування маршрутів у вебдодатку необхідно застосувати методи, що дозволять оптимально розподіляти локації на кожен день подорожі. У цьому контексті був обраний метод кластеризації, який використовує принцип групування локацій у кластери, що відповідають реалістичним і зручним маршрутам для одного дня. Такий підхід дозволяє зменшити час, витрачений на переміщення між точками маршруту.

Методи кластеризації широко застосовуються для виявлення структури даних та їх групування на основі метрик близькості у багатовимірному просторі. Серед найбільш ефективних підходів до кластеризації можна виділити методи, що розглядаються нижче.

Метод k-середніх (K-means) – це один із найпоширеніших методів кластеризації. Алгоритм починає з випадкового вибору центроїдів для k кластерів і ітераційно коригує їхнє положення до моменту, коли кожна точка даних буде належати кластеру з найближчим центроїдом. Алгоритм намагається мінімізувати суму квадратів відстаней між точками і центроїдами їх кластерів. Попри простоту реалізації, метод добре працює для великих обсягів даних, але може бути чутливим до початкового вибору центроїдів та кількості кластерів [12].

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) – метод, що базується на щільності точок у просторі. DBSCAN добре визначає кластери будь-якої форми, ігноруючи умовний шум (точки, що не належать до жодного кластеру). Алгоритм працює на основі двох параметрів – мінімальної кількості точок у кластері та радіусу околу, в якому повинна бути ця кількість точок. Перевага DBSCAN у здатності працювати з шумовими даними та виявляти довільні за формою кластери [13].

Ієрархічна кластеризація – підхід, що передбачає побудову дерева кластерів (дендрограми). Алгоритм може працювати у вигляді агломеративного (знизу вверх) або дивізивного (зверху вниз) процесу. На кожному кроці об’єднуються або розділяються кластери на основі подібності. Цей метод зручний для аналізу різних рівнів кластеризації, однак може бути обчислювально дорогим для великих наборів даних [14].

Для вирішення поставленої задачі необхідно врахувати специфічні обмеження щодо розміру кластерів та/або кількості точок у них, зважаючи на фізичні можливості обходу такого набору точок людиною. З цією метою було розроблено модифікацію алгоритму K-means, що передбачає імплементацію додаткових обмежень на площу кожного кластера, забезпечуючи таким чином формування оптимальних за розміром груп точок.

На початку роботи алгоритму створюється перший кластер, центр якого (формула 2.1) визначається випадковою вибіркою початкової точки з множини вхідних елементів X.

$$\mu_1 = x_{\text{rand}}, \quad x_{\text{rand}} \in X, \tag{2.1}$$

де μ_1 – це центр першого кластеру, що вибирається випадково з множини елементів X. x_{rand} – випадково вибрана точка з набору елементів X.

Для кожного елементу $x \in X$ обчислюється відстань до центрів усіх існуючих кластерів (формула 2.2):

$$d(x, \mu_i) = \sqrt{\sum_{j=1}^{n} \left(x_j - \mu_{i,j}\right)^2}, \quad i = 1, \dots, k, \tag{2.2}$$

де $d(x, \mu_i)$ – відстань між точкою x та центром кластеру μ_i;

x_j – компоненти координат точки x;

$\mu_{i,j}$ – компоненти координат центру μ_i кластеру;

k – кількість кластерів.

Елемент x додається до найближчого кластеру, якщо його відстань до центру кластеру менша за заданий поріг thr або дорівнює нулю. Якщо умова не виконується, створюється новий кластер, де x стає його центром.

Додавання точки у існуючий кластер (формула 2.3):

$$C_i \leftarrow C_i \cup \{x: \min_{i=1,\dots,k} d(x, \mu_i) \leq \text{thr}\} \tag{2.3}$$

Створення нового кластеру (формула 2.4):

$$\mu_{k+1} = x, \quad C_{k+1} = \{x\}, \tag{2.4}$$

де C_i – множина точок, що належать кластеру i, де відстань від кожної точки x до центру μ_i не перевищує порогу thr;

μ_{k+1} – новий центр створеного кластеру, що дорівнює точці x, якщо точка не підходить до жодного існуючого кластеру;

C_{k+1} – новий кластер, що містить лише точку x.

Після розподілу всіх точок обчислюються нові центри для кожного кластеру. Центр кластеру (формула 2.5) обчислюється як середнє арифметичне координат точок, які до нього належать.

$$\mu_i^{\text{new}} = \frac{1}{|C_i|} \sum_{x \in C_i} x\,, \quad i = 1, \dots, k, \tag{2.5}$$

де μ_i^{new} – новий центр кластеру i, який обчислюється як середнє арифметичне координат усіх точок, що належать цьому кластеру C_i;

$|C_i|$ – кількість точок у кластері C_i;

x – точка, яка належить кластеру C_i.

Алгоритм перевіряє, чи змінилися центри кластерів порівняно з попередньою ітерацією. Якщо зміни є, алгоритм продовжує виконання. В іншому випадку, він завершується (формула 2.6).

$$\mu_i^{\text{new}} = \mu_i^{\text{old}}, \quad i = 1, \dots, k, \tag{2.6}$$

де μ_i^{new} – новий центр кластеру i;

μ_i^{old} – попередній центр кластеру i;

Алгоритм завершується, коли жодна точка не змінює свій кластер, а зміни в центрах кластерів стають незначними.

Алгоритм ефективно формує кластери із заданим порогом на відстань між точкою та центром кластеру, забезпечуючи рівність максимальної площі кластеру.

Для побудови оптимальних маршрутів у визначених кластерах система використовує функціонал MapBox API [15], який надає потужний

інструментарій для маршрутизації через інтеграцію з Directions API [16] та Matrix API [17].

Directions API забезпечує можливість побудови оптимальних маршрутів з урахуванням різноманітних параметрів, таких як тип транспорту, обмеження руху, час подорожі та відстань. Matrix API, у свою чергу, дозволяє ефективно обчислювати матриці відстаней та часу подорожі між множиною точок, що є критично важливим для оптимізації маршрутів при обході кластерів.

Такий підхід забезпечує ефективну інтеграцію можливостей MapBox API з розробленою системою кластеризації для формування оптимальних маршрутів обходу точок.

3 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ВЕБОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ТУРИСТИЧНИХ ПОДОРОЖЕЙ

3.1Структурно-функціональне моделювання

Моделювання відіграє ключову роль у розробці інформаційних систем, оскільки дозволяє чітко спланувати етапи проєкту, оптимізувати структуру системи та проаналізувати функції та їх взаємозв’язки. Це підвищує ефективність управління ресурсами і якість результату.

Контекстна діаграма моделі IDEF0 [18] є потужним інструментом для візуалізації бізнес-процесів. Вона структуровано інтегрує вхідні та вихідні дані, дозволяючи легко описувати функції системи. Використання IDEF0 підвищує адаптивність системи, що допомагає досягти успішності виконання проєктів. Контекстна діаграма IDEF0 не лише дозволяє зрозуміти логіку функціонування бізнес-процесів, а й сприяє їх оптимізації. Вона дає змогу чітко ідентифікувати взаємозв’язки між елементами, що в свою чергу допомагає у плануванні та реалізації проєктів, забезпечуючи їхню ефективність та адаптивність до змін.

У контексті розробки веборієнтованої інформаційної системи для планування туристичних подорожей, діаграми IDEF0 можуть бути використані для опису взаємодії користувача із системою та визначення основних функціональних модулів. Це дозволить структуровано відобразити процеси, такі як створення та планування подорожей, а також інші супутні етапи. Контекстна діаграма моделі IDEF0, що ілюструє процес виконання планування туристичних подорожей, наведена на рисунку 3.1.

Рисунок 3.1 – Контекстна діаграма бізнес-процесів у нотації IDEF0 з точки зору користувача

Джерело: зроблено автором

На основі представленої контекстної діаграми бізнес-процесів у нотації IDEF0 з точки зору користувача можна здійснити детальний аналіз функціонування інформаційної системи планування туристичних подорожей. Система має чітко визначену структуру, яка охоплює вхідні дані, методи управління, що встановлюють правила виконання операцій, механізми для реалізації робочих процесів, а також вихідні дані.

До вхідних потоків даних системи належать персональні дані користувачів, їх облікові дані у вигляді логіну та паролю, дати початку та кінця подорожі, основні дані про локації для відвідування з попереднім запланованим бюджетом для них, а також перелік необхідних речей для подорожі. Ці дані формують базовий інформаційний фундамент для подальшого планування.

Керуючі елементи системи включають інструкцію з використання вебдодатку, географічні дані для надання інформації про визначні пам’ятки та інші локації на карті, а також дані про готелі, що дозволяють здійснювати бронювання через систему.

Механізми реалізації робочих процесів включають вебдодаток «TripPlanner» як ключовий інструмент взаємодії, користувача системи як активного учасника процесу, базу даних для надійного зберігання інформації, а також технічне забезпечення, яке підтримує стабільне функціонування системи.

Результатами функціонування системи є сформовані дані подорожі, оптимізовані щоденні маршрути, інформація про бронювання, детальний список необхідних речей та розрахований бюджет подорожі. Така структуризація вихідних даних дозволяє реалізувати комплексний підхід до планування подорожі, що підвищує зручність і ефективність користування системою.

Для досягнення глибшого розуміння функціональності системи важливо провести декомпозицію основного процесу (рис. 3.2), що дозволить детальніше проаналізувати кожен підпроцес та оптимізувати їх взаємодію. Такий аналіз сприятиме підвищенню ефективності системи та покращенню якості планування туристичних подорожей відповідно до індивідуальних потреб користувачів.

Виходячи з представленої декомпозиції IDEF0-діаграми, можна провести аналіз основних функціональних складових інформаційної системи для планування туристичних подорожей. Дана декомпозиція відображає чотири основні процеси, кожен з яких має свої вхідні та вихідні дані, а також методи управління та механізми реалізації.

Перший функціональний блок «Реєстрація» (A1) відповідає за первинну обробку персональних даних користувачів та створення облікового запису в системі. На вхід цього блоку надходять персональні дані, а результатом його роботи є створений акаунт у вебдодатку. Керуючим елементом є інструкція по використанню вебдодатку, а механізмами реалізації є вебдодаток «TripPlanner», користувач, база даних та технічне забезпечення.

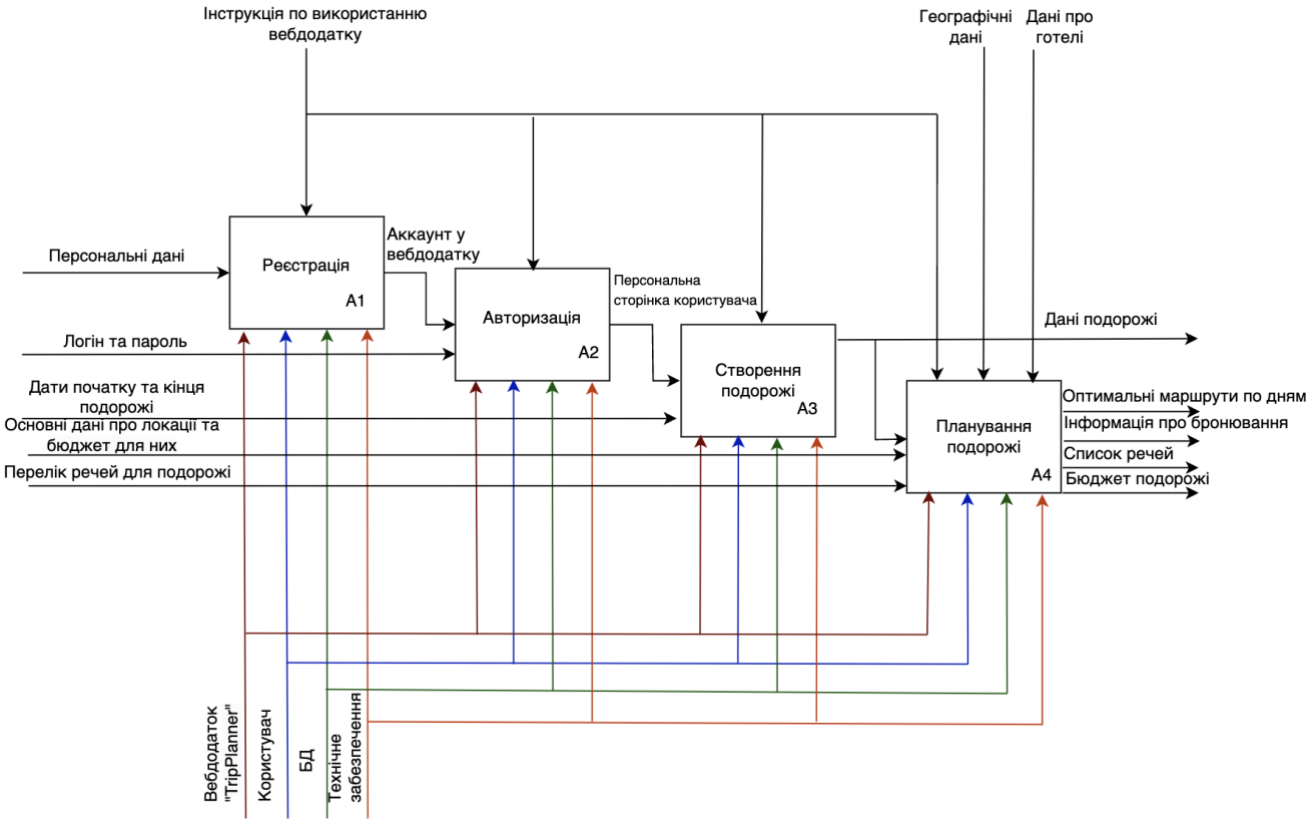


Рисунок 3.2 – Перший рівень декомпозиції IDEF0-діаграми з точки зору користувача

Джерело: зроблено автором

Наступний блок «Авторизація» (A2) забезпечує процес автентифікації користувачів у системі. Вхідними даними для цього блоку є логін та пароль користувача, а результатом – доступ до персональної сторінки. Цей блок використовує ті ж механізми реалізації, що й попередній та керується інструкцією по використанню вебдодатку.

Блок «Створення подорожі» (A3) обробляє основні параметри майбутньої подорожі. На вхід надходять дати початку та кінця подорожі. Результатом роботи цього блоку є сформовані базові дані подорожі. Керуючим елементом також є інструкція по використанню вебдодатку.

Завершальний блок «Планування подорожі» (A4) є найбільш комплексним та відповідає за формування детального плану подорожі. Цей блок обробляє дані про основні локації подорожі та запланований на них бюджет, а також додаткову інформацію про речі, які необхідно зібрати для подорожі. На виході формуються

оптимальні маршрути по днях, інформація про бронювання, список речей та загальний бюджет подорожі. Керуючими елементами є інструкція по використанню вебдодатку, географічні дані та дані про готелі.

Враховуючи складність та багатофункціональність блоку «Планування подорожі» (A4), доцільним є проведення подальшої декомпозиції саме цього процесу. Це дозволить детально проаналізувати підпроцеси додавання додаткої інформації стосовно подорожі, побудови маршрутів, здійснення бронювань, управління списком речей та розрахунку бюджету. Другий рівень декомпозиції, що відображає взаємозв'язки між підпроцесами та оптимізує їхню взаємодію, представлений у діаграмі IDEF0 з точки зору користувача на рисунку 3.3.

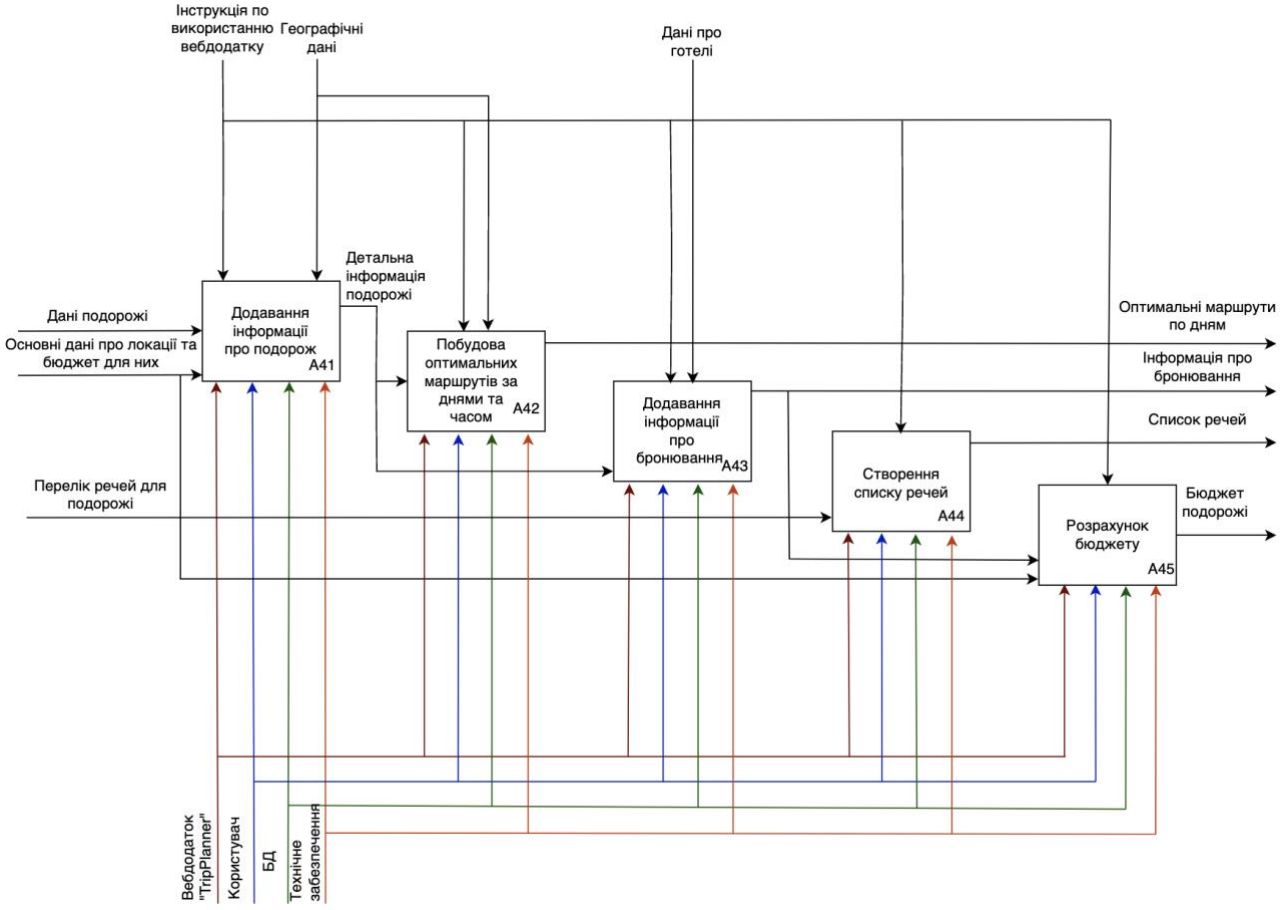


Рисунок 3.3 – Другий рівень декомпозиції IDEF0-діаграми з точки зору користувача

Джерело: зроблено автором

Другий рівень декомпозиції IDEF0-діаграми детально розкриває процес «Планування подорожі» (A4), який є ключовим етапом у функціонуванні всієї інформаційної системи планування туристичних поїздок. Для кожного блоку механізмами реалізації є вебдодаток «TripPlanner», користувач, база даних та технічне забезпечення. Крім цього, кожен блок містить інструкцію про використанню вебдодатку як метод управління.

Першим блоком декомпозиції другого рівня є «Додавання інформації про подорож» (A41). На вході до цього блоку надходять базові дані про подорож, а також основні локації та запланований на них бюджет. Додатковим методом управління, окрім інструкції, також є географічні дані, що будуть надавати всю необхідну інформацію для представлення пам’яток та інших об’єктів на карті.

Блок «Побудова оптимальних маршрутів за днями та часом» (A42) відповідає за генерацію оптимізованих щоденних маршрутів на основі заданих локацій для всієї подорожі. На вхід подається детальна інформація про подорож, яка включає в себе локації. Використовуючи ці вхідні дані і спеціальні алгоритми кластеризації та оптимізації, блок формує оптимальні маршрути на кожен день для користувача. Додатково, одним з методів управління є географічні дані.

Наступний блок «Додавання інформації про бронювання» (A43) обробляє та систематизує дані про всі бронювання, пов’язані з подорожжю. У ролі вхідної інформації тут виступають основні дані подорожі, а результатом роботи стає інформація про бронювання, тобто деталізований перелік всіх резервацій, зроблених для учасника поїздки. Ще одним методом управління для цього блоку є дані про готелі, в якому міститься загальний перелік готелей та їх опису.

Блок «Створення списку речей» (A44) реалізує формування персоналізованого переліку необхідних речей для подорожі. На вхід йому подаються речі для подорожі, а на виході отримаємо готовий список речей.

Завершальний блок «Розрахунок бюджету» (A45) відповідає за комплексний аналіз та структурування всіх фінансових аспектів поїздки. На основі попередньо зібраних даних про бронювання та заплановані витрати

відповідно до локацій подорожі, цей блок на виході формує загальний бюджет, враховуючи різні категорії витрат.

Декомпозиція процесу «Планування подорожі» на п’ять основних підпроцесів дозволяє комплексно дослідити всі ключові етапи формування повноцінного плану туристичної поїздки.

Отже, побудована IDEF0-модель дає змогу глибоко оцінити можливості розробленого програмного забезпечення та виявити потенціал для його подальшої оптимізації. Такий підхід до структурно-функціонального моделювання забезпечує міцне підґрунтя для реалізації високоефективного програмного рішення, здатного задовольнити потреби користувачів у плануванні та організації туристичних поїздок.

3.2UML-моделювання

3.2.1 Діаграма варіантів використання

Розуміння взаємодії користувачів із системою є ключовим елементом ефективного проєктування. Моделювання діаграм варіантів використання (Use Case діаграм) дозволяє чітко візуалізувати зв’язки між акторами та функціональними можливостями системи, що слугує важливим інструментом для аналізу вимог та проєктування [19]. Такі діаграми допомагають виявити основні сценарії використання системи, розкривають, яким чином вона задовольняє потреби користувачів.

Дана діаграма моделює функціональні вимоги з точки зору користувача та включає ключові компоненти:

–актор – це зовнішня сутність, що контактує з системою;

–прецедент – це дія, яку виконує система;

–система та зв’язки між ними.

Також існує кілька типів зв’язків між варіантами використання:

–асоціація – це зв’язок актора з прецедентом;

–розширення – це додаткові необов’язкові дії, які можуть бути активовані за певних умов;

–включення – це обов’язкові допоміжні дії, які є частиною основного сценарію;

–генералізація – це спільні функції в ієрархії прецедентів, що дозволяє ієрархічно організувати різні варіанти використання [19].

Таким чином, Use Case діаграми є невід’ємною частиною процесу проєктування систем, забезпечуючи глибоке розуміння вимог та сприяючи створенню функціональних, орієнтованих на користувача рішень.

Отже, діаграму варіантів використання для веборієнтованої інформаційної системи планування туристичних подорожей було представлено на рисунку 3.4.

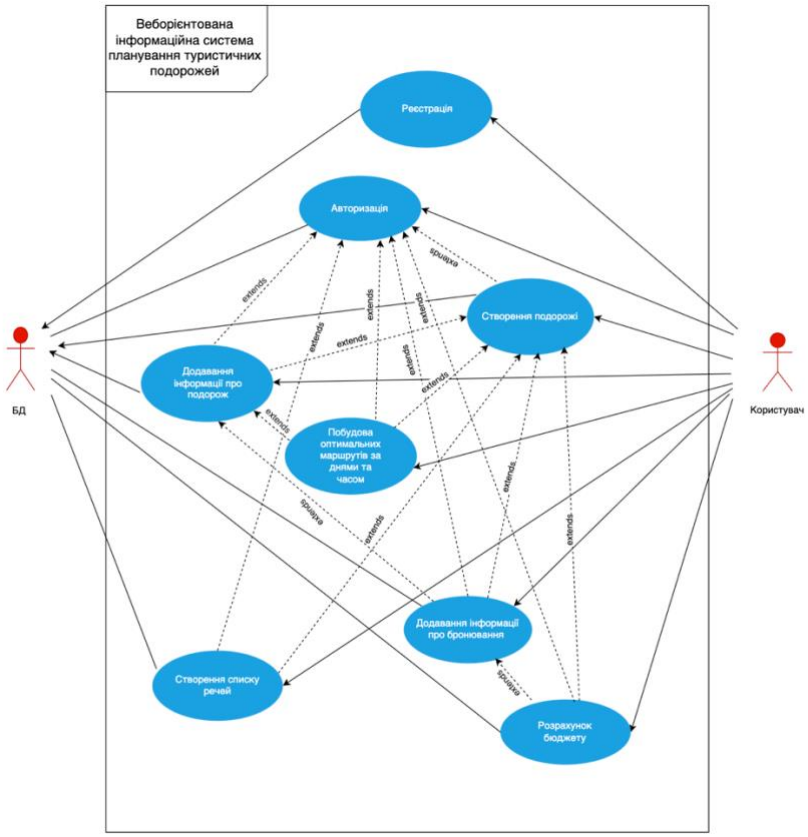


Рисунок 3.4 – Діаграма варіантів використання

Джерело: зроблено автором

Основні актори:

–Користувач – головний суб’єкт, який взаємодіє із всіма сценаріями системи.

–База даних (БД) – відповідає за зберігання, обробку та управління даними, необхідними для роботи системи і також за взаємодію з усіма варіантами використання.

Основні варіанти використання:

–Реєстрація – початковий етап для будь-якого користувача системи, що дозволяє створити обліковий запис і забезпечує персоналізований доступ до функцій платформи. Після успішної реєстрації користувач може налаштувати свій профіль та розпочати роботу з системою.

–Авторизація – процес входу до системи, який здійснюється на основі облікових даних (логін та пароль), що забезпечує захищений доступ до всіх персоналізованих функцій та конфіденційних даних користувача.

–Створення подорожі – користувач може розпочати створення нової подорожі, вказуючи назву, дати початку та завершення.

–Додавання інформації про подорож – користувач може додавати й оновлювати інформацію про заплановані локації, бюджет на кожну локацію, а також окремі заходи, що сприяють зручності планування.

–Побудова оптимальних маршрутів – на основі введених даних система автоматично генерує оптимальні маршрути по дням з урахуванням часу і місць призначення.

–Додавання інформації про бронювання – система дозволяє користувачеві зберігати інформацію про бронювання.

–Розрахунок бюджету – користувач може виконати розрахунок витрат і переглянути їх аналіз за окремими категоріями, включаючи бронювання та виділені кошти на певну локацію, наприклад, ресторан.

Створення списку речей – функціональність, що дозволяє користувачеві сформувати перелік речей, необхідних для подорожі.

3.2.2 Діаграми послідовності

Діаграми послідовності (Sequence Diagrams) в нотації UML є ефективним інструментом моделювання для відображення взаємодії між об’єктами чи компонентами системи в контексті певного сценарію або процесу. Цей тип діаграм дозволяє наглядно представити хронологічну послідовність обміну повідомленнями між об’єктами, а також етапи виконання операцій. Кожен елемент діаграми послідовності відображає конкретний момент часу, при цьому передача повідомлень між об’єктами здійснюється горизонтально [20].

Для прикладу оберемо та розглянемо варіант використання «Побудова оптимальних маршрутів за днями та часом», діаграма якого представлена на рисунку 3.5.

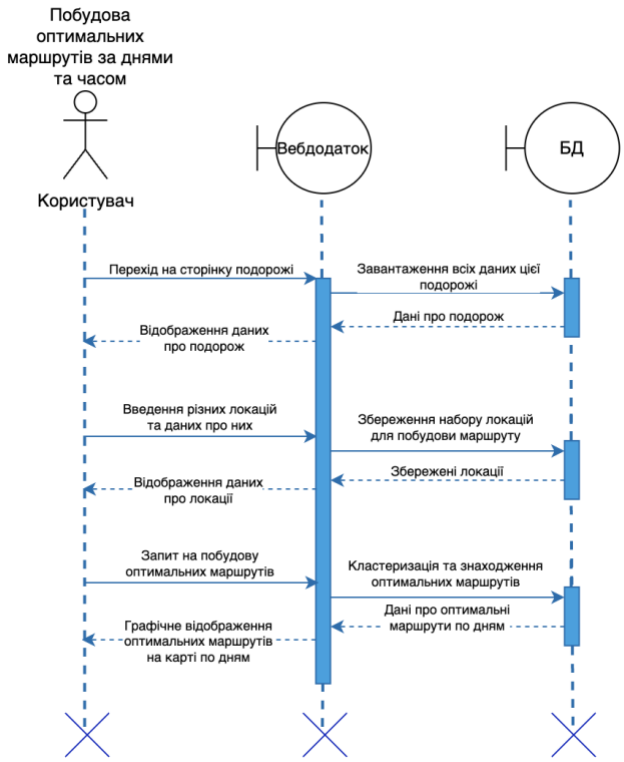


Рисунок 3.5 – Діаграма послідовності для ВВ «Побудова оптимальних маршрутів за днями та часом»

Джерело: зроблено автором

Дана діаграма послідовності ілюструє процес генерації оптимальних маршрутів для туристичних подорожей у веборієнтованій інформаційній

системі. Ця діаграма демонструє взаємодію між основними сутностями системи: користувачем, вебдодатком та базою даних.

Процес починається з ініціалізації користувачем сторінки планування подорожі, що призводить до завантаження відповідної інформації з бази даних. Далі користувач вводить дані про різні локації, які також зберігаються у базі даних. Потім вебдодаток здійснює запит на кластеризацію та пошук оптимальних маршрутів між локаціями. Результати цього аналітичного процесу візуалізуються у вигляді графічної карти, що відображає оптимальні маршрути для кожного дня подорожі.

Таким чином, представлена діаграма послідовності надає детальне уявлення про взаємодію основних компонентів системи, а саме: процес ініціалізації, отримання даних, обробки інформації та візуалізації результатів. Ця схема демонструє логічну структуру та функціональність веборієнтованого рішення для планування оптимальних маршрутів туристичних подорожей.

3.2.3 Діаграми діяльності

Діаграми діяльності (Activity Diagrams) є одним із ключових інструментів в нотації UML, що використовуються для графічного моделювання послідовності виконання дій чи процесів у певній системі. Вони дозволяють наочно відобразити, як різні кроки, задачі або дії пов’язані між собою, а також яким чином потоки дій змінюються в процесі досягнення певної мети. Кожен елемент діаграми діяльності може представляти окрему операцію, етап або умову, а зв’язки між ними ілюструють логіку виконання [21].

До розгляду візьмемо знову ВВ «Побудова оптимальних маршрутів за днями та часом», на основі якого побудуємо детальну діаграму діяльності, щоб визначити всі проміжні етапи цього процесу. Результат побудованої діаграми представлено на рисунку 3.6.

Рисунок 3.6 – Діаграма діяльності для ВВ «Побудова оптимальних маршрутів за днями та часом»

Джерело: зроблено автором

Представлена на рисунку діаграма діяльності демонструє основні етапи та потоки управління в процесі генерації оптимальних маршрутів для туристичних подорожей у веб-орієнтованій інформаційній системі.

Процес розпочинається з відкриття вебдодатку, після чого користувач переходить до сторінки подорожі, що призводить до завантаження відповідних даних з БД. Далі користувач отримує доступ до основних даних подорожі та вводить додаткову інформацію про заплановані для відвідування локації, які також зберігається у базі даних. Після цього вебдодаток надає нам збережені дані про всі наявні локації подорожі та виконує процес кластеризації та пошуку оптимальних маршрутів на кожен день, на основі отриманих даних. Результати виконання алгоритмів графічно відображаються на карті, що наочно презентує оптимальні маршрути для кожного дня подорожі.

3.3Проєктування моделі бази даних

Ефективна організація структури зберігання даних є одним із ключових етапів розробки інформаційної системи. Цей процес включає в себе проєктування бази даних, яке дозволяє сформувати концептуальну модель, що встановлює логічні зв’язки між сутностями та визначає необхідні атрибути для побудови таблиць.

На початковому етапі проєктування було проаналізовано предметну область та визначено основні сутності, що мають бути відображені в інформаційній системі. Результатом даного аналізу стала побудова концептуальної моделі, яка візуалізує взаємозв’язки між ключовими сутностями системи.

В результаті проведеного проєктування була розроблена схема бази даних, яка відображає взаємодію сутностей. Дану модель можна побачити на рисунку 3.7.

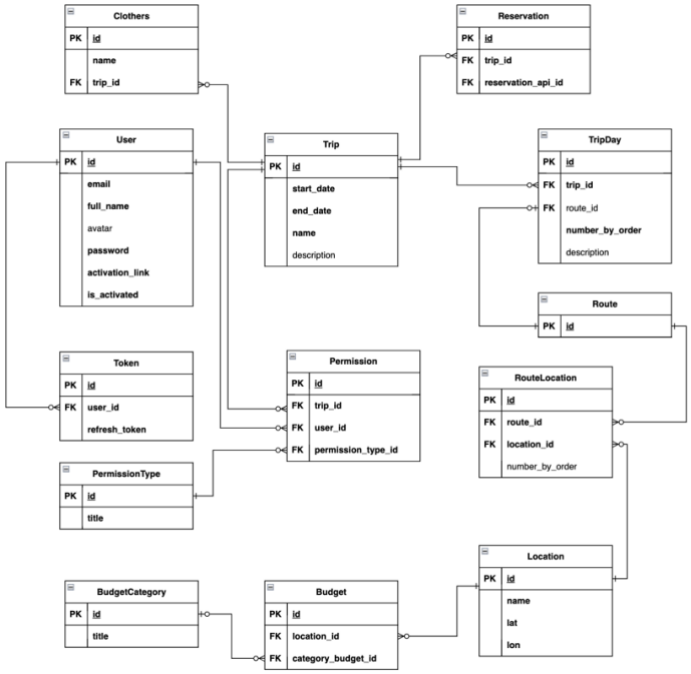


Рисунок 3.7 – Схема бази даних

Джерело: зроблено автором

До переліку сутностей моделі бази даних наданої діаграми входять такі елементи:

User – користувач системи з основними атрибутами, такими як email, повне ім’я, аватар, пароль, посилання на активацію персональної сторінки та статус активації.

Trip – подорож з атрибутами початкової та кінцевої точок, назви та опису.

Clothers – список речей для подорожі, прив’язаний до конкретної подорожі.

Reservation – таблиця для бронювання, пов’язана з подорожжю та конкретним ID з API бронювання.

TripDay – день подорожі, який містить зв’язок з подорожжю та маршрутом, а також порядковий номер і опис.

Route – маршрут, що об’єднує кілька точок на карті для подорожі.

RouteLocation – таблиця для опису належності локацій до маршруту

Location – локація з атрибутами назви, широти та довготи.

Permission – дозвіл на доступ до подорожі для різних користувачів, містить зв’язок з подорожжю, користувачем та статусом дозволу, таким як читання або редагування.

PermissionType – тип дозволу з атрибутом назви.

Token – токени для авторизації користувача з атрибутами ідентифікатора та токена оновлення.

Budget – бюджет подорожі, прив’язаний до локації та категорії бюджету.

BudgetCategory – категорія бюджету з атрибутом назви.

Зв’язки між сутностями:

User та Token: Користувач може мати кілька токенів для авторизації, а токен повинен бути пов’язаним з одним користувачем.

Trip та Clothers: У подорож можна взяти кілька речей, кожна річ повинна бути пов’язана з конкретною подорожжю.

Trip та Reservation: Для подорожі можна здійснити кілька бронювань, кожне з яких повинно належити до певної подорожі.

Trip та TripDay: Подорож може складатися з декількох днів, кожен з яких повинен належати певній подорожі.

TripDay та Route: День подорожі може мати один маршрут, а сам маршрут повинен бути пов’язаним тільки з одним днем подорожі.

Route та Location: Маршрут може складається з кількох локацій, які можуть належати кільком.

User та Permission: Користувач може мати декілька дозволів, а сам дозвіл повинен належати лише одному користувачу.

Trip та Permission: Можна видати кілька дозволів на доступ до подорожі, кожен дозвіл повинен бути на одну подорож.

PermissionType та Permission: Тип дозволу може бути у кількох дозволах, кожен з яких повинен містити тип.

Location та Budget: Локація може мати декілька варіантів бюджету, а сам конкретний бюджет повинен належати тільки одній локації.

BudgetCategory та Budget: Категорія бюджету може відноситися до декількох різних бюджетів, а сам конкретний бюджет може мати тільки одну категорію.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ТУРИСТИЧНИХ ПОДОРОЖЕЙ

4.1 Архітектура програмного додатку

Для розробки додатка було обрано архітектурну модель типу «клієнт-сервер», яка є загальноприйнятим стандартом у створенні сучасних розподілених систем.

Архітектура клієнт-серверної взаємодії репрезентує складну парадигму розподілених інформаційних систем, що базується на диференційованій та модульній комунікативній інфраструктурі. Її концептуальна модель складається з трьох критичних інтегральних елементів:

–сервери надають сервіси та ресурси для клієнтів, працюючи автономно;

–клієнти запитують і використовують сервіси серверів, діючи незалежно;

–мережа забезпечує зв’язок між клієнтами та серверами.

Архітектура забезпечує масштабованість і гнучкість, дозволяючи серверам обслуговувати численні запити від різних клієнтів одночасно. Клієнти, у свою чергу, можуть звертатися до будь-якого сервера за потреби, не залежачи від конкретного. Вони мають доступ до даних про сервери, проте не взаємодіють між собою, що гарантує їх автономність [22].

Структурна архітектура веборієнтованої інформаційної системи планування подорожей базується на HTTP-протоколі та передбачає багаторівневу інтеграцію компонентів з чітким розподілом функціональних обов’язків, що забезпечує ефективну клієнт-серверну взаємодію. Схема даної архітектури додатку представлена на рисунку 4.1.

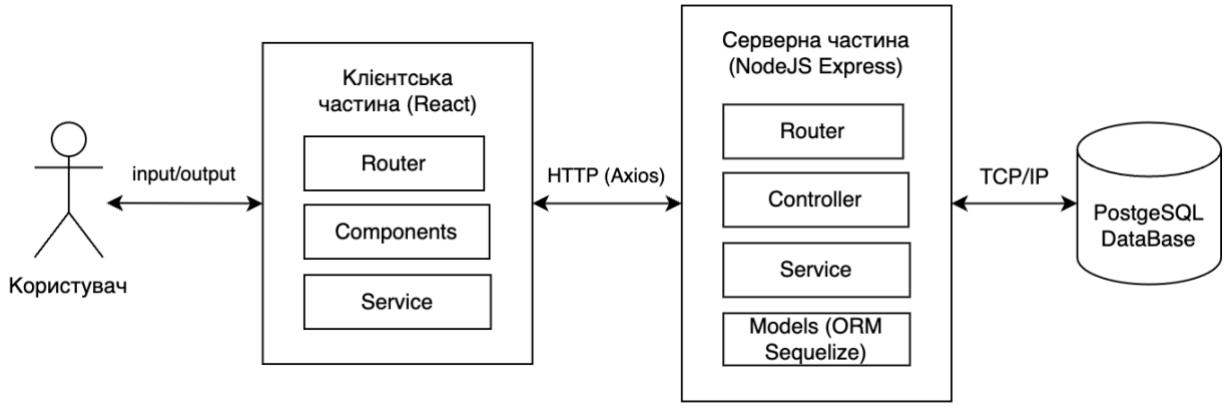


Рисунок 4.1 – Архітектура вебдодатку веборієнтованої інформаційної системи планування подорожей

Джерело: зроблено автором

Архітектура клієнтської частини веборієнтованої інформаційної системи планування подорожей базується на бібліотеці React, що репрезентує сучасний підхід до розробки інтерактивних інтерфейсів з оптимізованою структурою рендерингу [23]. Використання віртуального DOM забезпечує високоефективне оновлення контенту шляхом часткової трансформації компонентів, що суттєво підвищує продуктивність веб-додатку [24]. Системна архітектура клієнтського рівня побудована на функціональних компонентах, які реалізують принципи SOLID та забезпечують модульність і розширюваність програмного забезпечення [25]. Сучасна методологія розробки React передбачає впровадження хуків, таких як useState, або useEffect для централізованого управління станом та життєвими циклами компонентів, що дозволяє створювати динамічні та адаптивні інтерфейсні рішення [26].

Навігаційна підсистема реалізована за допомогою бібліотеки react-router-dom, яка забезпечує багаторівневу маршрутизацію з підтримкою захищених маршрутів та динамічного контролю доступу до різних компонентів системи. Комунікація з серверним API здійснюється через сервісні модулі з використанням бібліотеки Axios, що уможливлює ефективну реалізацію CRUD-операцій та забезпечує уніфікований механізм HTTP-взаємодії [27].

Серверна частина реалізована за допомогою Node.js, що дозволяє побудувати архітектуру з використанням REST API. Основним фреймворком для серверної частини був Express.js, який у поєднанні з Sequelize ORM дозволяє ефективно працювати з реляційною базою даних PostgreSQL, що забезпечує нормалізовану структуру даних, підтримку транзакцій та механізми для забезпечення їх цілісності й узгодженості [28]. Контролери виступають інтерфейсним механізмом обробки HTTP-комунікацій, забезпечуючи первинну маршрутизацію вхідних запитів. Маршрутизаційні механізми, тобто роути системи є координаторами між клієнтськими запитами та відповідними контролерами. Сервісний рівень реалізує низькорівневі операційні завдання, зокрема безпосередню взаємодію з базою даних.

Картографічний сервіс розроблено на базі Mapbox GL JS, що забезпечує високоінтерактивну геопросторову візуалізацію з розширеними можливостями кастомізації. Інтеграція спеціалізованих Mapbox API, а саме: Directions, Matrix та Geocoding, – надає системі потужний інструментарій для геолокаційного аналізу. Directions API відповідає за побудову маршрутів, Matrix API здійснює розрахунок часових інтервалів між географічними точками, а Geocoding API забезпечує конвертацію текстових локацій у географічні координати та навпаки, що значно розширює функціональні можливості геоінформаційного компоненту вебдодатку [29].

4.2Програмна реалізація

4.2.1 Серверна частина

Після визначення функціональних вимог до системи, моделювання та проєктування бази даних, першим кроком було створення серверу за допомогою NodeJS, на якому будуть створені моделі для БД PostgreSQL за допомогою Sequelize ORM. Отже, початково були розроблені моделі майбутніх таблиць та

їх зв’язки. Кожна модель повинна мати власні атрибути, типи та інші ключові параметри, що будуть описувати таблицю в БД. Такий спосіб взаємодії з БД є дуже зручним для її швидкого налаштування та зміни за потреби під час розробки вебдодатку.

Після цього в основному файлі серверу були підключені всі необхідні елементи для його коректної працездатності, а саме: cookieParser для збереження файлів кукі на сайті, cors для налагодження процесу обміну даними між сервером та клієнтом, fileUpload для збереження фотографій на сервері, Middleware, наприклад, такі як ErrorMiddleware для перехвату та обробки всіх помилок і маршрутизація, що визначає за якими саме шляхами будуть надходити запити. Для полегшення роботи з сервером було обрано та підключено також фреймворк Express.

Наступним кроком було розроблено роути, контроллери та сервіси для основних моделей БД, в яких була налаштована вся логіка отримання, обробки та передачі даних, пов’язаних з клієнтською частиною.

Розглядаючи процес обробки даних на сервері, цей шлях можна описати наступним чином, на прикладі розробки функціоналу створення подорожі у вебдодатку. Спочатку за допомогою маршрутизації через клієнтський сервіс надсилаються дані за певним шляхом на сервер, де через роут вже потрапляємо до контроллера. Приклад роуту наведений нижче:

```
tripRoutes.post('/', tripController.createTrip);
```

Далі, перейшовши всередину контроллера, беремо дані з параметрів, файлів та тіла запиту, щоб передати їх на обробку в сервіс. Після цієї обробки можна отримати результат, який буде повертатися у вигляді json об’єкту. Приклад контроллеру для створення подорожі наведений нижче:

```
async pcreateTrip(req, res) {
   try {
     const trip = await TripService.createTrip(req.body, req.files.image);
     res.json(trip);
   } catch (error) {
     res.status(500).json(error);
   }
 }
```

У сервісі виконуються низькорівневі задачі, а саме зв’язок з БД і повертаються отримані результати. Вигляд сервісу для створення подорожі наведено нижче:

```
async createTrip(trip, image) {
  const fileName = FileService.saveFile(image);
  return await Trip.create({ ...trip, image: fileName });
}
```

Також під час процесу обробки запитів оброблюються помилки або перевіряється захищеність маршрутів за допомогою Middleware. При підключенні роуту створення подорожі до всіх роутів з перевіркою на авторизацію це буде мати наступний вигляд:

```
mainRoutes.use('/trip', AuthMiddleware, tripRoutes);
```

Окремо треба відзначити захист даних користувача, який відбувається за допомогою, наприклад, хешування паролів, використовуючи бібліотеку bcrypt. Також сесія користувача налаштована таким чином, щоб зберігати два види токенів – «access» і «refresh». Access токен призначений для короткотермінової авторизації та забезпечує миттєвий доступ до системних ресурсів. Натомість, refresh токен, що зберігається в захищеній базі даних, виступає механізмом відновлення сесії, дозволяючи здійснювати верифікацію без повторної авторизації користувача та підтримувати безперервність роботи системи. Для забезпечення аутентифікації у додатку використовується бібліотека jsonwebtoken, яка реалізує механізм криптографічного захищеного обміну даними на основі JSON Web Token (JWT) технології.

4.2.2 Клієнтська частина

Під час розробки клієнтської частини були створені компоненти інтерфейсу, які розбивались на менш дрібні для повторного використання в багатьох місцях програми одночасно. Для полегшення сприйняття користувачами інтерфейсу було вирішено використовувати загальний стандарт компонентів, тому було обрано бібліотеку MUI, що містить вже базові готові

компоненти, які можна корегувати, відповідно до стилю вебдодатку. Основна структура додатку була створена на основі бібліотеки React з використанням TypeScript для відслідковування більш чіткої поведінки функціоналу.

Для навігації по вебдодатку було використано React Router DOM, що дозволило визначити всі наявні маршрути між сторінками та допомогло у вирішенні таких питань як фільтрація або сортування за допомогою вбудованих хуків.

Також для отримання доступу до даних з серверу, тобто надсилання HTTP-запитів, були прописані сервіси з використанням зручної бібліотеки Axios. Ця бібліотека дозволяє використовувати перехоплювачі відповідей для обробки даних під час відправки запитів або отримання відповідей. Такий функціонал надає змогу автоматично додавати токен до запитів, які вимагають авторизації, або здійснювати їх перевірку та оновлення за необхідності.

Для реалізації функціоналу відображення та управління бронюванням готелів на даному етапі розроблено власний сервер і базу даних, які виконують роль тимчасового рішення. Це дозволяє забезпечити працездатність системи до інтеграції з більш потужними сервісами, такими як Booking.com, використання яких наразі потребує фінансових вкладень.

Реалізація роботи з мапою та її візуальне відображення базується на використанні MapBox API, функціонал якого містить велику кількість потужного інструментарію, наприклад, визначення відстаней між локаціями, побудову маршруту, пошук локації та інші. Для під’єднання до цього API був створений власний токен доступу. Гнучкий функціонал MapBox дозволяє налаштовувати власні стилі карти та додавати туди власну логіку розробки, наприклад, кластеризацію обраних користувачем локацій.

Для розробки кластеризації було розглянуто чимало алгоритмів, серед яких були k-means, DBSCAN та ієрархічна кластеризація. Всі вони показали гарні результати, проте для вирішення поставлених задач було створено модифікацію методу K-means, що передбачає імплементацію додаткових

обмежень на площу кожного кластера, забезпечуючи таким чином формування оптимальних за розміром груп точок.

4.3Приклад використання програмного додатку

Перед початком роботи з вебдодатком користувач повинен пройти процедуру реєстрації або авторизації в системі. Для цього передбачено форму введення даних, яка включає поля для зазначення повного імені, електронної пошти, пароля та завантаження зображення профілю. У процесі заповнення форми здійснюється валідація введених даних, яка відображає повідомлення про помилки в разі їх виникнення. Після успішного введення всіх необхідних даних користувач спостерігає момент завантаження системи та отримує повідомлення про успішну авторизацію. Приклади модальних вікон для реєстрації, входу до системи, а також валідації даних наведено на рисунках 4.2–4.4.

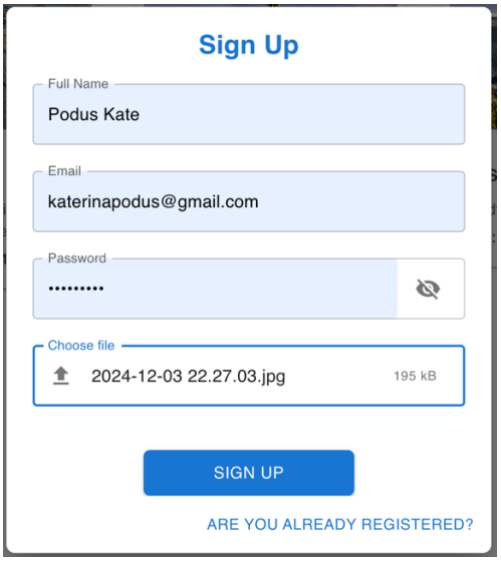


Рисунок 4.2 – Форма реєстрації

Джерело: зроблено автором

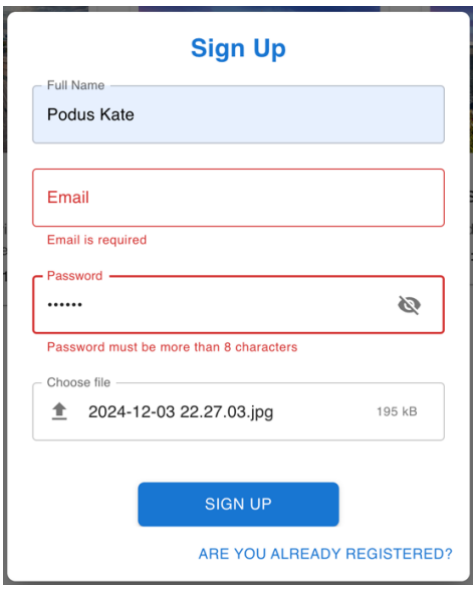


Рисунок 4.3 – Форма реєстрації з прикладом валідації даних

Джерело: зроблено автором

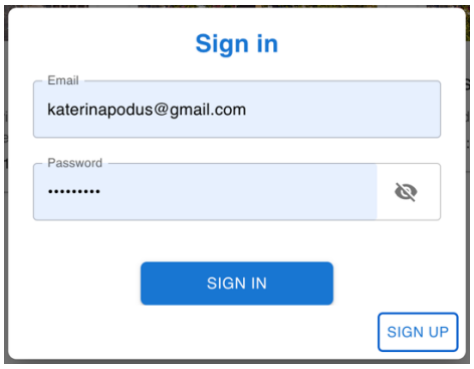


Рисунок 4.4 – Форма логіну в системі

Джерело: зроблено автором

Після авторизації користувач автоматично перенаправляється на сторінку свого профілю, де відображаються основні персональні дані з можливістю їх редагування (рис. 4.5). Якщо у користувача є заброньовані готелі, вони також відображаються на цій сторінці. Зліва розташоване меню навігації, яке містить посилання на сторінки профілю та списку всіх подорожей. У нижній частині бокового меню відображається інформація про активний профіль із можливістю виходу або повторного входу до системи. У верхній частині сайту розміщено логотип із назвою, елементи для розгортання та згортання меню, а також перемикач теми оформлення.

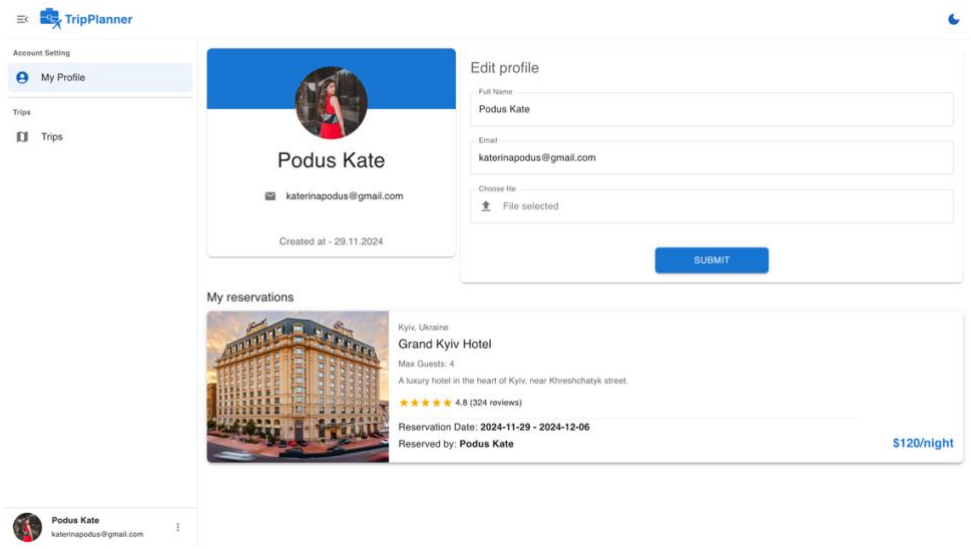


Рисунок 4.5 – Сторінка профілю користувача

Джерело: зроблено автором

На сторінці всіх подорожей користувач має можливість переглядати перелік існуючих подорожей або створювати нові. Для створення нової подорожі передбачено функціонал виклику модального вікна, яке відкривається при натисканні на кнопку з іконкою плюса. Візуальне представлення модального вікна створення подорожі наведено на рисунку 4.6.

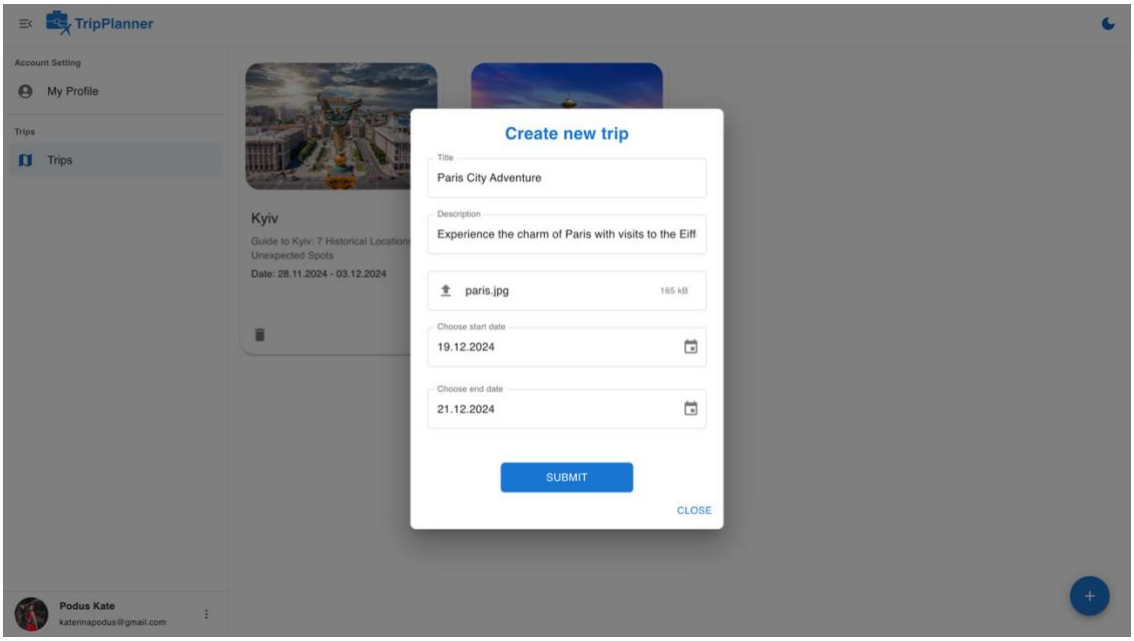


Рисунок 4.6 – Модальне вікно створення подорожі

Джерело: зроблено автором

Також на рисунку 4.7 можна переглянути демонстрацію всіх подорожей на сторінці.

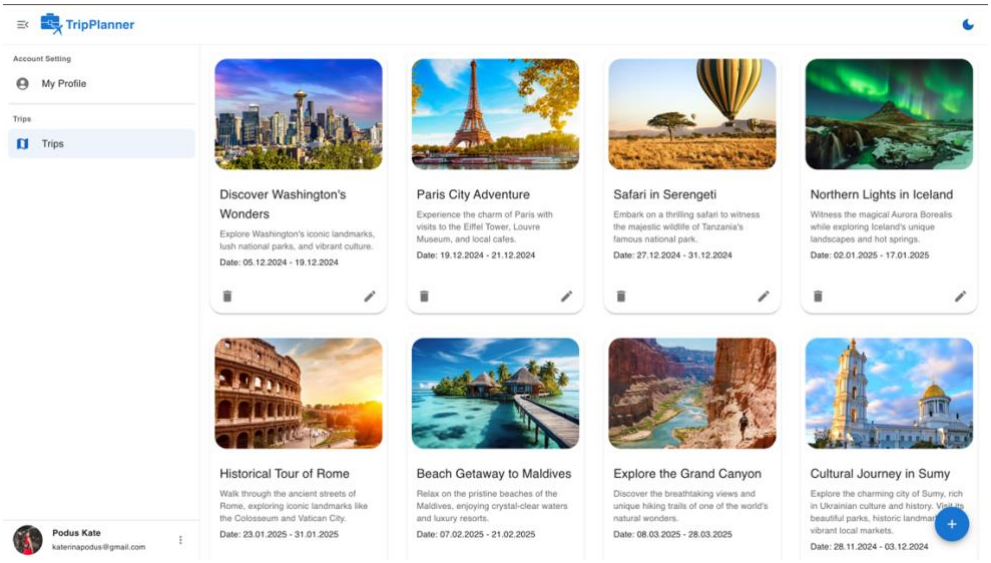


Рисунок 4.7 – Сторінка всіх подорожей

Джерело: зроблено автором

Для видалення подорожі передбачена кнопка видалення на карточці кожної подорожі у вигляді іконки смітнику, по натисканню якої відкривається підтвердження запиту видалення. Приклад наведено на рисунку 4.8.

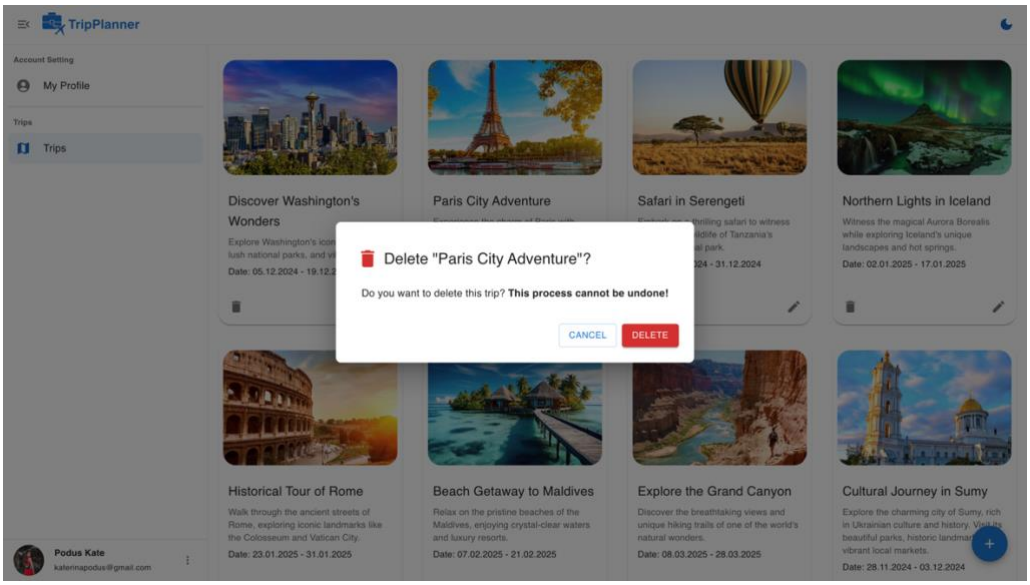


Рисунок 4.8 – Модальне вікно видалення подорожі

Джерело: зроблено автором

Також по натисканню на кнопку з іконкою олівця на карточці подорожі можна відредагувати її дані, це показано на рисунку 4.9.

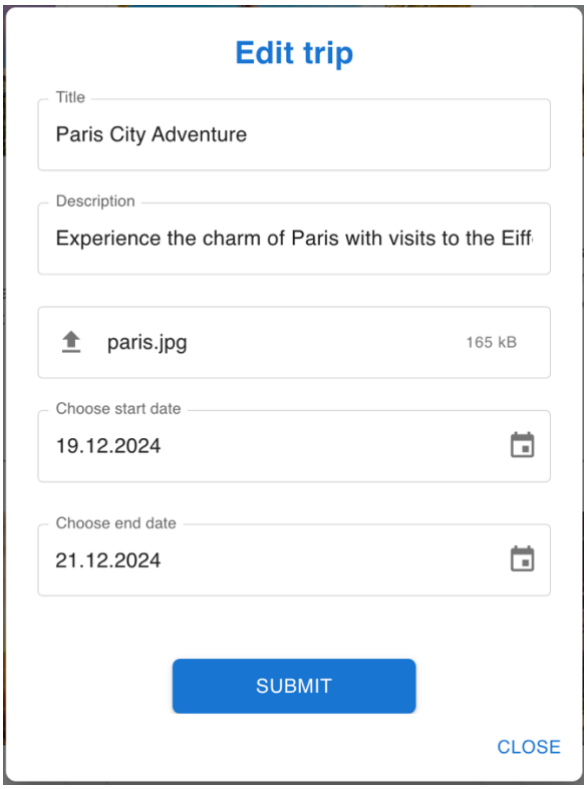


Рисунок 4.9 – Модальне вікно редагування подорожі

Джерело: зроблено автором

Наступним етапом є перехід на сторінку конкретної подорожі, яка призначена для планування її деталей. Після відкриття цієї сторінки в боковому меню з’являються додаткові вкладки: бронювання, створення списку необхідних речей та розрахунок і аналіз бюджету.

На сторінці конкретної подорожі користувач має можливість додати всі бажані локації для відвідування. У процесі пошуку локацій на мапі користувач може зіткнутися з обмеженнями, зокрема, коли бажані місця, такі як маловідомі місцеві пам’ятки, відсутні у базі даних. Для вирішення цієї проблеми було реалізовано функціонал ручного додавання локацій. Цей функціонал передбачає можливість введення користувачем назви об’єкта та його географічних координат. Інтерфейс для мануального додавання локацій реалізовано у вигляді модального вікна, приклад якого представлений на рисунку 4.10.

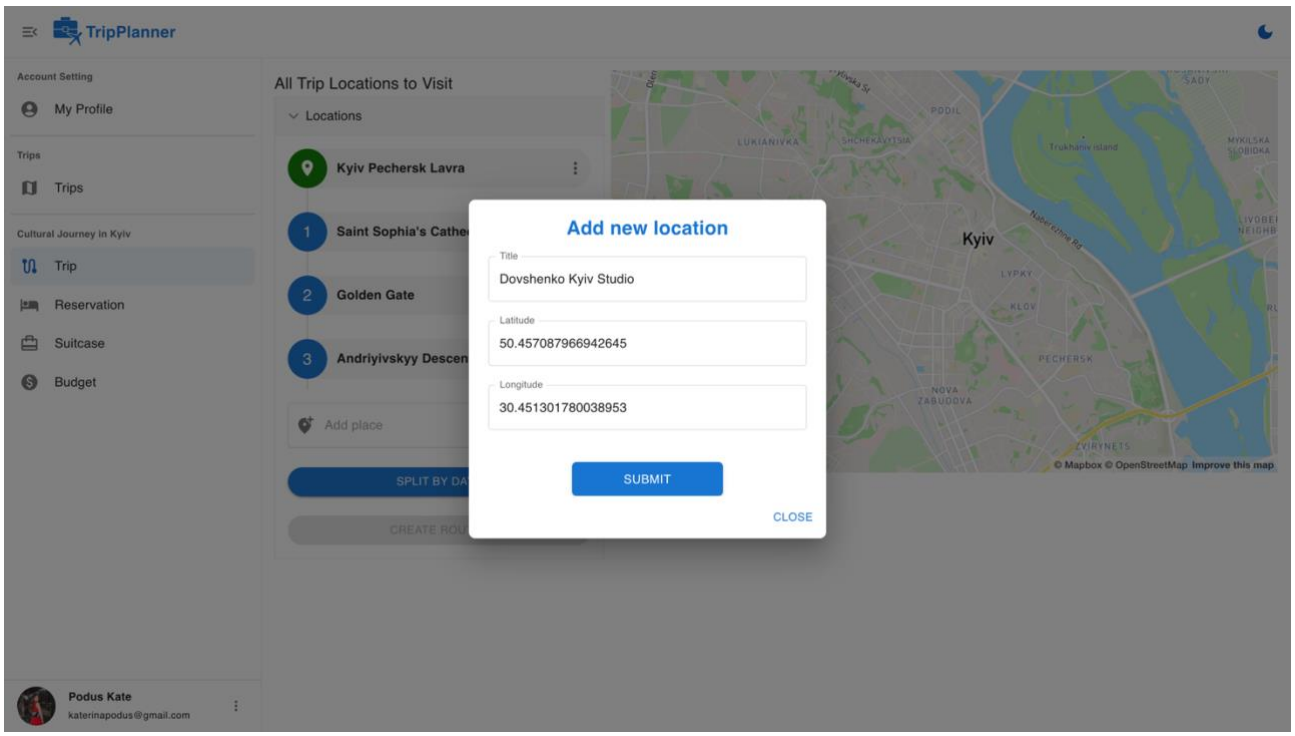


Рисунок 4.10 – Модальне вікно додавання маловідомої нової локації

Джерело: зроблено автором

Після внесення повного списку локацій із зазначенням їхніх деталей система дозволяє кластеризувати всі точки на схожі за площиною кластери за кількістю днів подорожі, натискаючи кнопку «Split by days» . Справа від локацій, на інтерактивній карті відображаються розраховані кластери, за необхідності користувач може повторювати процес кластеризації стільки разів, поки не буде задовільнений результатом. Результат представлений на рисунку 4.11.

Після кластеризації, всі локації заносяться відповідно до їх назначених днів і за допомогою кнопки «Create routes» будуються оптимальні маршрути на кожен день подорожі. Маршрути формуються таким чином, щоб людині було максимально зручно обходити всі точки по часу та дням. Система також враховує обраний вид пересування (пішохідний, автомобільний чи інший транспорт), відповідно до чого під картою генеруються докладні інструкції для проходження обраного шляху.

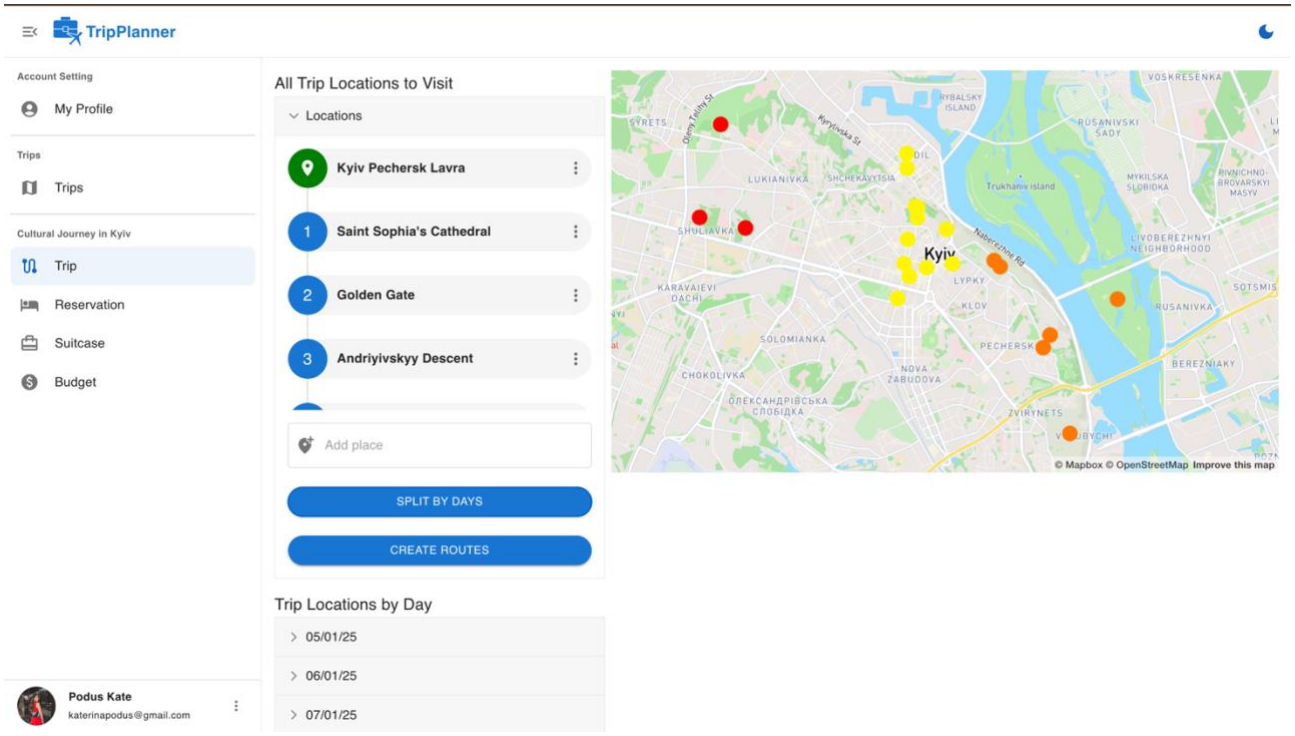


Рисунок 4.11 – Кластеризація всіх локацій подорожі та відображення кластерів на карті, розподілення локацій по дням

Джерело: зроблено автором

Далі під час відкриття певного дня подорожі можна за допомогою кнопки «Show route» відобразити маршрут за цей день на карті та побачити інструкції щодо його обходу. На рисунку 4.12 показаний відображений побудований оптимальний маршрут на перший день подорожі та інструкції до нього.

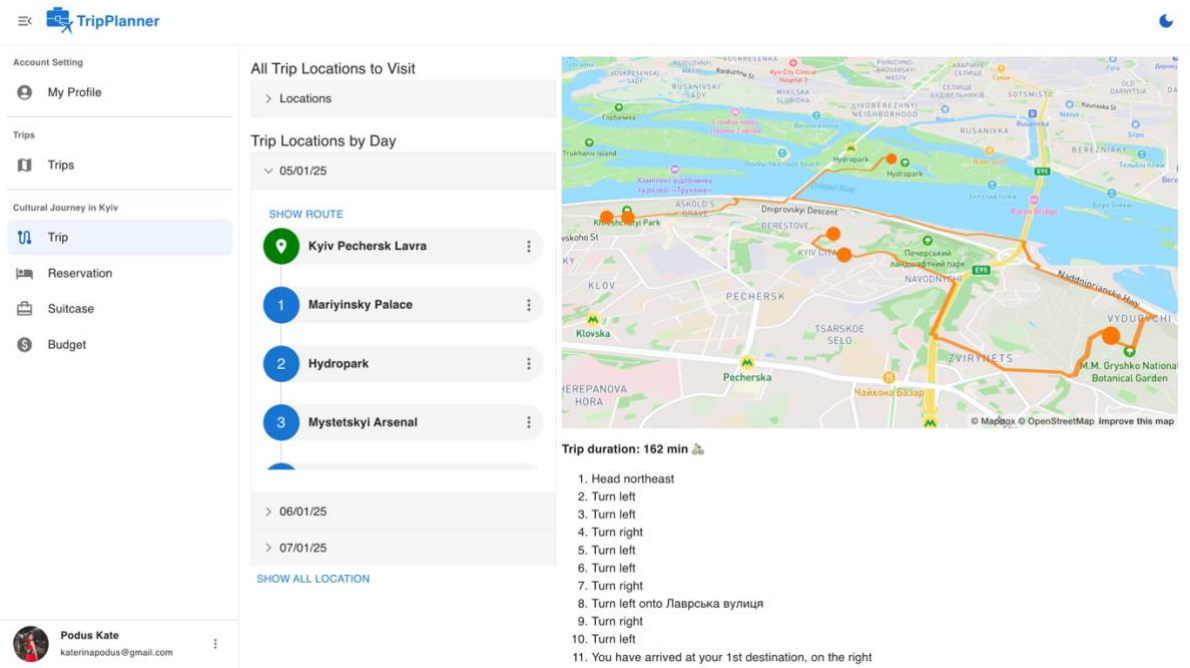


Рисунок 4.12 – Відображення маршруту певного дня та інструкцій його обходу

Джерело: зроблено автором

Кожна додана локація може бути піддана редагуванню, видаленню або доповненню інформацією про бюджет із зазначенням різних категорій витрат. Віконце з даними опціями можна переглянути на рисунку 4.13.

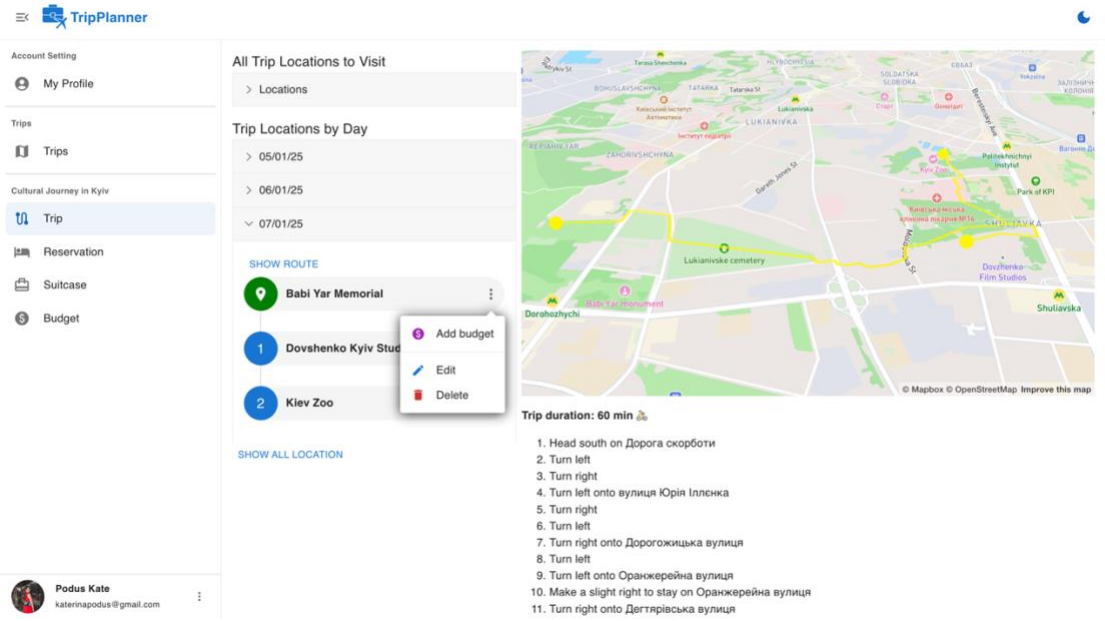


Рисунок 4.13 – Відображення маршруту певного дня та інструкцій його обходу

Джерело: зроблено автором

Для додавання бюджету до локації за категоріями передбачено функціонал управління бюджетом через модальне вікно. У даному вікні користувач має можливість створити нову категорію витрат, вказати для неї відповідну суму та додати її до загального бюджету.

Система також пропонує повторно використовувати раніше створені категорії, полегшуючи процес заповнення. У разі необхідності існуючі категорії можуть бути видалені. Візуальне представлення модального вікна для внесення витрат, пов’язаних із конкретною локацією, наведено на рисунку 4.14.

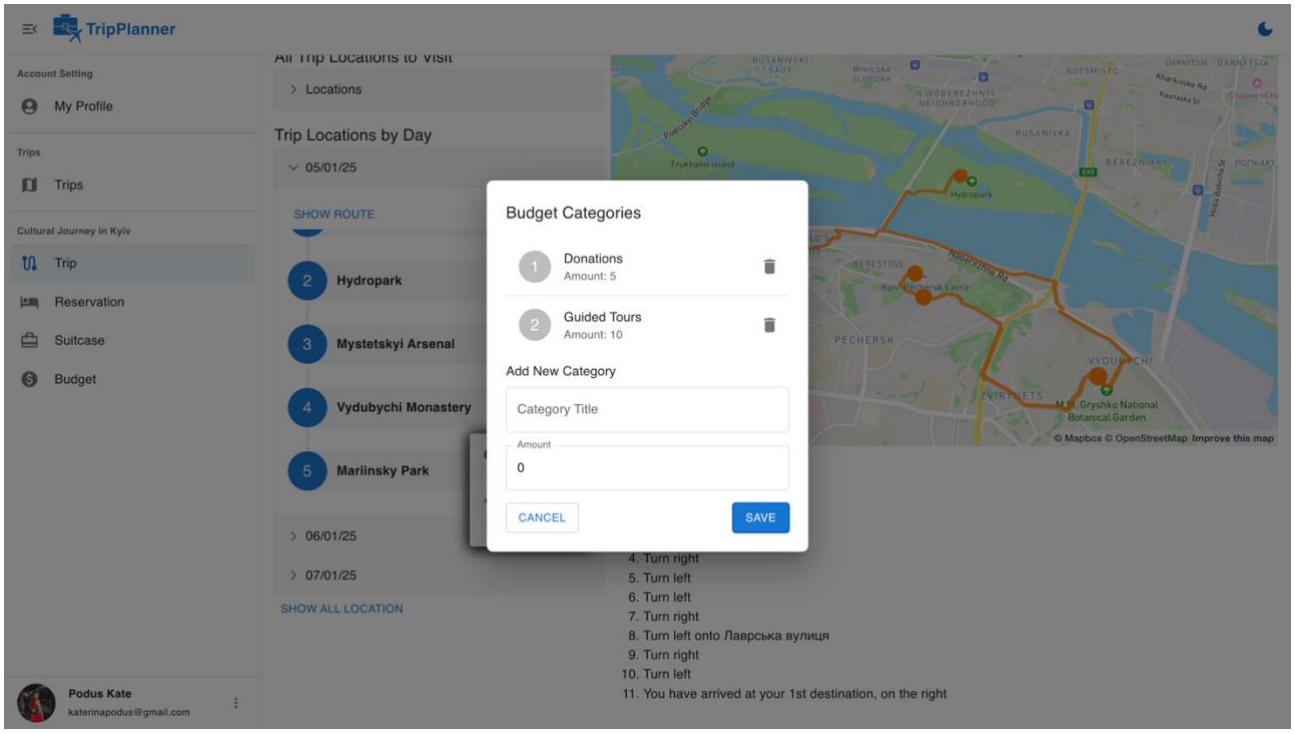


Рисунок 4.14 – Модальне вікно додавання бюджету до локації

Джерело: зроблено автором

На сторінці готелів користувач має можливість переглянути повний список доступних готелів, а також застосувати фільтрацію за необхідними параметрами чи відсортувати їх відповідно до обраних критеріїв (рис. 4.15). Кожна картка готелю містить детальну інформацію про об’єкт, а також надає можливість перейти до сторінки бронювання або додати готель до списку бажаних для подальшого перегляду.

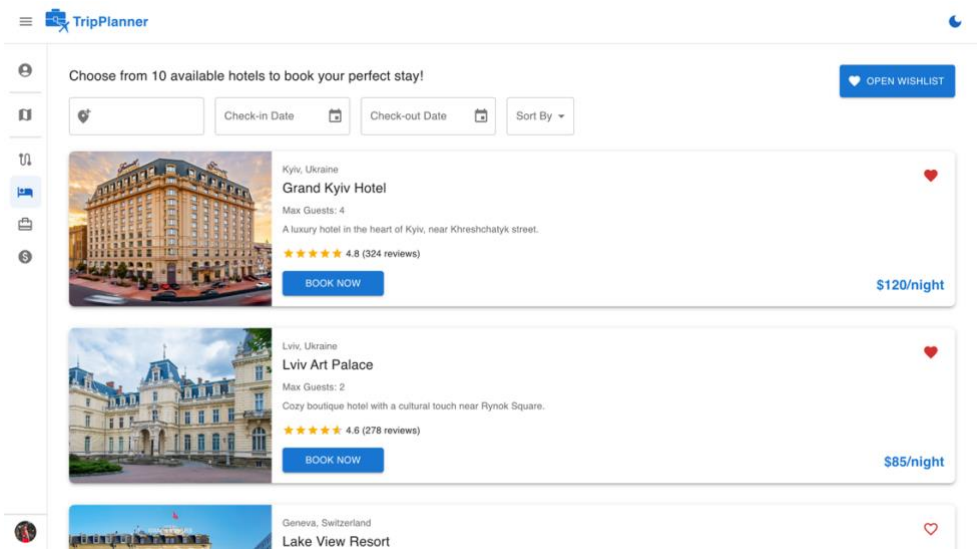


Рисунок 4.15 – Сторінка перегляду всіх готелів для бронювання

Джерело: зроблено автором

Після додавання готелів до списку бажаних користувач може переглянути їх, натиснувши кнопку «Open Wishlist». Це відкриває сторінку зі списком усіх доданих готелів (рис. 4.16).

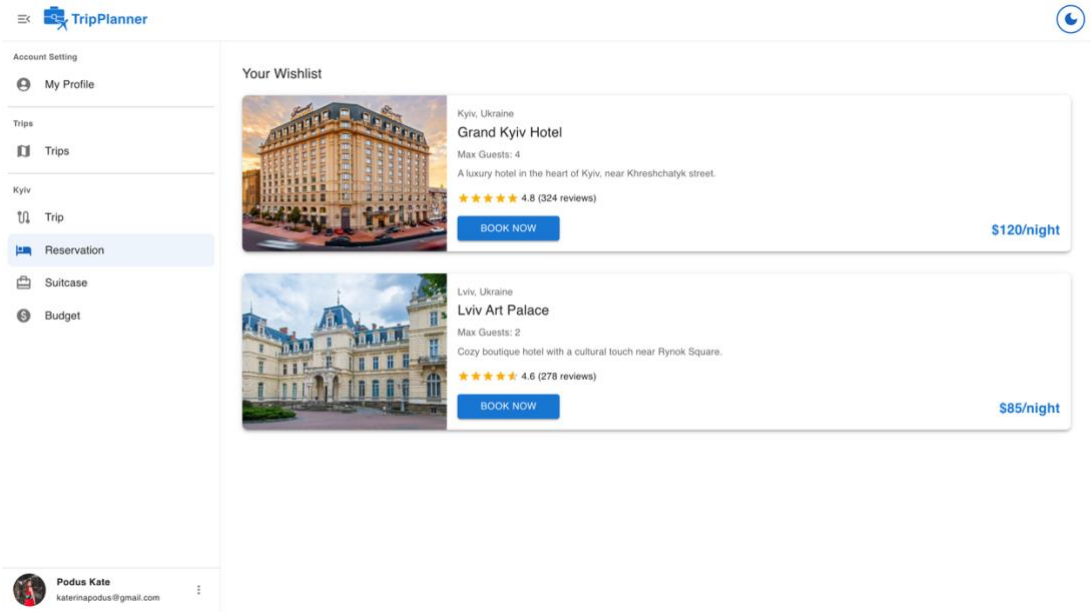


Рисунок 4.16 – Сторінка перегляду готелей, які були додані у список бажаних

Джерело: зроблено автором

Для здійснення бронювання необхідно натиснути кнопку «Book» на картці обраного готелю, після чого з’явиться форма, яку потрібно заповнити. Після

підтвердження введених даних, цей готель буде відображено у профілі користувача. Приклад форми бронювання представлено на рисунку 4.17.

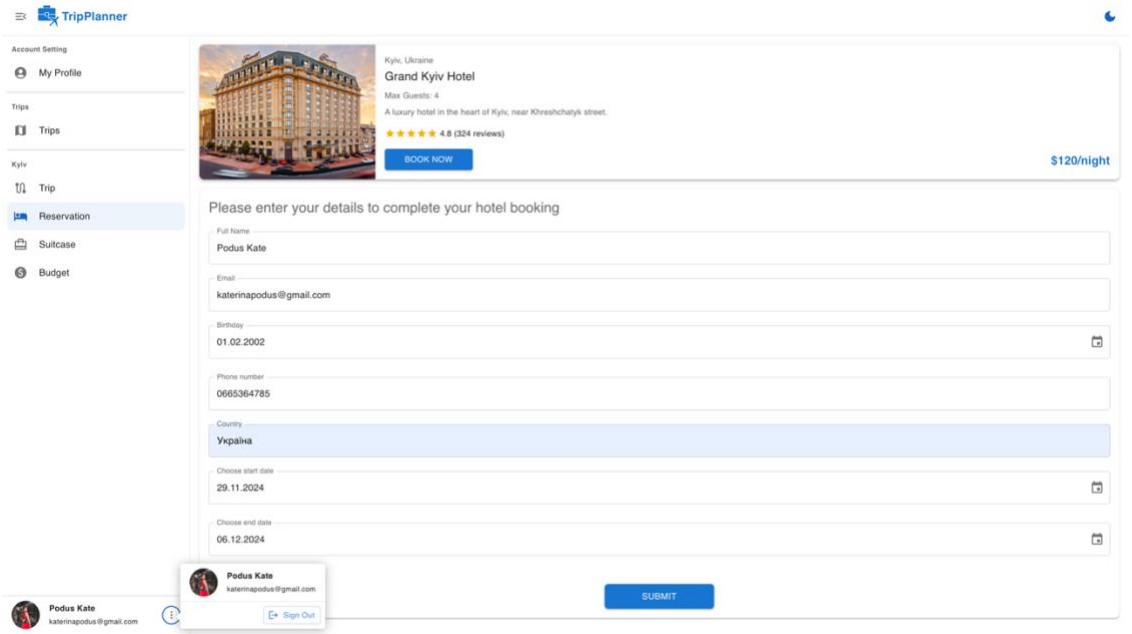


Рисунок 4.17 – Сторінка бронювання готелю

Джерело: зроблено автором

На сторінці розрахунку та аналізу бюджету подорожі користувач має змогу переглянути загальний витрачений бюджет на всю подорож. Окрім цього, на сторінці представлені графіки, які візуалізують розподіл витрат за різними категоріями.

Нижче графіків розміщена історія витрат з детальною інформацією по кожній локації, де наведено категорії витрат, їх суми та загальну суму для кожної локації. Візуальне представлення сторінки бюджету подорожі наведено на рисунку 4.18.

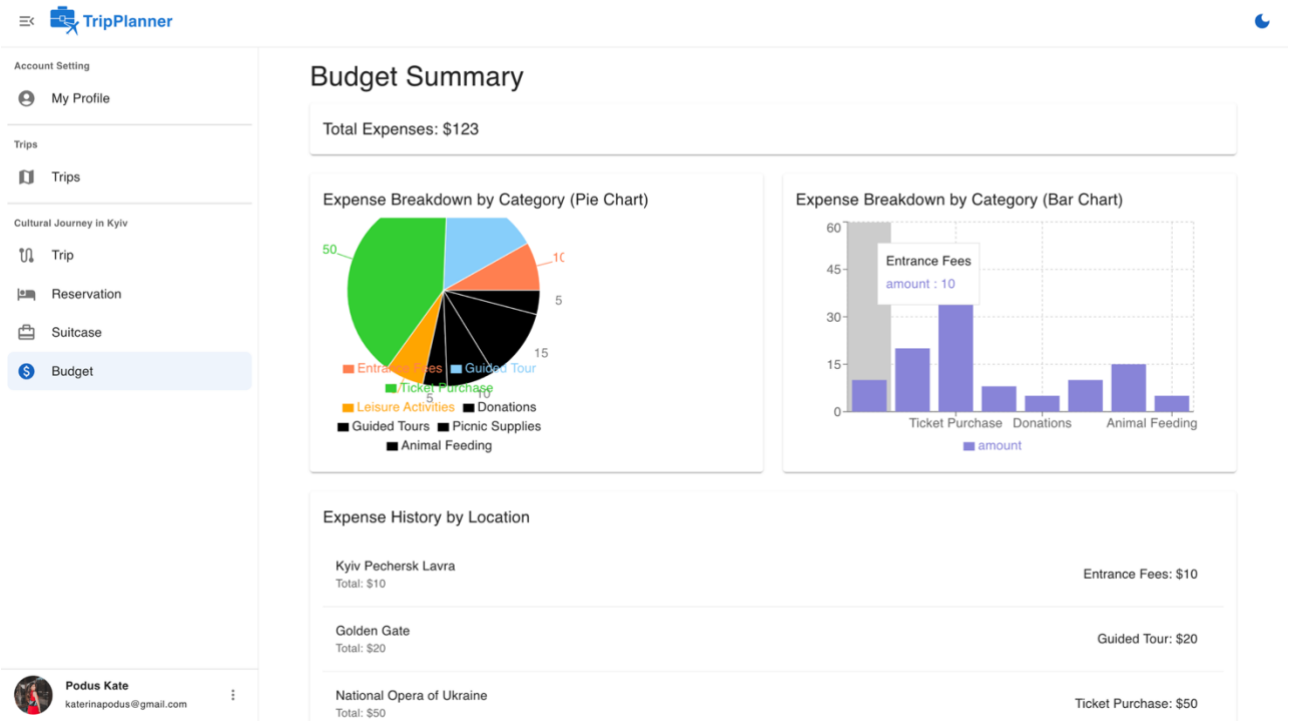


Рисунок 4.18 – Сторінка розрахунку та аналізу бюджету подорожі

Джерело: зроблено автором

Також для кожної подорожі передбачено додаткову сторінку, на якій користувач може додавати перелік необхідних речей. Ці елементи списку можна не лише додавати, а й редагувати або видаляти. Кожну річ можна позначити як таку, що вже взята. Візуальне представлення сторінки для додавання списку речей наведено на рисунку 4.19.

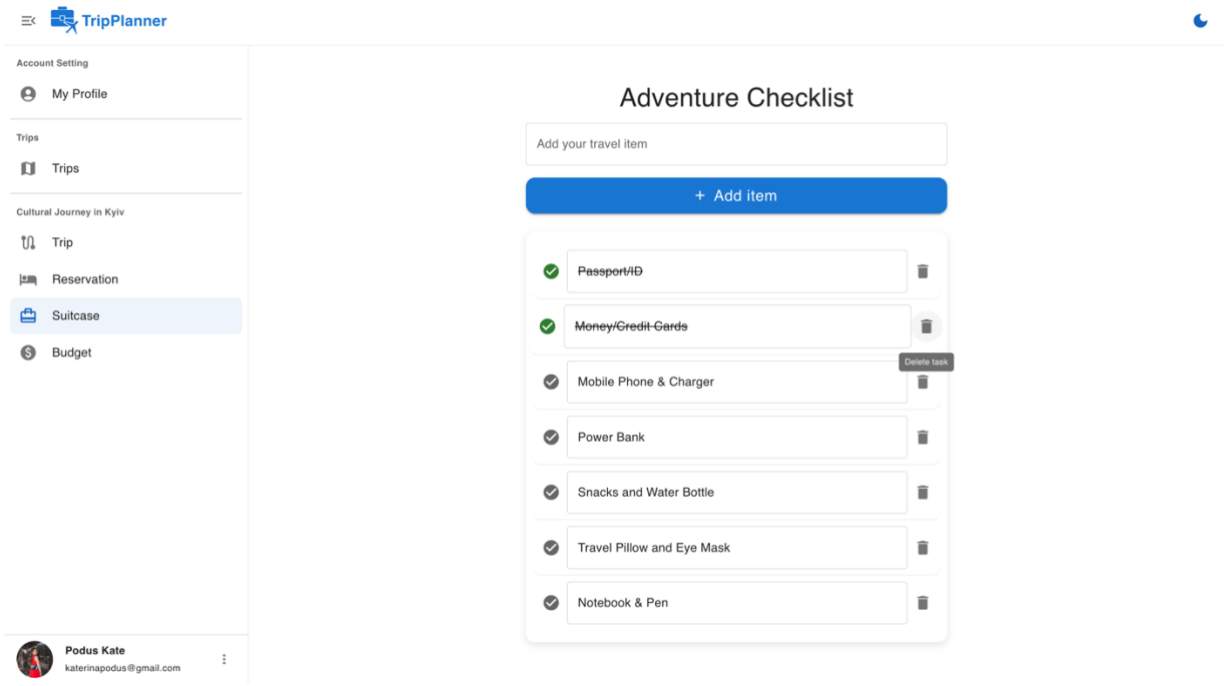


Рисунок 4.19 – Сторінка для переліку списку речей подорожі

Джерело: зроблено автором

Крім цього, на рисунку 4.20 наведений приклад темного режиму вебдодатку на основі сторінки всіх подорожей.

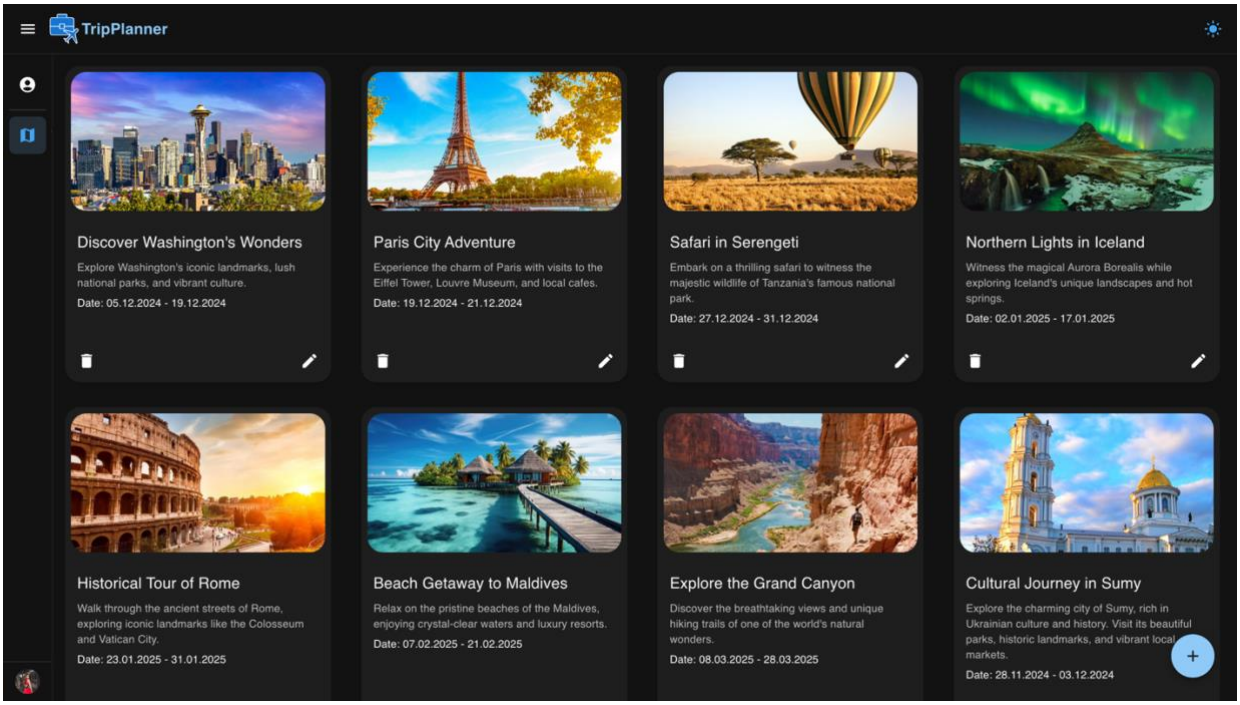


Рисунок 4.20 – Відображення додатку у темному режимі

Джерело: зроблено автором

ВИСНОВКИ

У ході розробки магістерської роботи була створена веборієнтована інформаційна система планування туристичних подорожей. Під час виконання роботи було проаналізовано сучасний стан ринку програмних продуктів для планування подорожей, таких як TripAdvisor, Roadtrippers та Wanderlog. Було встановлено, що існуючі рішення мають зручний інтерфейс і базову функціональність, але не забезпечують достатнього рівня персоналізації та оптимізації маршрутів з урахуванням обмежень часу та інших індивідуальних параметрів користувачів. Порівняльний аналіз існуючих рішень дозволив визначити основні потреби користувачів, які повинні бути враховані при розробці нової системи.

На основі отриманих результатів було сформульовано ключові функціональні вимоги для системи, включаючи можливість створення акаунтів, додавання та редагування інформації про подорож, оптимізацію маршрутів з урахуванням часу та інтеграцію з даними про бронювання. Важливими вимогами також стали функціонал для планування бюджету подорожі, зокрема для аналізу витрат за категоріями, а також можливість формування списку необхідних речей. Ці функціональні можливості дозволяють забезпечити комплексне та зручне планування подорожей для користувачів.

Під час розробки додатку була обрана архітектурна модель «клієнт-сервер», яка є загальноприйнятим стандартом для створення розподілених інформаційних систем. Для реалізації клієнтської частини було застосовано React, що дозволяє створювати інтерактивні та високопродуктивні інтерфейси користувача. У свою чергу, серверна частина побудована на технологіях Node.js та Express, що забезпечує стабільну та ефективну роботу з реляційною базою даних PostgreSQL через ORM Sequelize.

Особливу увагу було приділено інтеграції картографічних сервісів на базі Mapbox GL JS, що дозволяє забезпечити потужну геопросторову візуалізацію та

оптимізацію маршрутів. Використання Mapbox API (Directions, Matrix, Geocoding) надає системі можливість не тільки побудови маршрутів, але й аналізу часових інтервалів між точками маршруту та переведення текстових локацій у географічні координати, що значно розширює функціональні можливості додатку.

Особливістю розробленої системи є модуль кластеризації пам’яток, який був реалізований на основі модифікованого методу K-means, що дозволяє автоматично формувати оптимальні маршрути з урахуванням географічного розташування точок та часу. Використання даної кластеризації забезпечує ефективне групування пам'яток за днями подорожі, що дозволяє значно зменшити час на їх відвідування і підвищити комфортність користувачів.

Таким чином, розробка веборієнтованої інформаційної системи для планування туристичних подорожей сприяє автоматизації створення оптимізованих маршрутів, надаючи користувачам зручний інструментарій для ефективної організації подорожей, включаючи навігацію, розрахунок витрат та управління бронюванням. Адаптивний дизайн і підтримка кросплатформності роблять систему доступною на різних пристроях, що підвищує її зручність і доступність для широкого кола користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Shakhovska N., Sydor P. Development of The Architecture OF The Safe Travel Planning System. Herald of Khmelnytskyi National University. 2022. Vol. 305, no. 1. P. 96–101. URL: https://doi.org/10.31891/2307-5732-2022-305-1-96-101 (date of access: 02.09.2024).

2. Ukrinform. Торік зафіксовано 1,5 мільярда туристичних подорожей – UNWTO. Укрінформ - актуальні новини України та світу. URL: https://www.ukrinform.ua/rubric-tourism/2859960-torik-zafiksovano-15-milarda-turisticnih-podorozej-unwto.html (дата звернення: 04.09.2024).

3. International tourism reached 97% of pre-pandemic levels in the first quarter of 2024. UN Tourism. URL: https://www.unwto.org/news/international-tourism-reached-97-of-pre-pandemic-levels-in-the-first-quarter-of-2024 (date of access: 05.09.2024).

4. Паньків Н., Гуменяк В. Діджиталізація туристичних маршрутів в україні: сучасний стан та тенденції розвитку. Development Service Industry Management. 2024. № 1. С. 253–267. URL: https://doi.org/10.31891/dsim-2024-5(37) (дата звернення: 09.09.2024).

5. US Press Center | About Tripadvisor. MediaRoom. URL: https://tripadvisor.mediaroom.com/us-about-us (дата звернення: 11.09.2024).

6. Roadtrippers Autopilot. Roadtrippers. URL: https://roadtrippers.com/ (date of access: 12.09.2024).

7. Travel and Vacation Planner App LLC. Wanderlog - Travel Planner. App Store. URL: https://apps.apple.com/us/app/wanderlog-travel-planner/id1476732439 (date of access: 13.09.2024).

8. Booking: історія створення та маркетингові прийоми - Bazilik Media. Bazilik Media. URL: https://bazilik.media/booking-istoriia-stvorennia-ta-marketynhovi-pryjomy/ (дата звернення: 15.09.2024).

9. How To Get The Hotels.com API. Travel Solutions | Travel Website | Travel API | Travel GDS. URL: https://www.travelopro.com/how-to-get-the-hotels-com-api (date of access: 19.09.2024).

10. Tripadvisor Help Center. Tripadvisor Help Center. URL: https://www.tripadvisorsupport.com/en-US/hc/owner/articles/517 (date of access: 23.09.2024).

11. Travel APIs: In-depth Guide for Tech Product Companies - asd.team. asd.team. URL: https://asd.team/blog/travel-apis-types-and-integration-specifics/ (date of access: 26.09.2024).

12. Стандартні методи кластеризації даних. Факультет комп'ютерних наук та кібернетики. URL: https://csc.knu.ua/media/study/asp/mod_probl_inf_tech_sys_analysis_ivohin/lecture/lec2.pdf (дата звернення: 01.10.2024).

13. ОБГРУНТУВАННЯ ЗАСТОСУВАННЯ АЛГОРИТМУ КЛАСТЕРИЗАЦІЇ DBSCAN В СИСТЕМІ ДОСТАВКИ. Конференції Державного університету «Житомирська політехніка». URL: https://conf.ztu.edu.ua/wp-content/uploads/2021/01/57.pdf (дата звернення: 05.10.2024).

14. Методи кластерного аналізу. Ієрархічні методи. Система електронного забезпечення навчання ЗНУ. URL: https://moodle.znu.edu.ua/pluginfile.php/486140/mod_resource/content/1/Лекція%2010.pdf (дата звернення: 08.10.2024).

15. API Docs. Mapbox. URL: https://docs.mapbox.com/api/overview/ (date of access: 12.10.2024).

16. Directions API | API Docs. Mapbox. URL: https://docs.mapbox.com/api/navigation/directions/ (date of access: 13.10.2024).

17. Matrix API | API Docs. Mapbox. URL: https://docs.mapbox.com/api/navigation/matrix/ (date of access: 14.10.2024).

18. Лекція 2.1 Функціональне моделювання. Методології функціонального моделювання. Галицький фаховий коледж імені В'ячеслава Чорновола.

URL: https://moodle.gi.edu.ua/mod/book/tool/print/index.php?id=36262&chapterid=571 (дата звернення: 18.10.2024).

19. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. Dou. URL: https://dou.ua/forums/topic/40575/ (дата звернення: 21.10.2024).

20. Створення схеми послідовності UML - Підтримка від Microsoft. Microsoft Support. URL: https://support.microsoft.com/uk-ua/topic/створення-схеми-послідовності-uml-c61c371b-b150-4958-b128-902000133b26 (дата звернення: 24.10.2024).

21. Моралес Дж. UML Activity Diagram: Definition and Tutorial to Create. MindOnMap | Free Mind Mapping Tool to Draw Ideas Easily Online. URL: https://www.mindonmap.com/uk/blog/uml-activity-diagram/ (date of access: 26.10.2024).

22. Розуміння Клієнт-Серверної Архітектури на прикладах. Dou. URL: https://dou.ua/forums/topic/44636/. (дата звернення: 03.12.2024).

23. Посібник: знайомство з React – React. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. URL: https://uk.legacy.reactjs.org/tutorial/tutorial.html (дата звернення: 03.12.2024).

24. ReactDOM – React. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. URL: https://uk.legacy.reactjs.org/docs/react-dom.html (date of access: 03.12.2024).

25. Принципи SOLID. Навіщо вони потрібні програмістам та як з ними працювати. Epam | Campus. URL: https://campus.epam.ua/ua/blog/602. (дата звернення: 03.12.2024).

26. Огляд хуків – React. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. URL: https://uk.legacy.reactjs.org/docs/hooks-overview.html (дата звернення: 03.12.2024).

27. freeCodeCamp. How to Perform CRUD Operations using React, React Hooks, and Axios. freeCodeCamp.org.

URL: https://www.freecodecamp.org/news/how-to-perform-crud-operations-using-react/ (date of access: 03.12.2024).

28. Nasir T. Building Your First REST API with Node JS, Express, and Sequelize. Medium. URL: https://medium.com/@mtalhanasir96/building-your-first-rest-api-with-node-js-express-and-sequelize-b041f9910b8a (date of access: 03.12.2024).

29. API Docs. Mapbox. URL: https://docs.mapbox.com/api/overview/ (date of access: 03.12.2024).

30. Цілі SMART: посібник з постановки, з прикладами. Worksection. URL: https://worksection.com/ua/blog/smart-goals.html (дата звернення: 01.09.2024).

31. WBS: Що це таке та як його використовувати у проекті?. PMTips. URL: https://pmtips.com.ua/post/dlia-cogo-proetku-wbs (дата звернення: 04.09.2024).

32. Тема 4 Структуризація проекту з курсу Управління спеціальними проектами, НУДПСУ. Kursoviks. URL: https://ua.kursoviks.com.ua/metodychni_vkazivky/article_post/869-tema-4-strukturizatsiya-proyektu-z-kursu-upravlinnya-spetsialnimi-proyektami-nudpsu (дата звернення: 06.09.2024).

33. Діаграма Ганта: застосування у менеджменті проєктів | Shelfy. Shelfy. URL: https://shelfy.com.ua/newsroom/diagrama-ganta/ (дата звернення: 03.09.2024).

ДОДАТОК А
Планування робіт

Деталізація мети проєкту здійснюється із застосуванням SMART-методології. Даний підхід дозволяє формулювати цілі, які є конкретними (Specific), вимірюваними (Measurable), досяжними (Achievable), релевантними (Relevant) та обмеженими у часі (Time-framed). Використання цієї методики сприяє чіткому розумінню завдань проєкту та підвищує ймовірність його успішного завершення [30]. Результати застосування SMART-методу наведені у таблиці А.1.

Таблиця А.1 – Деталізація мети проєкту SMART-методом

Критерій	**Мета**
Specific (конкретна)	Створити веборієнтовану інформаційну систему для планування туристичних подорожей, яка дозволить користувачам створювати, зберігати та візуалізувати маршрути, кластеризувати пам’ятки
Measurable (вимірювана)	Приблизно 100 активних користувачів щомісяця та більше 50 створених маршрутів
Achievable (досяжна)	Досвід роботи з подібними проєктами; визначені функціональні вимоги та необхідні інструменти для розробки системи, включаючи знання мов програмування, фреймворків та баз даних: TypeScript, React, NodeJS, PostgreSQL.

Продовження таблиці А.1

Критерій	**Мета**
Relevant (реалістична)	Веборієнтована система для планування подорожей надасть користувачам інструменти для легкого створення маршрутів і їх кластеризації, зберігання важливої інформації про подорож та оптимізації планів, враховуючи індивідуальні інтереси і потреби
Time-framed (обмежена в часі)	Термін виконання проєкту – до кінця грудня 2024 року

Джерело: зроблено автором

Планування змісту структури робіт передбачає створення ієрархічної моделі робіт у вигляді WBS-діаграми (Work Breakdown Structure) – це структурована декомпозиція завдань проєкту, представлена у вигляді графічного дерева. Діаграма демонструє взаємозв'язок між елементами, що входять до складу проєкту, і розподіляє роботи за рівнями деталізації: перший рівень представляє основний продукт, а наступні рівні – складові частини й етапи реалізації [31]. На рисунку А.1 зображено WBS-діаграму, що ілюструє розробку веб-додатку для підтримки дистанційного навчання.

Для впровадження готового проєкту важливим є планування організаційної структури виконавців (OBS – Organization Breakdown Structure). OBS-діаграма формується на основі WBS-структури та визначає відповідальних за конкретні завдання. У даній діаграмі замість окремих видів робіт вказуються виконавці, закріплені за їх виконання [32]. На рисунку А.2 зображено структуру OBS, а таблиця А.2 містить перелік усіх учасників проєкту та їхні обов'язки.

Таблиця А.2 – Виконавці проєкту

Роль	**Ім’я**	**Проєктна роль**
Розробник	Подус К.О.	Відповідає за створення клієнтської та серверної частин веб-додатку.
Проєктувальник	Подус К.О.	Здійснює проєктування бази даних та розробляє архітектуру веб-додатку.
Тестувальник	Подус К.О.	Проводить тестування функціональності веб-додатку та забезпечує якість роботи.
Керівник проєкту	Ващенко С.М.	Координує розробку, визначає задачі та слідкує за дотриманням термінів виконання проєкту.
Менеджер проєкту	Подус К.О.	Здійснює управління ресурсами, розподіл завдань між учасниками та аналіз зібраних даних.

Джерело: зроблено автором

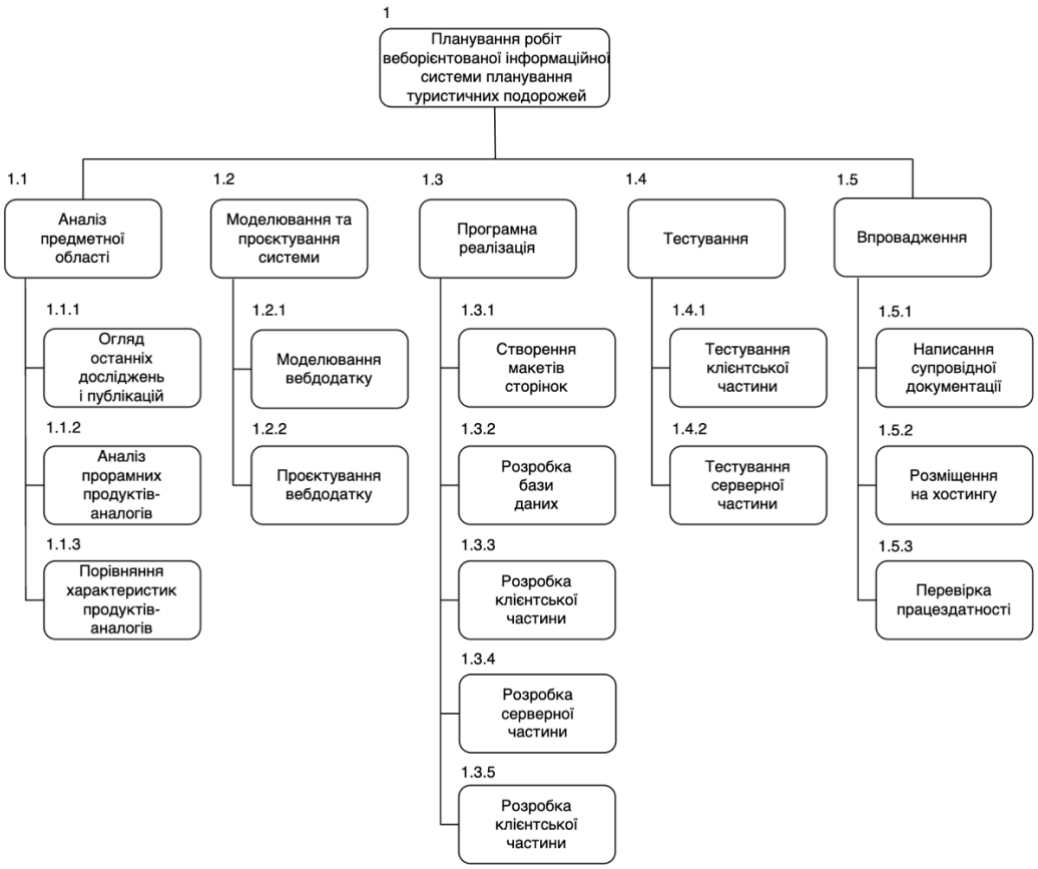


Рисунок А.1 – WBS-структура робіт проєкту

Джерело: зроблено автором

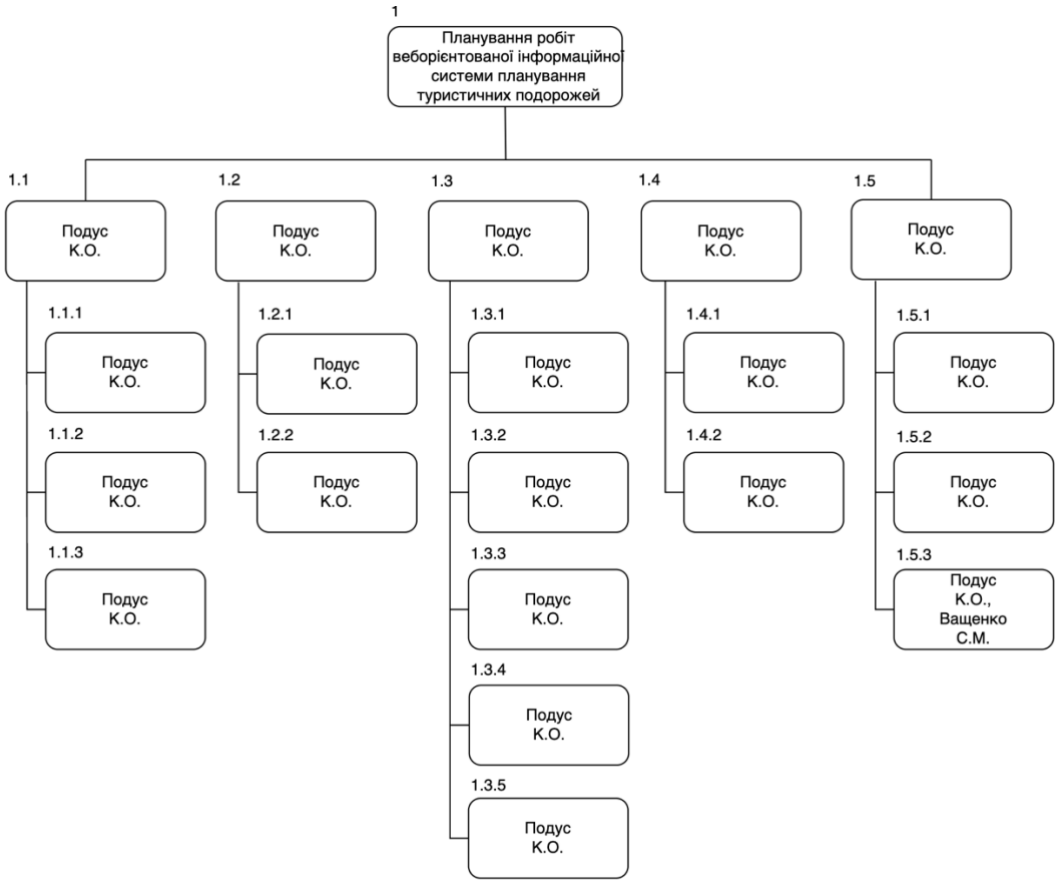
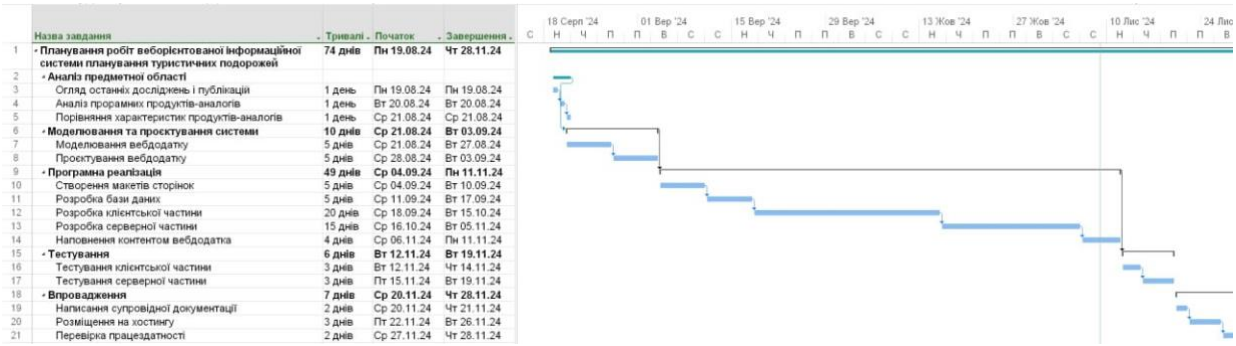


Рисунок А.2 – OBS-структура робіт проєкту

Джерело: зроблено автором

Діаграма Ганта є одним із найпоширеніших інструментів для візуалізації календарних планів виконання робіт. Цей метод надає можливість відобразити всі етапи проєкту, вказуючи точні дати початку та завершення завдань. Завдяки діаграмі можна легко оцінити тривалість виконання проєкту в цілому, а також проаналізувати час, необхідний для реалізації кожного окремого етапу. Вона є важливим інструментом для ефективного управління часом і контролю над процесом розробки продукту, що дозволяє своєчасно коригувати стратегію виконання завдань [33]. Календарний графік проєкту можна побачити на рисунку А.3.



	Назва завдання	Тривалі	Початок	Завершення
1	Планування робіт веборієнтованої інформаційної системи планування туристичних подорожей	74 днів	Пн 19.08.24	Чт 28.11.24
2	Аналіз предметної області			
3	Огляд останніх досліджень і публікацій	1 день	Пн 19.08.24	Пн 19.08.24
4	Аналіз програмних продуктів-аналогів	1 день	Вт 20.08.24	Вт 20.08.24
5	Порівняння характеристик продуктів-аналогів	1 день	Ср 21.08.24	Ср 21.08.24
6	Моделювання та проєктування системи	10 днів	Ср 21.08.24	Вт 03.09.24
7	Моделювання вебдодатку	5 днів	Ср 21.08.24	Вт 27.08.24
8	Проєктування вебдодатку	5 днів	Ср 28.08.24	Вт 03.09.24
9	Програмна реалізація	49 днів	Ср 04.09.24	Пн 11.11.24
10	Створення макетів сторінок	5 днів	Ср 04.09.24	Вт 10.09.24
11	Розробка бази даних	5 днів	Ср 11.09.24	Вт 17.09.24
12	Розробка клієнтської частини	20 днів	Ср 18.09.24	Вт 15.10.24
13	Розробка серверної частини	15 днів	Ср 16.10.24	Вт 05.11.24
14	Наповнення контентом вебдодатка	4 днів	Ср 06.11.24	Пн 11.11.24
15	Тестування	6 днів	Вт 12.11.24	Вт 19.11.24
16	Тестування клієнтської частини	3 днів	Вт 12.11.24	Чт 14.11.24
17	Тестування серверної частини	3 днів	Пт 15.11.24	Вт 19.11.24
18	Впровадження	7 днів	Ср 20.11.24	Чт 28.11.24
19	Написання супровідної документації	2 днів	Ср 20.11.24	Чт 21.11.24
20	Розміщення на хостингу	3 днів	Пт 22.11.24	Вт 26.11.24
21	Перевірка працездатності	2 днів	Ср 27.11.24	Чт 28.11.24

Рисунок А.3 – Календарний план проєкту

Джерело: зроблено автором

Аналіз ризиків є невід'ємною складовою управління проєктом, оскільки практично кожен проєкт стикається із загрозами, що можуть впливати на кінцеві результати. Для забезпечення ефективного управління необхідно попередньо виявити потенційні ризики, оцінити їхню природу та знайти шляхи подолання. Важливо класифікувати ризики за рівнем їхнього впливу, категорією та ймовірністю виникнення, що представлено у таблиці А.3.

Таблиця А.3 – Шкала оцінювання ймовірності виникнення та впливу ризику на виконання проєкту

Оцінка	**Ймовірність виникнення**	**Вплив ризику**	**Тип ризику**
1	Низька	Низький	Прийнятні
2	Середня	Середній	Виправдані
3	Висока	Високий	Недопустимі

Джерело: зроблено автором

Ідентифікація ризиків проєкту. Для веборієнтованої інформаційної системи планування туристичних подорожей були визначені потенційні ризики, які можуть негативно вплинути на її функціонування та ефективність. Опис ймовірних ризиків подано в таблиці А.4.

Таблиця А.4 – Ризики проєкту

№ ризику	**Назва ризику**
1	Використання невідповідних інструментів для розробки
2	Постійні зміни та уточнення вимог до проєкту
3	Технічні недоліки та помилки в розробці
4	Невірне планування та розподіл часу між етапами
5	Недостатній рівень кваліфікації
6	Проблеми, пов’язані з програмним забезпеченням

Джерело: зроблено автором

Для повного розуміння оцінки ризиків, визначимо окремо кожний з них за впливом, ймовірністю виникнення та рангом. Всі дані описані в таблиці А.5.

Таблиця А.5 – Визначення ймовірності, впливу та рангу ризиків проєкту

№ ризику	**Назва ризику**	**Ймовірність (0.1-0.9)**	**Вплив (0.05-0.8)**	**Ранг**
1	Використання невідповідних інструментів для розробки	0.3	0.4	0.12
2	Постійні зміни та уточнення вимог до проєкту	0.5	0.2	0.1
3	Технічні недоліки та помилки в розробці	0.9	0.05	0.045
4	Невірне планування та розподіл часу між етапами	0.7	0.4	0.28
5	Недостатній рівень кваліфікації	0.1	0.1	0.01
6	Проблеми, пов’язані з програмним забезпеченням	0.1	0.8	0.08

Джерело: зроблено автором

Використовуючи значення, представлені в таблиці А.5, можна побудувати матрицю ймовірності та впливу ризиків для проєкту, яка наведена в таблиці А.6. Ризики, що розташовані у зелених комірках, вважаються прийнятними; ризики у жовтих комірках є виправданими; тоді як червоні комірки позначають недопустимі ризики.

Таблиця А.6 – Матриця ймовірності та впливу ризиків

Ймовірність ризику	**Вплив загрози**				
	Дуже малий	**Малий**	**Середній**	**Великий**	**Дуже великий**
	0,05	0,1	0,2	0,4	0,8
0,9	R3(0,045)				
0,7				R4(0,28)	
0,5			R2(0,1)		
0,3				R1(0,12)	
0,1		R5(0,01)			R6(0,08)

Джерело: зроблено автором

Згідно з отриманими індексами, ризики можна класифікувати за рівнями, що наведені в таблиці А.7. Окрім того, були розроблені заходи щодо запобігання виникненню ризиків і управління їх наслідками. Детальну інформацію про стратегії реагування можна переглянути в таблиці А.8.

Таблиця А.7 – Шкала оцінювання за рівнем ризику

№	**Назва**	**Межі**	**Ризики, які входять (номера)**
1	Прийнятні	$0,005 \leq R \leq 0,05$	3, 5
2	Виправдані	$0,05 < R \leq 0,14$	6, 2, 1
3	Недопустимі	$0,14 < R \leq 0,72$	4

Джерело: зроблено автором

Таблиця А.8 – Ризики та стратегії реагування

ID	**Статус**	**Опис**	**Ймовірність виникнення**	**Вплив**	**Ранг**	**План А (заходи запобігання виникненню ризику)**	**Тип стратегії реагування**	**План Б (заходи усунення наслідків ризику)**
RS_1	Відкритий	Використання невідповідних інструментів для розробки	0.3	0.4	0.12	Попереднє детальне вивчення відповідної предметної сфери та інструментів для створення подібних продуктів	Попередження	Адаптація інструментів розробки
RS_2	Відкритий	Постійні зміни та уточнення вимог до проєкту	0.5	0.2	0.1	Здійснити всебічний аналіз відповідної предметної області та встановити вимоги до проєкту	Попередження	Коригування вимог до проєкту

Продовження таблиці А.8

ID	**Статус**	**Опис**	**Ймовірність виникнення**	**Вплив**	**Ранг**	**План А (заходи запобігання виникненню ризику)**	**Тип стратегії реагування**	**План Б (заходи усунення наслідків ризику)**
RS_3	Відкритий	Технічні недоліки та помилки в розробці	0.9	0.05	0.045	Здійснювати зберігання копій різних версій коду	Зменшення	Виявлення та усунення помилок
RS_4	Відкритий	Невірне планування та розподіл часу між етапами	0.7	0.4	0.28	Оцінити час з урахуванням усіх факторів, що можуть вплинути на процес.	Попередження	Перерахувати оновлений час для виконання роботи, враховуючи нові змінні впливу.

Продовження таблиці А.8

ID	**Статус**	**Опис**	**Ймові-рність Виникне-ння**	**Вплив**	**Ранг**	**План А (заходи запобігання виникненню ризику)**	**Тип стратегії реагування**	**План Б (заходи усунення наслідків ризику)**
RS_5	Відкритий	Недостатній рівень кваліфікації	0.1	0.1	0.01	Провести аналіз предметної області для визначення необхідних знань	Зменшення	Пошук актуальної інформації для вирішення поставленої проблеми
RS_6	Відкритий	Проблеми, пов’язані з програмним забезпеченням	0,1	0.8	0.08	Робити резервні копії даних та мати додаткове ПЗ.	Попередження	Усунути проблеми або знайти альтернативне ПЗ.

Джерело: зроблено автором

ДОДАТОК Б

Коди основних модулів

centriod.ts

```
const centroid = (set: number[][]) =>
  set
    .reduce((s, e) => s.map((item, i) => item + e[i]), [0, 0])
    .map((e) => e / set.length);

export default centroid;
```

clusterize.ts

```
import centroid from "./centriod";
import planarDistance from "./planarDistance";

type Point = number[];
type Cluster = { centroid: Point; elements: Point[] };

const clusterize = (
  elements: Point[],
  distance: (point1: number[], point2: number[]) => number,
  threshold: (
    elements: Point[],
    distance: (point1: number[], point2: number[]) => number
  ) => number
) => {
  const thr = threshold(elements, distance);

  let clusterMap: Cluster[] = [];

  clusterMap.push({
    centroid: elements[Math.floor(Math.random() *
elements.length)],
    elements: [],
  });

  let changing = true;
  while (changing) {
    changing = false;
    clusterMap.forEach((cluster) => (cluster.elements = []));

    elements.forEach((element) => {
      let closest_dist = Infinity;
      let closest_cluster: number = 0;

      clusterMap.forEach((cluster, i) => {
```

```
        const dist = distance(element, cluster.centroid);

        if (dist < closest_dist) {
          closest_dist = dist;
          closest_cluster = i;
        }
      });

      if (closest_dist < thr || closest_dist === 0) {
        clusterMap[closest_cluster].elements.push(element);
      } else {
        clusterMap.push({ centroid: element, elements: [element]
});

        changing = true;
      }
    });

    clusterMap = clusterMap.filter((cluster) =>
cluster.elements.length > 0);

    clusterMap.forEach(function (cluster, i) {
      const centre = centroid(cluster.elements);
      if (planarDistance(centre, cluster.centroid) > 0) {
        clusterMap[i].centroid = centre;
        changing = true;
      }
    });
  }

  return clusterMap;
};

export default clusterize;
```

planarDistance.ts

```
const planarDistance = (point1: number[], point2: number[]):
number =>
  Math.hypot(...point1.map((p, i) => p - point2[i]));
export default planarDistance;
```

sphericalDistance.ts

```
const toRad = (deg: number): number => (deg * Math.PI) / 180;

const sphericalDistance = (point1: number[], point2: number[]):
number => {
  const [lat1, lon1] = point1;
  const [lat2, lon2] = point2;

  const dlat = toRad(lat2 - lat1);
```

```
  const dlon = toRad(lon2 - lon1);
  const a =
    Math.sin(dlat / 2) * Math.sin(dlat / 2) +
    Math.sin(dlon / 2) *
      Math.sin(dlon / 2) *
      Math.cos(toRad(lat1)) *
      Math.cos(toRad(lat2));
  return 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
};

export default sphericalDistance;
```

App.tsx

```
import { Outlet } from "react-router-dom";
import { AppProvider } from "@toolpad/core/react-router-dom";
import { Navigation } from "@toolpad/core/AppProvider";
import { createTheme } from "@mui/material";
import AccountCircle from "@mui/icons-material/AccountCircle";
import MapIcon from "@mui/icons-material/Map";
import RouteIcon from "@mui/icons-material/Route";
import CardTravel from "@mui/icons-material/CardTravel";
import HotelIcon from "@mui/icons-material/Hotel";
import Paid from "@mui/icons-material/Paid";
import { useCallback, useContext, useMemo, useState } from
"react";
import { TripContext } from "./contexts/TripContext";
import { LocalizationProvider } from "@mui/x-date-
pickers/LocalizationProvider";
import { AdapterDayjs } from "@mui/x-date-pickers/AdapterDayjs";
import "dayjs/locale/de";
import { DialogsProvider } from "@toolpad/core";
import { UserContext } from "./contexts/UserContext";
import { Trip } from "./services/TripService";

const theme = createTheme({
  cssVariables: {
    colorSchemeSelector: "data-toolpad-color-scheme",
  },
  colorSchemes: { light: true, dark: true },
  breakpoints: {
    values: {
      xs: 0,
      sm: 600,
      md: 600,
      lg: 1200,
      xl: 1536,
    },
  },
});

const branding = {
  title: "TripPlanner",
```

```
  logo: <img src="/logo.svg" alt="Logo" />,
};

export const App = () => {
  const { user, openAuthModal, logout } = useContext(UserContext);
  const [trip, setTrip] = useState<Trip | null>(null);



  const session = useMemo(
    () => ({
      user: user
        ? {
            id: `${user.id}`,
            name: user.fullName,
            image:  user.avatar ? import.meta.env.VITE_MEDIA_URL +
"/" + user.avatar : null,
            email: user.email,
          }
        : undefined,
    }),
    [user]
  );

  const authentication = useMemo(() => {
    return {
      signIn: openAuthModal,
      signOut: logout,
    };
  }, [openAuthModal, logout]);

  const activate = useCallback(
    (newTrip: Trip | null) => newTrip?.id !== trip?.id &&
setTrip(newTrip),
    [trip]
  );
  const navigation: Navigation = useMemo(
    () =>
      (
        [
          {
            kind: "header",
            title: "Account Setting",
          },
          {
            kind: "page",
            segment: "profile",
            title: "My Profile",
            icon: <AccountCircle />,
          },
          {
            kind: "divider",
          },
```

```
        {
          kind: "header",
          title: "Trips",
        },
        {
          kind: "page",
          segment: "trips",
          title: "Trips",
          icon: <MapIcon />,
        },
      ] as Navigation
    ).concat(
      trip
        ? ([
            {
              kind: "divider",
            },
            {
              kind: "header",
              title: trip.title,
            },
            {
              kind: "page",
              segment: `trips/${trip.id}`,
              title: "Trip",
              icon: <RouteIcon />,
            },
            {
              kind: "page",
              segment: `trips/${trip.id}/reservation`,
              title: "Reservation",
              icon: <HotelIcon />,
              pattern: "trips/{:id}/reservation{/:id}*",
            },
            {
              kind: "page",
              segment: `trips/${trip.id}/suitcase`,
              title: "Suitcase",
              icon: <CardTravel />,
            },
            {
              kind: "page",
              segment: `trips/${trip.id}/budget`,
              title: "Budget",
              icon: <Paid />,
            },
          ] as Navigation)
        : []
    ),
  [trip]
);

return (
```

```
      <AppProvider
        navigation={navigation}
        branding={branding}
        theme={theme}
        authentication={authentication}
        session={session}
      >
        <LocalizationProvider dateAdapter={AdapterDayjs}
adapterLocale="de">
          <DialogsProvider>
            <TripContext.Provider value={{ activate }}>
              <Outlet />
            </TripContext.Provider>
          </DialogsProvider>
        </LocalizationProvider>
      </AppProvider>
  );
};
```

config.ts

```
const config = {
  MAX_FILE_SIZE: 10 * (1 << 20),
};

export default config;
```

TripContext.ts

```
import { createContext } from "react";
import { Trip } from "../services/TripService";

export interface TripContextValue {
  activate: (newTrip: Trip | null) => void;
}

export const TripContext = createContext<TripContextValue>({
  activate: () => {},
});
```

UserContext.ts

```
import { createContext } from "react";

export interface User {
  id: number;
  email: string;
  full_name: string;
  avatar?: string;
  password?: string;
  activation_link?: string;
```

```
  is_activated: boolean;
  created_at: Date
}

interface UserContextType {
  user: User | null;
  isUserAuthorized: boolean;
  setUser: (user: User | null) => void;
  openAuthModal: () => void;
  logout: () => void;
}

export const UserContext = createContext<UserContextType>({
  user: null,
  isUserAuthorized: false,
  setUser: () => {},
  openAuthModal: () => {},
  logout: () => {},
});
```

useLoading.ts

```
import { useState, useCallback } from "react";

type UseLoadingReturn = {
  isLoading: boolean;
  error: Error | null;
  execute: (...args: unknown[]) => Promise<void>;
  reset: () => void;
};

function useLoading(
  func: (...args: unknown[]) => Promise<void> | void
): UseLoadingReturn {
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState<Error | null>(null);

  const execute = useCallback(
    async (...args: unknown[]) => {
      setIsLoading(true);
      setError(null);
      try {
        await func(...args);
      } catch (err) {
        setError(err as Error);
        throw err;
      } finally {
        setIsLoading(false);
      }
    },
    [func]
  );
```

```
  const reset = useCallback(() => {
    setIsLoading(false);
    setError(null);
  }, []);

  return { isLoading, error, execute, reset };
}

export default useLoading;
```

Api.ts

```
import axios from "axios";

const $api = axios.create({
  withCredentials: true,
  baseURL: import.meta.env.VITE_API_URL + "/api/",
});

$api.interceptors.request.use((config) => {
  config.headers.Authorization = `Bearer ${localStorage.getItem(
    "access-token"
  )}`;
  return config;
});

$api.interceptors.response.use(
  (config) => {
    return config;
  },
  async (error) => {
    const originalRequest = error.config;
    if (
      error.response.status === 401 &&
      error.config &&
      !error.config._isRetry
    ) {
      originalRequest._isRetry = true;
      try {
        const response = await $api.get("auth/refresh", {
          withCredentials: true,
        });
        localStorage.setItem("access-token",
response.data.accessToken);
        return $api.request(originalRequest);
      } catch (error) {
        console.log(error, "Not authorized");
      }
    }
    throw error;
  }
);
```

```
export default $api;
```

dashboard.tsx

```
import { useMemo } from "react";
import { Outlet } from "react-router-dom";
import { Box, Divider, Stack } from "@mui/material";
import {
  Account,
  AccountPreview,
  AccountPreviewProps,
  DashboardLayout,
  SidebarFooterProps,
} from "@toolpad/core";

function AccountSidebarFooter({ mini }: SidebarFooterProps) {
  const PreviewComponent = useMemo(
    () =>
      function (props: AccountPreviewProps) {
        const { handleClick, open } = props;
        return (
          <Stack direction="column" p={0} overflow="hidden">
            <Divider />
            <AccountPreview
              variant={mini ? "condensed" : "expanded"}
              handleClick={handleClick}
              open={open}
            />
          </Stack>
        );
      },
    [mini]
  );

  return (
    <Stack direction="column" p={0} overflow="hidden">
      <Divider />
      <Account
        slots={{ preview: PreviewComponent }}
        slotProps={{
          popover: {
            transformOrigin: { horizontal: "left", vertical:
"bottom" },
            anchorOrigin: { horizontal: "right", vertical:
"bottom" },
            disableAutoFocus: true,
            slotProps: {
              paper: {
                elevation: 0,
                sx: {
                  overflow: "visible",
                  filter: (theme) =>
```

```
                    `drop-shadow(0px 2px 8px ${theme.palette.mode
=== "dark" ? "rgba(255,255,255,0.10)" : "rgba(0,0,0,0.32)"})`,
                  mt: 1,
                  "&::before": {
                    content: '""',
                    display: "block",
                    position: "absolute",
                    bottom: 10,
                    left: 0,
                    width: 10,
                    height: 10,
                    bgcolor: "background.paper",
                    transform: "translate(-50%, -50%)
rotate(45deg)",
                    zIndex: 0,
                  },
                },
              },
            },
          },
        }}
      />
    </Stack>
  );
}

export default function Layout() {
  return (
    <DashboardLayout
      slots={{
        toolbarAccount: () => null,
        sidebarFooter: AccountSidebarFooter,
      }}
    >
      <Box m={2}>
        <Outlet />
      </Box>
    </DashboardLayout>
  );
}
```

main.tsx

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import "./index.css";
import { App } from "./App.tsx";
import { createBrowserRouter, RouterProvider } from "react-router-
dom";
import Layout from "./layouts/dashboard.tsx";
import { HomePage } from "./pages/HomePage/HomePage.tsx";
import { ProfilePage } from "./pages/ProfilePage/ProfilePage.tsx";
import { TripsPage } from "./pages/TripsPage/TripsPage.tsx";
```

```
import { ReservationPage } from
"./pages/ReservationPage/ReservationPage.tsx";
import { SuitcasePage } from
"./pages/SuitcasePage/SuitcasePage.tsx";
import { BudgetPage } from "./pages/BudgetPage/BudgetPage.tsx";
import { TripPage } from "./pages/TripPage/TripPage.tsx";
import { HotelPage } from "./pages/HotelPage/HotelPage.tsx";
import { HotelsWishlistPage } from
"./pages/HotelsWishlistPage/HotelsWishlistPage.tsx";
import { AuthWrapper } from
"./pages/Authorization/AuthWrapper.tsx";

const router = createBrowserRouter([
  {
    Component: App,
    children: [
      {
        path: "/",
        Component: Layout,
        children: [
          {
            path: "home",
            Component: HomePage,
          },
          {
            path: "profile",
            Component: ProfilePage,
          },
          {
            path: "trips/",
            children: [
              {
                index: true,
                Component: TripsPage,
              },
              {
                path: ":tripId",
                children: [
                  {
                    index: true,
                    Component: TripPage,
                  },
                  {
                    path: "reservation/",
                    children: [
                      {
                        index: true,
                        Component: ReservationPage,
                      },
                      {
                        path: ":hotelId",
                        Component: HotelPage,
                      },
```

```
                        {
                          path: "wishlist",
                          Component: HotelsWishlistPage,
                        },
                      ],
                    },
                    {
                      path: "suitcase",
                      Component: SuitcasePage,
                    },
                    {
                      path: "budget",
                      Component: BudgetPage,
                    },
                  ],
                },
              ],
            },
          ],
        },
      ],
    },
]);

createRoot(document.getElementById("root")!).render(
  <StrictMode>
    <AuthWrapper>
      <RouterProvider router={router} />
    </AuthWrapper>
  </StrictMode>
);
```

AuthWrapper.tsx

```
import { useState, useEffect, useCallback, ReactNode } from
'react';
import { User, UserContext } from '../../contexts/UserContext';
import { AuthModal } from './components/AuthModal/AuthModal';
import AuthService from '../../services/AuthService';
import $api from '../../http/Api';

interface AuthWrapperProps {
  children: ReactNode;
}

export const AuthWrapper = ({ children }: AuthWrapperProps) => {
  const [user, setUser] = useState<User | null>(null);
  const [showAuthModal, setShowAuthModal] =
useState<boolean>(false);

  const isUserAuthorized = !!user;

  const openAuthModal = useCallback(() => {
```

```
    setShowAuthModal(true);
  }, []);

  const logout = useCallback(async () => {
    await AuthService.logout();
    localStorage.removeItem("access-token");
    setUser(null);
  }, []);

  const checkAuth = async () => {
    try {
      const response = await $api.get(`auth/refresh`, {
withCredentials: true });
      localStorage.setItem('access-token',
response.data.accessToken);
      setUser(response.data.user);
    } catch (error: unknown) {
      console.error(error);
      if (error instanceof Error) {
        console.error(error);
      } else {
        console.log('An unknown error occurred');
      }
    }
  };

  useEffect(() => {
    if (localStorage.getItem('access-token')) {
      checkAuth();
    }
  }, []);

  return (
    <UserContext.Provider value={{ user, setUser,
isUserAuthorized, openAuthModal, logout }}>
      <AuthModal
        open={showAuthModal}
        onClose={() => {
          setShowAuthModal(false);
        }}
      />
      {children}
    </UserContext.Provider>
  );
};
```

AuthModal.tsx

```
import { useState } from 'react';
import { Dialog, DialogTitle, DialogContent, DialogActions,
Button, Box } from '@mui/material';
import { LoginForm } from '../LoginForm/LoginForm';
```

```
import { RegistrationForm } from
'../RegistrationForm/RegistrationForm';

interface AuthModalProps {
  open: boolean;
  onClose: () => void;
}

export const AuthModal = ({ open, onClose }: AuthModalProps) => {
  const [userRegistered, setUserRegistered] =
useState<boolean>(true);

  const onCloseModal = () => {
    onClose();
    setUserRegistered(true);
  };

  return (
    <Dialog
      open={open}
      onClose={onClose}
      scroll={'paper'}
      maxWidth="xs"
      fullWidth
      PaperProps={{
        sx: { borderRadius: 2 },
      }}
    >
      <Box sx={{ display: 'flex', justifyContent: 'center' }}>
        <DialogTitle sx={{ pt: 2, pb: 0, fontSize: '24px' }}
color="primary">
          <strong>{userRegistered ? 'Sign in' : 'Sign
Up'}</strong>
        </DialogTitle>
      </Box>
      <DialogContent sx={{ py: 0 }}>
        {userRegistered ? <LoginForm onClose={onCloseModal} /> :
<RegistrationForm onClose={onCloseModal} />}
      </DialogContent>
      <DialogActions>
        <Button onClick={() => setUserRegistered((prev) =>
!prev)}>
          {userRegistered ? 'Sign Up' : 'Are you already
registered?'}
        </Button>
      </DialogActions>
    </Dialog>
  );
};
```

LoginForm.tsx

```
import * as yup from 'yup';
```

```
import { useContext } from 'react';
import { TextField } from '@mui/material';
import { PasswordField } from
'../../../Shared/components/Fields/PasswordField/PasswordField';
import { UserContext } from '../../../../contexts/UserContext';
import AuthService, { LoginValues } from
'../../../../services/AuthService';
import { FieldRendererProps, Form } from
'../../../Shared/components/Form/Form';

const initialValues: LoginValues = {
  email: '',
  password: '',
};

const validationSchema = yup.object({
  email: yup.string().email('Enter a valid email').required('Email
is required'),
  password: yup
    .string()
    .min(8, 'Password should be of minimum 8 characters length')
    .required('Password is required'),
});

const fieldsRenderers = {
  email: ({key, ...props}: FieldRendererProps<string>) => (
    <TextField key={key} label="Email" {...props} />
  ),
  password: ({key, ...props}: FieldRendererProps<string>) => (
    <PasswordField key={key} label="Password" {...props} />
  ),
};

interface LoginFormProps {
  onClose: () => void;
}

export const LoginForm = ({ onClose }: LoginFormProps) => {
  const { setUser } = useContext(UserContext);

  const onLoginHandler = async (values: LoginValues) => {
    const response = await AuthService.login(values);
    localStorage.setItem('access-token', response.accessToken);
    setUser(response.user);
    onClose();
  };

  return (
    <Form<LoginValues>
      id="login"
      initialValues={initialValues}
      validationSchema={validationSchema}
```

```
      fieldsRenderers={fieldsRenderers}
      onSubmit={onLoginHandler}
      submitButtonLabel="Sign In"
    />
  );
};
```

RegistrationForm.tsx

```
import * as yup from 'yup';
import AuthService, { RegistrationValues } from
'../../../../services/AuthService';
import { useContext } from 'react';
import { TextField } from '@mui/material';
import { FileField } from
'../../../Shared/components/Fields/FileField/FileField';
import { PasswordField } from
'../../../Shared/components/Fields/PasswordField/PasswordField';
import { UserContext } from '../../../../contexts/UserContext';
import { FieldRendererProps, Form } from
'../../../Shared/components/Form/Form';
import config from '../../../../config';

const initialValues : RegistrationValues = {
  fullName: '',
  email: '',
  password: '',
  avatar: null,
};

const validationSchema = yup.object({
  fullName: yup
    .string()
    .min(3, 'Full name must be more than 3 characters')
    .max(64, 'Full name must be less than 64 characters')
    .matches(/^[A-Za-z\s'-]+$/, 'Full name must not contain
symbols or numbers')
    .required('Full name is required'),
  email: yup
    .string()
    .email('Enter a valid email')
    .matches(/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/,
'Enter a valid email')
    .required('Email is required'),
  password: yup
    .string()
    .min(8, 'Password must be more than 8 characters')
    .max(128, 'Password must not be more than 128 characters')
    .matches(
      /^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)\S+$/,
      'Password must contain at least one uppercase letter, one
lowercase letter, one number, and no spaces',
    )
```

```
    .required('Password is required'),
    avatar: yup
    .mixed<File>()
    .nullable()
    .test("fileType", "Only image files are allowed", (file) => {
      if (!file) return true;
      return file.type.startsWith("image/");
    })
    .test("fileSize", `File size must be less than
${config.MAX_FILE_SIZE / (1 << 20)}MB`, (file) => {
      if (!file) return true;
      return file.size <= config.MAX_FILE_SIZE;
    }),
});

const fieldsRenderers = {
  fullName: ({key, ...props}: FieldRendererProps<string>) => (
    <TextField key={key} label="Full Name" {...props} />
  ),
  email: ({key, ...props}: FieldRendererProps<string>) => (
    <TextField key={key} label="Email" {...props} />
  ),
  password: ({key, ...props}: FieldRendererProps<string>) => (
    <PasswordField key={key} label="Password" {...props} />
  ),
  avatar: ({key, ...props}: FieldRendererProps<File | null |
undefined>) => (
    <FileField key={key} label="Choose file" {...props} />
  ),
};

interface RegistrationFormProps {
  onClose: () => void;
}

export const RegistrationForm = ({ onClose }:
RegistrationFormProps) => {
  const { setUser } = useContext(UserContext);

  const onRegistrationHandler = async (values: RegistrationValues)
=> {
      const response = await AuthService.registration(values);
      localStorage.setItem('access-token', response.accessToken);
      setUser(response.user);
      onClose();
  };

  return (
    <Form<RegistrationValues>
      id="signUp"
      initialValues={initialValues}
      validationSchema={validationSchema}
      fieldsRenderers={fieldsRenderers}
```

```
      onSubmit={onRegistrationHandler}
      submitButtonLabel="Sign up"
    />
  );
};
```

BudgetPage.tsx

```
import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";
import {
  Container,
  Typography,
  Paper,
  List,
  ListItem,
  ListItemText,
  Box,
  Divider,
} from "@mui/material";
import {
  PieChart,
  Pie,
  Cell,
  Tooltip,
  Legend,
  BarChart,
  Bar,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip as RechartsTooltip,
  Legend as RechartsLegend,
} from "recharts";
import Grid from "@mui/material/Grid2";
import LocationService from "../../services/LocationService";

export const BudgetPage = () => {
  const { tripId } = useParams();

  const [expenses, setExpenses] = useState<
    { location: string; category: string; amount: number }[]
  >([]);

  useEffect(() => {
    const fetchData = async () => {
      if (!tripId) return;

      try {
        const locations = await
LocationService.getAllLocationsByTrip(tripId);
        const locationExpenses = [];
```

```
        for (const location of locations) {
          const categories = await
LocationService.getCategoriesForLocation(
            location.id
          );
          categories.forEach((category) => {
            locationExpenses.push({
              location: location.title,
              category: category.category_title,
              amount: category.category_amount,
            });
          });
        }

        setExpenses(locationExpenses);
      } catch (error) {
        console.error("Error fetching data:", error);
      }
    };

    fetchData();
  }, [tripId]);

  const totalExpenses = expenses.reduce(
    (acc, expense) => acc + expense.amount,
    0
  );

  const locationExpenses = expenses.reduce(
    (acc, expense) => {
      acc[expense.location] = acc[expense.location] || [];
      acc[expense.location].push(expense);
      return acc;
    },
    {} as Record<string, { category: string; amount: number }[]>
  );

  const categoryExpenses = expenses.reduce(
    (acc, expense) => {
      const existing = acc.find((item) => item.name ===
expense.category);
      if (existing) {
        existing.value += expense.amount;
      } else {
        acc.push({ name: expense.category, value: expense.amount
});
      }
      return acc;
    },
    [] as { name: string; value: number }[]
  );

  const barChartData = Object.entries(
```

```
    expenses.reduce(
      (acc, expense) => {
        acc[expense.category] = (acc[expense.category] || 0) +
expense.amount;
        return acc;
      },
      {} as Record<string, number>
    )
  ).map(([category, amount]) => ({ category, amount }));

  return (
    <Container>
      <Typography variant="h4" gutterBottom>
        Budget Summary
      </Typography>

      <Paper sx={{ padding: 2, marginBottom: 3 }}>
        <Typography variant="h6">Total Expenses:
${totalExpenses}</Typography>
      </Paper>

      <Grid
        container
        spacing={3}
        direction="row"
        justifyContent="space-between"
      >
        <Grid size={6}>
          <Paper sx={{ padding: 2 }}>
            <Typography variant="h6" gutterBottom>
              Expense Breakdown by Category (Pie Chart)
            </Typography>
            <Box width="100%" height={300}>
              <PieChart width={300} height={300}>
                <Pie
                  data={categoryExpenses}
                  dataKey="value"
                  nameKey="name"
                  cx="50%"
                  cy="50%"
                  outerRadius={120}
                  label
                >
                  {categoryExpenses.map((entry, index) => (
                    <Cell
                      key={`cell-${index}`}
                      fill={["#ff7f50", "#87cefa", "#32cd32",
"#ffa500"][index]}
                    />
                  ))}
                </Pie>
                <Tooltip />
                <Legend />
```

```
              </PieChart>
            </Box>
          </Paper>
        </Grid>

        <Grid size={6}>
          <Paper sx={{ padding: 2 }}>
            <Typography variant="h6" gutterBottom>
              Expense Breakdown by Category (Bar Chart)
            </Typography>
            <Box width="100%" height={300}>
              <BarChart width={500} height={300}
data={barChartData}>
                <CartesianGrid strokeDasharray="3 3" />
                <XAxis dataKey="category" />
                <YAxis />
                <RechartsTooltip />
                <RechartsLegend />
                <Bar dataKey="amount" fill="#8884d8" />
              </BarChart>
            </Box>
          </Paper>
        </Grid>
      </Grid>

      <Grid container spacing={3} sx={{ marginTop: 3 }}>
        <Grid size={12}>
          <Paper sx={{ padding: 2 }}>
            <Typography variant="h6" gutterBottom>
              Expense History by Location
            </Typography>
            <List>
              {Object.entries(locationExpenses).map(
                ([location, expenses], index) => (
                  <>
                    <ListItem key={location}>
                      <ListItemText
                        primary={location}
                        secondary={`Total: $${expenses.reduce(
                          (acc, expense) => acc + expense.amount,
                          0
                        )}`}
                      />
                      <List>
                        {expenses.map((expense, index) => (
                          <ListItem key={index}>
                            <ListItemText
                              primary={`${expense.category}:
$${expense.amount}`}
                            />
                          </ListItem>
                        ))}
                      </List>
```

```
                    </ListItem>
                    {index !==
Object.keys(locationExpenses).length - 1 && (
                      <Divider />
                    )}
                  </>
                )
              )}
            </List>
          </Paper>
        </Grid>
      </Grid>
    </Container>
  );
};
```

HomePage.tsx

```
import { PageContainer } from "@toolpad/core";

export const HomePage = () => {

  return (
    <PageContainer >
      Home
    </PageContainer>
  );
};
```

HotelPage.tsx

```
import * as yup from "yup";
import { Dayjs } from "dayjs";
import { Card, Typography, TextField, Grow, Box, Alert } from
"@mui/material";
import { FieldRendererProps, Form } from
"../Shared/components/Form/Form";
import { DateField } from
"../Shared/components/Fields/DateField/DateField";
import { HotelCard } from
"../ReservationPage/components/HotelCard/HotelCard";
import { useParams } from "react-router-dom";
import HotelService, { Hotel } from "../../services/HotelService";
import { useContext, useEffect, useState } from "react";
import { UserContext } from "../../contexts/UserContext";
import ReservationService from
"../../services/ReservationService";
import { FormikHelpers } from "formik";

interface HotelFormValues {
  fullName: string;
  email: string;
```

```
  birthday: string | Date | Dayjs;
  phone: string;
  country: string;
  startDate: string | Date | Dayjs;
  endDate: string | Date | Dayjs;
}

const validationSchema = yup.object({
  fullName: yup
  .string()
  .required("Name is required")
  .max(255, "Name must be less than 255 characters"),
  email: yup
    .string()
    .email("Enter a valid email")
    .required("Email is required"),
  phone: yup.string().required("Phone is required"),
  birthday: yup
  .date()
  .required("The birthday is required"),
  country: yup.string().required("Country is required"),
  startDate: yup.date().required("The start date is required"),
  endDate: yup
  .date()
  .required("The end date is required")
});

export const HotelPage = () => {
  const { hotelId } = useParams();
  const { tripId } = useParams();
  const [hotel, setHotel] = useState<Hotel>();
  const { user } = useContext(UserContext);

  useEffect(() => {
    if (hotelId) {
      HotelService.getHotel(hotelId).then((hotel) =>
setHotel(hotel));
    }
  }, [hotelId]);

  const initialValues: HotelFormValues = {
    fullName: user?.fullName,
    email: user?.email,
    birthday: user?.birthday,
    phone: user?.phone,
    country: user?.country,
    startDate: "",
    endDate: "",
  };

  const fieldsRenderers = {
    fullName: ({key, ...props}: FieldRendererProps<string>) => (
      <TextField key={key} label="Full Name" {...props} />
```

```
    ),
    email: ({key, ...props}: FieldRendererProps<string>) => (
      <TextField key={key} label="Email" {...props} />
    ),
    birthday: ({key, ...props}: FieldRendererProps<string | Date |
Dayjs>) => (
      <DateField key={key} label="Birthday" {...props} />
    ),
    phone: ({key, ...props}: FieldRendererProps<string>) => (
      <TextField key={key} label="Phone number" {...props} />
    ),
    country: ({key, ...props}: FieldRendererProps<string>) => (
      <TextField key={key} label="Country" {...props} />
    ),
    startDate: ({key, ...props}: FieldRendererProps<string | Date
| Dayjs>) => (
      <DateField key={key} label="Choose start date" {...props} />
    ),
    endDate: ({key, ...props}: FieldRendererProps<string | Date |
Dayjs>) => (
      <DateField key={key} label="Choose end date" {...props} />
    ),
  };

  const onBookHandler = async (values: HotelFormValues, {
resetForm }: FormikHelpers<HotelFormValues>) => {
    try {
      const reservationData = {
        tripId: tripId,
        startDate: values.startDate,
        endDate: values.endDate,
        status: "pending",
        fullName: values.fullName,
        email: values.email,
        phone: values.phone,
        country: values.country,
        hotelId: hotelId,
        userId: user?.id,
      };

      await ReservationService.createReservation(reservationData);
      <Alert severity="success">Reservation successfully
created.</Alert>
      resetForm();
    } catch (error) {
      console.log("Error creating reservation:", error);
    }
  };

  return (
    <Grow in={true} timeout={1000}>
      <Box>
        <HotelCard hotel={hotel} />
```

```
        <Card
          sx={{
            p: 2,
            borderRadius: 2,
          }}
        >
          <Typography variant="h5" gutterBottom
color="text.secondary">
            Please enter your details to complete your hotel
booking
          </Typography>
          <Form<HotelFormValues>
            id="bookingHotel"
            initialValues={initialValues}
            validationSchema={validationSchema}
            fieldsRenderers={fieldsRenderers}
            onSubmit={onBookHandler}
          />
        </Card>
      </Box>
    </Grow>
  );
};
```

HotelsWishlistPage.tsx

```
import { Box, Typography } from "@mui/material";
import Grid from "@mui/material/Grid2";
import { HotelCard } from
"../ReservationPage/components/HotelCard/HotelCard";
import { useContext, useEffect, useState } from "react";
import WishlistService, { WishlistItem } from
"../../services/WishlistService";
import HotelService, { Hotel } from "../../services/HotelService";
import { UserContext } from "../../contexts/UserContext";

export const HotelsWishlistPage = () => {
  const [wishlist, setWishlist] = useState<WishlistItem[]>([]);
  const [hotels, setHotels] = useState<Hotel[]>([]);
  const { user } = useContext(UserContext);

  useEffect(() => {
    if (user?.id) {
      WishlistService.getWishlist(user.id)
        .then((items) => {
          setWishlist(items);

          const hotelIds = items.map((item) => item.hotelId);

          Promise.all(hotelIds.map((id) =>
HotelService.getHotel(id)))
            .then((hotelData) => {
              setHotels(hotelData);
```

```
            })
            .catch((error) =>
              console.error("Error fetching hotel details:",
error)
            );
        })
        .catch((error) => console.error("Error fetching
wishlist:", error));
    }
  }, [user]);

  return (
    <Box sx={{ p: 2 }}>
      <Typography variant="h6" sx={{ mb: 2 }}>
        Your Wishlist
      </Typography>

      <Grid container spacing={2}>
        {wishlist?.length === 0 ? (
          <Typography variant="body1">No items in your
wishlist.</Typography>
        ) : (
          hotels.map((hotel) => (
            <Grid key={hotel.id} size={12}>
              <HotelCard hotel={hotel} />
            </Grid>
          ))
        )}
      </Grid>
    </Box>
  );
};
```

ProfileEditUserCardForm.tsx

```
import { Card, Typography, Grow, TextField } from "@mui/material";
import { FieldRendererProps, Form } from
"../../../Shared/components/Form/Form";
import * as yup from "yup";
import { FileField } from
"../../../Shared/components/Fields/FileField/FileField";
import { useContext } from "react";
import { UserContext } from "../../../../contexts/UserContext";
import UserService from "../../../../services/UserService";

interface ProfileFormValues {
  full_name: string;
  email: string;
  avatar?: string;
}

const validationSchema = yup.object({
  full_name: yup.string().required("Full name is required"),
```

```
  email: yup
    .string()
    .email("Enter a valid email")
    .required("Email is required"),
  avatar: yup.string(),
});

export const ProfileEditUserCardForm = () => {
  const { user, setUser } = useContext(UserContext);

  if (!user) {
    throw new Error("User is not available.");
  }

  const initialValues : ProfileFormValues = {
    full_name: user?.fullName,
    email: user?.email,
    avatar: user?.avatar,
  };

  const fieldsRenderers = {
    full_name: ({key, ...props}: FieldRendererProps<string>) => (
      <TextField key={key} label="Full Name" {...props} />
    ),
    email: ({key, ...props}: FieldRendererProps<string>) => (
      <TextField key={key} label="Email" {...props} />
    ),
    avatar: ({key, ...props}: FieldRendererProps<string |
undefined>) => (
      <FileField key={key} label="Choose file" {...props} />
    ),
  };

  const onEditHandler = async (values: ProfileFormValues) => {
    const formData = new FormData();
    Object.entries(values).forEach(([key, value]) =>
formData.append(key, value));
    const response = await UserService.updateUser(user.id,
formData);

    setUser(response);
  };

  return (
    <Grow in={true} timeout={1000}>
      <Card
        sx={{
          p: 2,
          borderRadius: 2,
        }}
      >
        <Typography variant="h5" gutterBottom
color="text.secondary">
```

```
          Edit profile
        </Typography>
        <Form<ProfileFormValues>
          id="profileEdit"
          initialValues={initialValues}
          validationSchema={validationSchema}
          fieldsRenderers={fieldsRenderers}
          onSubmit={onEditHandler}
        />
      </Card>
    </Grow>
  );
};
```

ProfileUserCard.tsx

```
import { useContext } from "react";
import {
  Avatar,
  Typography,
  Box,
  List,
  ListItem,
  ListItemText,
  Card,
  Grow,
  ListItemIcon,
} from "@mui/material";
import { Email } from "@mui/icons-material";
import { UserContext } from "../../../../contexts/UserContext";

const icons = {
  email: <Email fontSize="small" />,
};

export const ProfileUserCard = () => {
  const { user } = useContext(UserContext);
  const imageUrl = import.meta.env.VITE_MEDIA_URL + "/" +
user?.avatar;

  return (
    <Grow in={true} timeout={1000}>
      <Card
        sx={{
          maxHeight: "100%",
          display: "flex",
          flexDirection: "column",
          alignContent: "center",
          justifyContent: "center",
          alignItems: "center",
          borderRadius: 2,
        }}
      >
```

```
<Box
  sx={{
    bgcolor: "primary.main",
    height: 100,
    width: "100%",
    display: "flex",
    justifyContent: "center",
  }}
>
  <Avatar
    sx={{
      width: 120,
      height: 120,
      top: 30,
      fontSize: "60px",
      bgcolor: "neutral.main",
      color: "#fff",
    }}
    src={imageUrl}
    alt={user?.fullName}
  />
</Box>

<Box sx={{ mt: 8, mb: 2, px: 3, textAlign: "center" }}>
  <Typography gutterBottom variant="h4">
    {user?.fullName}
  </Typography>
  <List>
    {Object.keys(icons).map((item) => (
      <ListItem disablePadding key={item} sx={{ mb: 1 }}>
        <ListItemIcon
          sx={{
            justifyContent: "center",
            minWidth: 25,
            mr: 1,
            p: 1,
            borderRadius: 100,
            bgcolor: "neutral.main",
          }}
        >
          {icons[item]}
        </ListItemIcon>
        <ListItemText primary={user?.[item] || "-"} />
      </ListItem>
    ))}
  </List>
</Box>
<Box
  sx={{
    height: 50,
    width: "100%",
    display: "flex",
    justifyContent: "center",
```

```
            alignItems: "center",
            borderRadius: "0px  0px 12px 12px",
          }}
        >
          <Typography variant="body1" sx={{ color:
"text.secondary" }}>
            Created at - {new
Date(user?.createdAt).toLocaleDateString()}
          </Typography>
        </Box>
      </Card>
    </Grow>
  );
};
```

ProfilePage.tsx

```
import Grid from "@mui/material/Grid2";
import { ProfileUserCard } from
"./components/ProfileUserCard/ProfileUserCard";
import { ProfileEditUserCardForm } from
"./components/ProfileEditUserCardForm/ProfileEditUserCardForm";
import { Typography } from "@mui/material";
import { HotelCard } from
"../ReservationPage/components/HotelCard/HotelCard";
import ReservationService, { Reservation } from
"../../services/ReservationService";
import { useContext, useEffect, useState } from "react";
import { UserContext } from "../../contexts/UserContext";
import HotelService, { Hotel } from "../../services/HotelService";

export const ProfilePage = () => {
  const { user } = useContext(UserContext);
  const [hotels, setHotels] = useState<Hotel[] | null>(null);
  const [reservations, setReservations] = useState<Reservation[] |
null>(null);

  useEffect(() => {
    if (user) {
      ReservationService.getReservationsByUser(user.id)
        .then((reservations) => {
          setReservations(reservations);
          return Promise.all(
            reservations.map((reservation) =>
              HotelService.getHotel(reservation.hotelId)
            )
          );
        })
        .then((loadedHotels) => {
          setHotels(loadedHotels);
        })
        .catch((error) => {
          console.error("Error fetching hotels:", error);
```

```
        });
    }
  }, [user]);

  if (!user) {
    return (
      <Typography variant="h6" color="error" align="center">
        User data is unavailable. Please log in.
      </Typography>
    );
  }

  return (
    <Grid container spacing={1} justifyContent="center">
      <Grid size={4}>
        <ProfileUserCard />
      </Grid>
      <Grid size={8}>
        <ProfileEditUserCardForm />
      </Grid>
      {hotels && reservations ? (
        <Grid size={12}>
          <Typography variant="h6" gutterBottom>My
reservations</Typography>
          {hotels.map((hotel, index) => (
            <HotelCard key={hotel.id} hotel={hotel}
reservationInfo={{
              startDate: reservations[index]?.startDate,
              endDate: reservations[index]?.endDate,
              fullName: reservations[index]?.fullName,
            }}/>
          ))}
        </Grid>
      ) : (
        <Typography variant="body1" color="textSecondary"
align="center">
          No reservations found.
        </Typography>
      )}
    </Grid>
  );
};
```

HotelCard.tsx

```
import {
  Box,
  Typography,
  Card,
  CardContent,
  CardMedia,
  Rating,
  Button,
```

```
  Divider,
} from "@mui/material";
import { ReactNode } from "react";
import { Link, useParams } from "react-router-dom";

interface HotelCardProps {
  hotel: {
    id: number;
    image?: string | null;
    title: string;
    location: string;
    guests?: number;
    description?: string | null;
    pricePerNight: number;
    rating?: number | null;
    reviews?: number | null;
    reservationInfo?: [] | null;
  };
  children?: ReactNode;
}

export const HotelCard = ({
  hotel,
  reservationInfo,
  children,
}: HotelCardProps) => {
  const imageUrl = import.meta.env.VITE_HOTELS_MEDIA_URL + "/" +
hotel?.image;
  const { tripId } = useParams();

  return (
    <Card
      sx={{
        display: "flex",
        flexDirection: { xs: "column", sm: "row" },
        boxShadow: 3,
        borderRadius: 2,
        mb: 2,
        overflow: "hidden",
      }}
    >
      <CardMedia
        component="img"
        sx={{
          width: { xs: "100%", sm: 300 },
          height: { xs: 200, sm: "auto" },
          objectFit: "cover",
        }}
        image={imageUrl ||
"https://via.placeholder.com/300x200?text=No+Image"}
        alt={hotel?.title}
      />
```

```
      <CardContent
        sx={{
          display: "flex",
          flexDirection: "column",
          justifyContent: "space-between",
          flex: 1,
          p: 2,
        }}
      >
        <Box>
          <Typography variant="subtitle2" color="textSecondary">
            {hotel?.location}
          </Typography>

          <Typography variant="h6" gutterBottom>
            {hotel?.title}
          </Typography>

          <Typography variant="body2" color="textSecondary">
            Max Guests: {hotel?.guests}
          </Typography>

          {hotel?.description && (
            <Typography
              variant="body2"
              color="textSecondary"
              sx={{
                mt: 1,
                overflow: "hidden",
                textOverflow: "ellipsis",
                display: "-webkit-box",
                WebkitBoxOrient: "vertical",
                WebkitLineClamp: 3,
              }}
            >
              {hotel?.description}
            </Typography>
          )}

          <Box sx={{ display: "flex", alignItems: "center", mt: 2
}}>
            <Rating
              value={hotel?.rating || 0}
              readOnly
              size="small"
              precision={0.1}
            />
            <Typography variant="body2" sx={{ ml: 0.5 }}>
              {hotel?.rating
                ? `${hotel?.rating} (${hotel?.reviews || 0}
reviews)`
                : "No reviews yet"}
            </Typography>
```

```
          </Box>
        </Box>
        <Box sx={{ mt: 2 }}>
          {reservationInfo ? (
            <>
              <Divider />
              <Typography variant="subtitle1">
                Reservation Date:{" "}
                <b>
                  {reservationInfo.startDate} -
{reservationInfo.endDate}
                </b>
              </Typography>
              <Typography variant="subtitle1">
                Reserved by: <b>{reservationInfo.fullName}</b>
              </Typography>
            </>
          ) : (
            <Button
              variant="contained"
              component={Link}
              to={`/trips/${tripId}/reservation/${hotel?.id}`}
              sx={{ minWidth: "150px" }}
            >
              Book Now
            </Button>
          )}
        </Box>
      </CardContent>
      <Box
        sx={{
          display: "flex",
          flexDirection: "column",
          alignItems: "flex-end",
          justifyContent: "space-between",
          p: 2,
          minWidth: { xs: "100%", sm: 150 },
          backgroundColor: "background.paper",
        }}
      >
        {children}
        <Typography
          variant="h6"
          color="primary"
          sx={{ mt: "auto", fontWeight: "bold" }}
        >
          ${hotel?.pricePerNight}/night
        </Typography>
      </Box>
    </Card>
  );
};
```

ReservationPage.tsx

```
import {
  Box,
  Typography,
  IconButton,
  Button,
  FormControl,
  InputLabel,
  Select,
  MenuItem,
} from "@mui/material";
import Grid from "@mui/material/Grid2";
import FavoriteBorderIcon from "@mui/icons-
material/FavoriteBorder";
import { useContext, useEffect, useState } from "react";
import { HotelCard } from "./components/HotelCard/HotelCard";
import { Link, useParams, useSearchParams } from "react-router-
dom";
import HotelService, { Hotel } from "../../services/HotelService";
import { Favorite } from "@mui/icons-material";
import { DatePicker } from "@mui/x-date-pickers";
import WishlistService, { WishlistItem } from
"../../services/WishlistService";
import { UserContext } from "../../contexts/UserContext";
import { LocationsAutocomplete } from
"../TripPage/components/LocationsAutocomplete/LocationsAutocomplet
e";
import { Dayjs } from "dayjs";

export const ReservationPage = () => {
  const [hotels, setHotels] = useState<Hotel[]>([]);
  const [wishlist, setWishlist] = useState<WishlistItem[]>([]);
  const [chosenLocation, setChosenLocation] =
useState<Location>();
  const [searchParams, setSearchParams] = useSearchParams();
  const sortValue = searchParams.get("sort");
  const { user } = useContext(UserContext);
  const [startDate, setStartDate] = useState<Dayjs | null>(null);
  const [endDate, setEndDate] = useState<Dayjs | null>(null);
  const { tripId } = useParams();

  useEffect(() => {
    if (user?.id) {
      const filters = {
        location: chosenLocation?.title,
        startDate,
        endDate,
        sortBy: sortValue || "price",
      };

      HotelService.getAllHotels(filters)
        .then((hotels) => setHotels(hotels))
```

```
        .catch((error) => console.error("Error fetching hotels:",
error));

      WishlistService.getWishlist(user.id)
        .then((items) => setWishlist(items))
        .catch((error) => console.error("Error fetching
wishlist:", error));
    } else {
      console.log("User doesn't exist or not authenticated");
    }
  }, [user, chosenLocation, startDate, endDate, sortValue]);

  const addToWishlist = (hotelId: number) => {
    if (user?.id) {
      WishlistService.addToWishlist(user.id, hotelId)
        .then((newItem) => setWishlist((prev) => [...prev,
newItem]))
        .catch((error) => console.error("Error adding to
wishlist:", error));
    }
  };

  const removeFromWishlist = (hotelId: number) => {
    if (user?.id) {
      WishlistService.removeFromWishlist(user.id, hotelId)
        .then(() =>
          setWishlist((prev) => prev.filter((item) => item.hotelId
!== hotelId))
        )
        .catch((error) =>
          console.error("Error removing from wishlist:", error)
        );
    }
  };

  const handleLocationSelect = (selectedLocation: Location) => {
    setChosenLocation(selectedLocation);
  };

  const handleSortChange = (e: any) => {
    setSearchParams({ sort: e.target.value });
  };

  return (
    <Box sx={{ p: 2 }}>
      <Box sx={{ display: "flex", justifyContent: "space-between"
}}>
        <Typography variant="h6" sx={{ mb: 2 }}>
          Choose from {hotels.length} available hotels to book
your perfect
          stay!
        </Typography>
        <Button
```

```
    variant="contained"
    startIcon={<Favorite />}
    component={Link}
    to={`/trips/${tripId}/reservation/wishlist`}
  >
    Open Wishlist
  </Button>
</Box>

<Box
  sx={{
    display: "flex",
    flexDirection: "row",
    mb: 3,
    width: "100%",
  }}
>
  <Box sx={{ mr: 2, width: "200px" }}>
    <LocationsAutocomplete
      onLocationSelect={handleLocationSelect}
      chosenLocations={chosenLocation}
    />
  </Box>
  <Box sx={{ mr: 2, width: "200px" }}>
    <DatePicker
      label="Check-in Date"
      onChange={(newValue) => setStartDate(newValue)}
      slotProps={{
        textField: { fullWidth: true },
      }}
    />
  </Box>
  <Box sx={{ mr: 2, width: "200px" }}>
    <DatePicker
      label="Check-out Date"
      onChange={(newValue) => setEndDate(newValue)}
      slotProps={{
        textField: { fullWidth: true },
      }}
    />
  </Box>
  <Box sx={{ mr: 2, width: "100px" }}>
    <FormControl fullWidth>
      <InputLabel id="sort-label">Sort By</InputLabel>
      <Select
        labelId="sort"
        value={sortValue || ""}
        onChange={handleSortChange}
        autoWidth
        label="Sorting"
        sx={{ width: "100%" }}
      >
        <MenuItem value="price">Price</MenuItem>
```

```
              <MenuItem value="rating">Rating</MenuItem>
            </Select>
          </FormControl>
        </Box>
      </Box>

      <Grid container spacing={2}>
        {hotels.map((hotel) => (
          <Grid key={hotel.id} size={12}>
            <HotelCard hotel={hotel}>
              <IconButton
                color="error"
                onClick={() =>
                  wishlist.some((item) => item.hotelId ===
hotel.id)
                    ? removeFromWishlist(hotel.id)
                    : addToWishlist(hotel.id)
                }
              >
                {wishlist.some((item) => item.hotelId ===
hotel.id) ? (
                  <Favorite />
                ) : (
                  <FavoriteBorderIcon />
                )}
              </IconButton>
            </HotelCard>
          </Grid>
        ))}
      </Grid>
    </Box>
  );
};
```

Router.tsx

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import "./index.css";
import { createBrowserRouter, RouterProvider } from "react-router-
dom";
import { App } from "../../App";
import Layout from "../../layouts/dashboard";
import { HomePage } from "../HomePage/HomePage";
import { ProfilePage } from "../ProfilePage/ProfilePage";
import { TripsPage } from "../TripsPage/TripsPage";
import { TripPage } from "../TripPage/TripPage";
import { ReservationPage } from
"../ReservationPage/ReservationPage";
import { HotelPage } from "../HotelPage/HotelPage";
import { HotelsWishlistPage } from
"../HotelsWishlistPage/HotelsWishlistPage";
import { SuitcasePage } from "../SuitcasePage/SuitcasePage";
```

```
import { BudgetPage } from "../BudgetPage/BudgetPage";


const router = createBrowserRouter([
  {
    Component: App,
    children: [
      {
        path: "/",
        Component: Layout,
        children: [
          {
            path: "home",
            Component: HomePage,
          },
          {
            path: "profile",
            Component: ProfilePage,
          },
          {
            path: "trips/",
            children: [
              {
                index: true,
                Component: TripsPage,
              },
              {
                path: ":tripId",
                Component: TripPage,
              },
            ],
          },
          {
            path: "reservation/",
            children: [
              {
                index: true,
                Component: ReservationPage,
              },
              {
                path: ":hotelId",
                Component: HotelPage,
              },
              {
                path: "wishlist",
                Component: HotelsWishlistPage,
              },
            ],
          },
          {
            path: "suitcase",
            Component: SuitcasePage,
          },
```

```
          {
            path: "budget",
            Component: BudgetPage,
          },
        ],
      },
    ],
  },
]);

createRoot(document.getElementById("root")!).render(
  <StrictMode>
    <RouterProvider router={router} />
  </StrictMode>
);
```

DeleteModal.tsx

```
import {
  Button,
  Dialog,
  DialogTitle,
  DialogContent,
  DialogContentText,
  DialogActions,
  IconButton,
  Typography,
} from '@mui/material';
import { Delete } from '@mui/icons-material';

export const DeleteModal = ({ open, onClose, onDelete, title,
description }) => {
  return (
    <Dialog
      open={open}
      onClick={onClose}
      maxWidth="sm"
      PaperProps={{
        sx: { borderRadius: 2, bgcolor: 'message.primary', px: 2,
py: 2 },
      }}
    >
      <DialogTitle variant="h5" sx={{ display: 'flex', alignItems:
'center', px: 1 }}>
        <IconButton>
          <Delete fontSize="large" color='error' />
        </IconButton>
        <Typography variant="title" color="text.primary">
          {title}
        </Typography>
      </DialogTitle>
      <DialogContent>
        <DialogContentText sx={{ mb: 1 }}>
```

```
          <Typography color="text.primary">
            {description} <strong> This process cannot be
undone!</strong>
          </Typography>
        </DialogContentText>
      </DialogContent>
      <DialogActions>
        <Button variant="outlined" onClick={onClose}>
          Cancel
        </Button>
        <Button variant="contained" color="error"
onClick={onDelete}>
          Delete
        </Button>
      </DialogActions>
    </Dialog>
  );
};
```

ErrorPage.ts

```
import Typography from '@mui/material/Typography';
import Box from '@mui/material/Box';
import ErrorPageImage from '../../../../icons/ErrorPage.svg';

export const ErrorPage = () => {
  return (
    <Box
      sx={{
        display: 'flex',
        flexDirection: 'column',
        alignItems: 'center',
        justifyContent: 'center',
      }}
    >
      <Box sx={{ height: 480, ml: -7, mb: -1 }}>
        <img src={ErrorPageImage} alt="Error page" height="100%"
width="100%" />
      </Box>
      <Typography variant="h5" sx={{ ml: -6 }}>
        The page you were looking for does not exist!
      </Typography>
    </Box>
  );
};
```

DateField.tsx

```
import { DateTimePicker } from "@mui/x-date-
pickers/DateTimePicker";
import dayjs, { Dayjs } from "dayjs";
```

```
interface DateFieldProps {
  name: string;
  label?: string;
  time?: boolean;
  value?: string | Date | Dayjs;
  onChange: (event: React.ChangeEvent) => void;
  error?: boolean;
  helperText?: React.ReactNode;
}

export const DateField = ({
  name,
  label,
  time = false,
  value,
  onChange,
  ...props
}: DateFieldProps) => {
  const handleChange = (value: Dayjs | null) => {
    onChange({
      currentTarget: { name, value },
    } as unknown as React.ChangeEvent);
  };

  return (
    <DateTimePicker
      label={label}
      onChange={handleChange}
      value={dayjs(value)}
      slotProps={{ textField: props }}
      views={
        time ? ["year", "day", "hours", "minutes", "seconds"] :
["year", "day"]
      }
      sx={{ my: 2, width: "100%" }}
    />
  );
};
```

FileField.tsx

```
import { InputAdornment } from "@mui/material";
import { MuiFileInput } from "mui-file-input";
import FileUploadIcon from "@mui/icons-material/FileUpload";

interface FileFieldProps {
  onChange: (event: React.ChangeEvent) => void;
  name: string;
  [key: string]: unknown;
}

export const FileField = ({ name, onChange, ...props }:
FileFieldProps) => {
```

```
  const handleChange = (value: File | null) => {
    onChange({ currentTarget: { name, value } } as unknown as
React.ChangeEvent);
  };

  return (
    <MuiFileInput
      InputProps={{
        startAdornment: (
          <InputAdornment position="start">
            <FileUploadIcon />
          </InputAdornment>
        ),
      }}
      placeholder="Insert a file"
      getInputText={(value) =>  value ? value.name || "File
selected" : "Insert a file"}
      onChange={handleChange}
      {...props}
    />
  );
};
```

PasswordField.tsx

```
import { TextField, IconButton, InputAdornment, TextFieldProps }
from '@mui/material';
import { Visibility, VisibilityOff } from '@mui/icons-material';
import { useState } from 'react';

export const PasswordField = (props: Omit<TextFieldProps, 'type' |
'InputProps'>) => {
  const [showPassword, setShowPassword] = useState(false);

  const handleClickShowPassword = () => setShowPassword((show) =>
!show);

  const handleMouseDownPassword = (event:
React.MouseEvent<HTMLButtonElement>) => {
    event.preventDefault();
  };

  return (
    <TextField
      type={showPassword ? 'text' : 'password'}
      InputProps={{
        endAdornment: (
          <InputAdornment position="end">
            <IconButton onClick={handleClickShowPassword}
onMouseDown={handleMouseDownPassword}>
              {showPassword ? <Visibility /> : <VisibilityOff />}
            </IconButton>
          </InputAdornment>
```

```
        ),
      }}
      {...props}
    />
  );
};
```

TimeCustomField.tsx

```
import dayjs from 'dayjs';
import { TimeField } from '@mui/x-date-pickers';

export const TimeCustomField = ({ onChange, ...props }) => {
  const handlerChange = (value) => {
    onChange({ currentTarget: { name: props.name, value: value }
});
  };
  return (
    <TimeField
      label={props.label}
      onChange={handlerChange}
      value={dayjs(props.value)}
      format="HH:mm:ss"
      sx={{ my: 1, width: '100%' }}
    />
  );
};
```

Form.tsx

```
import { Alert, Button, LinearProgress } from "@mui/material";
import { useFormik, FormikValues, FormikHelpers } from "formik";
import { ObjectSchema } from "yup";
import { ReactNode } from "react";
import useLoading from "../../../../hooks/useLoading";

export type FieldRendererProps<T> = {
  id: string;
  key: string;
  name: string;
  fullWidth: boolean;
  margin: "normal";
  value: T;
  onChange: (e: React.ChangeEvent) => void;
  error?: boolean;
  helperText?: ReactNode;
};

interface FormProps<FormValues extends FormikValues> {
  id: number | string;
  initialValues: FormValues;
```

```
  validationSchema: ObjectSchema<FormValues>;
  fieldsRenderers: {
    [K in keyof FormValues]: (
      options: FieldRendererProps<FormValues[K]>
    ) => ReactNode;
  };
  onSubmit: (
    values: FormValues,
    formikHelpers: FormikHelpers<FormValues>
  ) => void | Promise<unknown>;
  submitButtonLabel?: string;
  children?: ReactNode;
}

export const Form = <FormValues extends FormikValues>({
  id,
  initialValues,
  validationSchema,
  fieldsRenderers,
  onSubmit,
  submitButtonLabel = "Submit",
  children,
}: FormProps<FormValues>) => {
  const { isLoading, error, execute, reset } = useLoading(
    onSubmit as (...args: unknown[]) => Promise<void> | void
  );
  const formik = useFormik<FormValues>({
    initialValues,
    validationSchema,
    onSubmit: execute,
  });

  return (
    <form
      onSubmit={formik.handleSubmit}
      style={{
        display: "flex",
        flexDirection: "column",
        alignItems: "center",
      }}
    >
      {Object.keys(fieldsRenderers).map((item: string) => {
        return fieldsRenderers[item]({
          id: id + "-" + item,
          key: id + "-" + item,
          name: item,
          fullWidth: true,
          margin: "normal",
          value: formik.values[item] || undefined,
          onChange: formik.handleChange,
          error: formik.touched[item] &&
Boolean(formik.errors[item]),
          helperText:
```

```
            formik.touched[item] && (formik.errors[item] as
ReactNode),
        });
      })}
      {children}
      {error && (
        <Alert severity="error" onClose={reset}>
          {error.message}
        </Alert>
      )}
      {isLoading && <LinearProgress color="inherit" />}
      <Button
        type="submit"
        variant="contained"
        disabled={isLoading}
        size="large"
        sx={{ mt: 4, px: 8, borderRadius: "6px" }}
      >
        {submitButtonLabel}
      </Button>
    </form>
  );
};
```

Modal.tsx

```
import {
  Box,
  Button,
  Dialog,
  DialogActions,
  DialogContent,
  DialogTitle,
} from "@mui/material";
import { ReactNode } from "react";

interface ModalProps {
  title: string;
  open: boolean;
  onClose: () => void;
  children: ReactNode;
}

export const Modal = ({ title, open, onClose, children }:
ModalProps) => {
  return (
    <Dialog
      open={open}
      onClose={onClose}
      scroll={"paper"}
      maxWidth="xs"
      fullWidth
      PaperProps={{
```

```
        sx: { borderRadius: 2 },
      }}
    >
      <Box sx={{ display: "flex", justifyContent: "center" }}>
        <DialogTitle sx={{ pt: 2, pb: 0, fontSize: "24px" }}
color="primary">
          <strong>{title}</strong>
        </DialogTitle>
      </Box>
      <DialogContent sx={{ py: 0 }}>{children}</DialogContent>
      <DialogActions>
        <Button onClick={onClose}>Close</Button>
      </DialogActions>
    </Dialog>
  );
};
```

PopupMenu.tsx

```
import { Menu } from "@mui/material";
import { ReactNode } from "react";

const paperProps = {
  elevation: 0,
  sx: {
    overflow: "visible",
    filter: "drop-shadow(0px 2px 8px rgba(0,0,0,0.32))",
    mt: 1.5,
    "&:before": {
      content: '""',
      display: "block",
      position: "absolute",
      top: 0,
      right: 15,
      width: 10,
      height: 10,
      bgcolor: "background.paper",
      transform: "translateY(-50%) rotate(45deg)",
      zIndex: 0,
    },
  },
};

interface PopupMenuProps {
  anchorEl: HTMLElement | null;
  onClose: () => void;
  children: ReactNode;
}

export const PopupMenu = ({ anchorEl, onClose, children }:
PopupMenuProps) => {
  return (
    <Menu
```

```
      id="menu-appbar"
      anchorEl={anchorEl}
      keepMounted
      open={!!anchorEl}
      onClose={onClose}
      PaperProps={paperProps}
      transformOrigin={{
        horizontal: "right",
        vertical: "top",
      }}
      anchorOrigin={{
        horizontal: "right",
        vertical: "bottom",
      }}
    >
      {children}
    </Menu>
  );
};
```

SuitcasePage.tsx

```
import { useEffect, useState } from "react";
import {
  Container,
  IconButton,
  List,
  ListItem,
  Checkbox,
  Typography,
  TextField,
  Button,
  Tooltip,
  Card,
} from "@mui/material";
import { Add, Delete, CheckCircle } from "@mui/icons-material";
import TripService from "../../services/TripService";
import { useParams } from "react-router-dom";

export const SuitcasePage = () => {
  const { tripId } = useParams();
  const [tasks, setTasks] = useState<
    { id: number; title: string; completed: boolean }[]
  >([]);
  const [inputValue, setInputValue] = useState("");

  useEffect(() => {
    const fetchTripItems = async () => {
      try {
        const items = await TripService.getTripItems(tripId);
        setTasks(items.map((item) => ({ ...item, completed: false
}))));
      } catch (error) {
```

```
        console.error("Error fetching trip items:", error);
      }
    };
    fetchTripItems();
  }, [tripId]);

  const handleAddTask = async () => {
    if (inputValue.trim() === "") return;

    try {
      const newTask = await TripService.addTripItem(tripId,
inputValue);
      setTasks([...tasks, { ...newTask, completed: false }]);
      setInputValue("");
    } catch (error) {
      console.error("Error adding trip item:", error);
    }
  };

  const handleDeleteTask = async (id: number) => {
    try {
      await TripService.deleteTripItem(id);
      setTasks(tasks.filter((task) => task.id !== id));
    } catch (error) {
      console.error("Error deleting trip item:", error);
    }
  };

  const handleEditTask = async (id: number, newText: string) => {
    try {
      const updatedTask = await TripService.updateTripItem(id, {
        title: newText,
      });
      setTasks((prevTasks) =>
        prevTasks.map((task) =>
          task.id === id ? { ...task, title: updatedTask.title } :
task
        )
      );
    } catch (error) {
      console.error("Error editing trip item:", error);
    }
  };

  const toggleTaskCompletion = (id: number) => {
    setTasks((prevTasks) =>
      prevTasks.map((task) =>
        task.id === id ? { ...task, completed: !task.completed } :
task
      )
    );
  };
```

```
return (
  <Container maxWidth="sm" sx={{ mt: 4 }}>
    <Typography variant="h4" gutterBottom align="center">
      Adventure Checklist
    </Typography>
    <TextField
      fullWidth
      label="Add your travel item"
      value={inputValue}
      onChange={(e) => setInputValue(e.target.value)}
      onKeyPress={(e) => {
        if (e.key === "Enter") handleAddTask();
      }}
      sx={{ mb: 2 }}
    />
    <Button
      fullWidth
      variant="contained"
      color="primary"
      startIcon={<Add />}
      onClick={handleAddTask}
      sx={{
        mb: 3,
        textTransform: "none",
        fontSize: "20px",
        borderRadius: "12px",
      }}
    >
      Add item
    </Button>
    <Card
      sx={{
        p: 1,
        boxShadow: "0px 4px 10px rgba(0, 0, 0, 0.1)",
        borderRadius: "12px",
      }}
    >
      <List>
        {tasks.map((task, index) => (
          <ListItem
            key={task.id}
            divider={!(index === tasks.length - 1)}
            sx={{
              borderRadius: "12px",
              transition: "transform 0.2s",
              "&:hover": { transform: "scale(1.02)" },
            }}
          >
            <Checkbox
              edge="start"
              checked={task.completed}
              onChange={() => toggleTaskCompletion(task.id)}
              icon={<CheckCircle />}
```

```
                checkedIcon={<CheckCircle color="success" />}
              />
              <TextField
                fullWidth
                defaultValue={task.title}
                onBlur={(e) => handleEditTask(task.id,
e.target.value)}
                sx={{
                  textDecoration: task.completed ? "line-through"
: "none",
                  cursor: "pointer",
                }}
              />
              <Tooltip title="Delete task">
                <IconButton
                  edge="end"
                  aria-label="delete"
                  onClick={() => handleDeleteTask(task.id)}
                >
                  <Delete />
                </IconButton>
              </Tooltip>
            </ListItem>
          ))}
        </List>
      </Card>
    </Container>
  );
};
```

BudgetModal.tsx

```
import React, { useState, useEffect } from "react";
import {
  Modal,
  TextField,
  Button,
  Box,
  Typography,
  List,
  ListItem,
  ListItemText,
  IconButton,
  ListItemAvatar,
  Avatar,
  Divider,
} from "@mui/material";
import LocationService from
"../../../../services/LocationService";
import { Delete } from "@mui/icons-material";

interface BudgetModalProps {
  open: boolean;
```

```
  onClose: () => void;
  locationId: number;
}

interface BudgetCategory {
  id?: number;
  title: string;
  amount: number;
}

export const BudgetModal = ({
  open,
  onClose,
  locationId,
}: BudgetModalProps) => {
  const [category, setCategory] = useState<BudgetCategory>({
    title: "",
    amount: 0,
  });
  const [categories, setCategories] =
useState<BudgetCategory[]>([]);

  const fetchCategories = () => {
    LocationService.getCategoriesForLocation(locationId)
      .then((response) => {
        setCategories(response);
      })
      .catch((error) => {
        console.error("Error fetching categories:", error);
      });
  };

  useEffect(() => {
    if (open) {
      fetchCategories();
    }
  }, [open]);

  const handleInputChange = (e:
React.ChangeEvent<HTMLInputElement>) => {
    const { name, value } = e.target;
    setCategory((prevCategory) => ({
      ...prevCategory,
      [name]: value,
    }));
  };

  const handleSubmit = async () => {
    if (category.title && category.amount > 0) {
      try {
        await
LocationService.createCategoryForLocation(locationId, category);
        setCategory({ title: "", amount: 0 });
```

```
        fetchCategories();
      } catch (error) {
        console.error("Error creating category:", error);
      }
    }
  };

  const handleDelete = async (categoryId: number) => {
    try {
      await LocationService.deleteCategoryFromLocation(locationId,
categoryId);
      fetchCategories();
    } catch (error) {
      console.error("Error deleting category:", error);
    }
  };

  return (
    <Modal open={open} onClose={onClose}>
      <Box
        sx={{
          position: "absolute",
          top: "50%",
          left: "50%",
          transform: "translate(-50%, -50%)",
          width: 400,
          bgcolor: "background.paper",
          borderRadius: 2,
          boxShadow: 24,
          p: 3,
        }}
      >
        <Typography variant="h6" gutterBottom>
          Budget Categories
        </Typography>

        <List>
          {categories.map((category, index) => (
            <>
              <ListItem
                key={category.category_id}
                secondaryAction={
                  <IconButton
                    edge="end"
                    aria-label="delete"
                    onClick={() =>
handleDelete(category.category_id)}
                  >
                    <Delete />
                  </IconButton>
                }
              >
                <ListItemAvatar>
```

```
                  <Avatar>{index + 1}</Avatar>
                </ListItemAvatar>
                <ListItemText
                  primary={category.category_title}
                  secondary={`Amount:
${category.category_amount}`}
                />
              </ListItem>
              {index < categories.length - 1 && <Divider />}
            </>
          ))}
        </List>

        <Typography variant="subtitle1" gutterBottom>
          Add New Category
        </Typography>
        <TextField
          label="Category Title"
          variant="outlined"
          fullWidth
          name="title"
          value={category.title}
          onChange={handleInputChange}
          sx={{ mb: 2 }}
        />
        <TextField
          label="Amount"
          variant="outlined"
          fullWidth
          type="number"
          name="amount"
          value={category.amount}
          onChange={handleInputChange}
          sx={{ mb: 2 }}
        />
        <Box sx={{ display: "flex", justifyContent: "space-
between" }}>
          <Button variant="outlined" onClick={onClose}>
            Cancel
          </Button>
          <Button variant="contained" onClick={handleSubmit}>
            Save
          </Button>
        </Box>
      </Box>
    </Modal>
  );
};
```

LocationCreationModal.tsx

```
import * as yup from "yup";
import { TextField } from "@mui/material";
```

```
import LocationService, {
  Location,
} from "../../../../services/LocationService";
import { Modal } from "../../../Shared/components/Modal/Modal";
import { FieldRendererProps, Form } from
"../../../Shared/components/Form/Form";

interface LocationFormValues {
  title: string;
  latitude: string;
  longitude: string;
}

const validationSchema = yup.object().shape({
  title: yup
    .string()
    .required("Name is required")
    .max(255, "Name must be less than 255 characters"),
  latitude: yup
    .string()
    .required("Latitude is required")
    .test(
      "is-latitude",
      "Latitude must be a valid number between -90 and 90",
      (value) => {
        const num = parseFloat(value || "");
        return !isNaN(num) && num >= -90 && num <= 90;
      }
    ),
  longitude: yup
    .string()
    .required("Longitude is required")
    .test(
      "is-longitude",
      "Longitude must be a valid number between -180 and 180",
      (value) => {
        const num = parseFloat(value || "");
        return !isNaN(num) && num >= -180 && num <= 180;
      }
    ),
});

const fieldsRenderers = {
  title: ({ key, ...props }: FieldRendererProps<string>) => (
    <TextField key={key} label="Title" {...props} />
  ),
  latitude: ({ key, ...props }: FieldRendererProps<string>) => (
    <TextField key={key} label="Latitude" {...props} />
  ),
  longitude: ({ key, ...props }: FieldRendererProps<string>) => (
    <TextField key={key} label="Longitude" {...props} />
  ),
};
```

```
interface LocationCreationModalProps {
  modalTitle: string;
  open: boolean;
  onClose: () => void;
  onSubmit?: (location: Location) => void;
  item?: Partial<Location>;
}

export const LocationCreationModal = ({
  modalTitle,
  open,
  onClose,
  onSubmit,
  item,
}: LocationCreationModalProps) => {
  const initialValues: LocationFormValues = {
    title: item?.title || "",
    latitude: `${item?.latitude || ""}`,
    longitude: `${item?.longitude || ""}`,
  };

  const onCreateHandler = async (values: LocationFormValues) => {
    const castedValues = {
      title: values.title,
      latitude: +values.latitude,
      longitude: +values.longitude,
      trip_id: item?.trip_id
    };
    const response = item?.id
      ? await LocationService.update(item.id, castedValues)
      : await LocationService.create(castedValues);
    onSubmit?.(response);
    onClose();
  };

  return (
    <Modal title={modalTitle} open={open} onClose={onClose}>
      <Form<LocationFormValues>
        id="tripCreateUpdate"
        initialValues={initialValues}
        validationSchema={validationSchema}
        fieldsRenderers={fieldsRenderers}
        onSubmit={onCreateHandler}
      />
    </Modal>
  );
};
```

LocationLineCard.tsx

```
import {
  ListItem,
```

```
  ListItemButton,
  ListItemText,
} from "@mui/material";
import { Location } from
"../../../../../../services/LocationService";

interface LocationLineCardProps {
  location: Location;
}

export const LocationLineCard = ({
  location,
  ...props
}: LocationLineCardProps) => {
  return (
    <>
      <ListItem disablePadding>
        <ListItemButton {...props}>
          <ListItemText primary={location.title} />
        </ListItemButton>
      </ListItem>
    </>
  );
};
```

LocationsAutocomplete.tsx

```
import { Autocomplete, TextField, InputAdornment } from
"@mui/material";
import { useState } from "react";
import { AddLocationAlt } from "@mui/icons-material";
import { LocationLineCard } from
"./components/LocationLineCard/LocationLineCard";
import LocationService, {
  Location,
} from "../../../../services/LocationService";

interface LocationsAutocompleteProps {
  label?: string;
  onLocationSelect?: (locations: Partial<Location>) => void;
}

const preventErase = (event: React.KeyboardEvent) => {
  if (event.key === "Backspace" || event.key === "Delete") {
    event.stopPropagation();
  }
};

export const LocationsAutocomplete = ({
  label,
  onLocationSelect,
}: LocationsAutocompleteProps) => {
  const [inputValue, setInputValue] = useState("");
```

```
  const [options, setOptions] = useState<Location[]>([]);

  const fetchLocations = async (query: string) => {
    if (!query) return;
    setOptions(
      (await LocationService.geocode(query)).map((result) => ({
        title: result.place_name,
        latitude: result.center[1],
        longitude: result.center[0],
      }))
    );
  };

  const handleInputChange = (
    event: React.SyntheticEvent,
    newInputValue: string
  ) => {
    setInputValue(newInputValue);
    fetchLocations(newInputValue);
  };

  const handleLocationChange = (
    event: React.SyntheticEvent,
    value: string | Location | null
  ) => {
    if (value) {
      onLocationSelect?.(typeof value === "string" ? { title:
value } : value);
      setInputValue("");
    }
  };

  return (
    <Autocomplete
      value={null}
      freeSolo
      options={options}
      getOptionLabel={(location) =>
        typeof location === "string" ? location : location.title
      }
      onChange={handleLocationChange}
      inputValue={inputValue}
      onInputChange={handleInputChange}
      renderInput={(params) => (
        <TextField
          {...params}
          margin="dense"
          placeholder={label}
          onKeyDown={preventErase}
          InputProps={{
            ...params.InputProps,
            startAdornment: (
              <>
```

```
                <InputAdornment position="start">
                  <AddLocationAlt />
                </InputAdornment>
              </>
            ),
          }}
        />
      )}
      renderOption={(props, location) => (
        <LocationLineCard location={location} {...props} />
      )}
      sx={{
        ".MuiFormControl-root": {
          m: "0",
        },
      }}
    />
  );
};
```

LocationsList.tsx

```
import Typography from "@mui/material/Typography";
import {
  Box,
  Stepper,
  Step,
  StepLabel,
  StepContent,
  Stack,
  Icon,
  IconButton,
  Tooltip,
  MenuItem,
  ListItemIcon,
  Divider,
} from "@mui/material";
import { Delete, Edit, LocationOn, MoreVert, Paid } from
"@mui/icons-material";
import { useState } from "react";
import { PopupMenu } from
"../../../Shared/components/PopupMenu/PopupMenu";
import { DeleteModal } from
"../../../Shared/components/DeleteModal/DeleteModal";
import LocationService, {
  Location,
} from "../../../../services/LocationService";
import { BudgetModal } from "../BudgetModal/BudgetModal";
import { LocationCreationModal } from
"../LocationCreationModal/LocationCreationModal";

interface LocationsListProps {
  tripLocations: Location[];
```

```
  setTripLocations?: (cb: (prev: Location[]) => Location[]) =>
void;
}

export const LocationsList = ({
  tripLocations,
  setTripLocations,
}: LocationsListProps) => {
  const [anchorEl, setAnchorEl] = useState<HTMLElement |
null>(null);
  const [openModal, setOpenModal] = useState(false);
  const [isEditOpen, setIsEditOpen] = useState(false);
  const [isDeleteOpen, setIsDeleteOpen] = useState(false);
  const [selectedLocation, setSelectedLocation] = useState<
    Location | undefined
  >();
  const [categoryExpenses, setCategoryExpenses] = useState<any>({
    Food: 0,
    Transport: 0,
    Museums: 0,
    Other: 0,
  });

  const handleMenu = (location: any) => (e:
React.MouseEvent<HTMLElement>) => {
    setAnchorEl(e.currentTarget);
    setSelectedLocation(location);
  };

  const handleClose = () => {
    setAnchorEl(null);
    setSelectedLocation(undefined);
  };

  const handleOpenModal = () => {
    setOpenModal(true);
  };

  const handleCloseModal = () => {
    setOpenModal(false);
  };

  const handleChange = (
    e: React.ChangeEvent<HTMLInputElement>,
    category: string
  ) => {
    setCategoryExpenses((prevState: any) => ({
      ...prevState,
      [category]: Number(e.target.value),
    }));
  };

  const onEditCloseHandler = () => {
```

```
    setIsEditOpen(() => false);
  };

  const onEditOpenHandler = (location: Location) => (e:
React.MouseEvent) => {
    setIsEditOpen(true);
  };

  const onEditHandler = async (editLocation: Location) => {
    setTripLocations?.((prevLocations) =>
      prevLocations.map((location) =>
        location.id !== editLocation.id ? location : editLocation
      )
    );
  };

  const onDeleteCloseHandler = () => {
    setIsDeleteOpen(() => false);
  };

  const onDeleteOpenHandler = (location: any) => (e:
React.MouseEvent) => {
    e.stopPropagation();
    e.preventDefault();
    setIsDeleteOpen(true);
  };

  const onDeleteHandler = async () => {
    if (selectedLocation) {
      try {
        await LocationService.delete(selectedLocation.id);
        setTripLocations?.((prevLocations) =>
          prevLocations.filter(
            (location) => location.id !== selectedLocation.id
          )
        );
        setIsDeleteOpen(false);
      } catch (error) {
        console.error("Error deleting location:", error);
      }
    }
  };

  return (
    <Box
      sx={{
        width: "100%",
        mb: 2,
        maxHeight: "calc(5 * 65px)",
        overflowY: "auto",
      }}
    >
      <Stepper
```

```
          orientation="vertical"
          activeStep={tripLocations?.length - 1}
          sx={{
            ".MuiStepConnector-root": {
              ml: "calc(10px + 14px)",
            },
            ".MuiStepConnector-line": {
              minHeight: "30px",
              borderLeftStyle: "dashed",
              borderColor: "#757575",
            },
          }}
        >
          {tripLocations.map((location, index) => (
            <Step
              key={index}
              sx={{ backgroundColor: "#F5F5F5", borderRadius: 25 }}
            >
              <StepLabel
                StepIconComponent={() => (
                  <Icon
                    sx={{
                      backgroundColor: index === 0 ? "green" :
"#1976d2",
                      borderRadius: "50%",
                      display: "flex",
                      alignItems: "center",
                      justifyContent: "center",
                      p: 2,
                      fontSize: 49,
                    }}
                  >
                    {index === 0 ? (
                      <LocationOn sx={{ color: "#ffff" }} />
                    ) : (
                      <Typography variant="h6" sx={{ color: "#ffff"
}}>
                        {index}
                      </Typography>
                    )}
                  </Icon>
                )}
                sx={{ py: 0 }}
              >
                <Box
                  sx={{
                    display: "flex",
                    justifyContent: "space-between",
                    alignItems: "center",
                  }}
                >
                  <Stack sx={{ ml: 0.5 }}>
```

```
                    <Typography variant="subtitle1" sx={{ color:
"#232323" }}>
                      <b> {location.title} </b>
                    </Typography>
                  </Stack>
                  <Box>
                    <Tooltip title="Details" placement="top">
                      <IconButton onClick={handleMenu(location)}>
                        <MoreVert />
                      </IconButton>
                    </Tooltip>
                  </Box>
                  <PopupMenu anchorEl={anchorEl}
onClose={handleClose}>
                    <MenuItem onClick={handleOpenModal}>
                      <ListItemIcon>
                        <Paid fontSize="small" color="secondary" />
                      </ListItemIcon>
                      Add budget
                    </MenuItem>
                    <Divider />
                    <MenuItem onClick={onEditOpenHandler(location)}>
                      <ListItemIcon>
                        <Edit fontSize="small" color="primary" />
                      </ListItemIcon>
                      Edit
                    </MenuItem>
                    <MenuItem
onClick={onDeleteOpenHandler(location)}>
                      <ListItemIcon>
                        <Delete fontSize="small" color="error" />
                      </ListItemIcon>
                      Delete
                    </MenuItem>
                  </PopupMenu>
                </Box>
              </StepLabel>
              <StepContent></StepContent>
            </Step>
          ))}
        </Stepper>

        <BudgetModal
          open={openModal}
          onClose={handleCloseModal}
          locationId={selectedLocation?.id}
        />
        <LocationCreationModal
          open={isEditOpen}
          onClose={onEditCloseHandler}
          onSubmit={onEditHandler}
          modalTitle={`Edit location`}
          item={selectedLocation}
```

```
      />
      <DeleteModal
        open={isDeleteOpen}
        onClose={onDeleteCloseHandler}
        onDelete={onDeleteHandler}
        title={`Delete location?`}
        description={`Do you want to delete
"${selectedLocation?.title}"?`}
      />
    </Box>
  );
};
```

Map.tsx

```
import { useRef, useEffect, useMemo } from "react";

import mapboxgl from "mapbox-gl";
import * as mbgl from "mapbox-gl";
import "mapbox-gl/dist/mapbox-gl.css";

import MapboxGeocoder from "@mapbox/mapbox-gl-geocoder";
import "@mapbox/mapbox-gl-geocoder/dist/mapbox-gl-geocoder.css";

import MapInstructions from "./MapInstructions";
import { Coordinates } from "@mapbox/mapbox-sdk/lib/classes/mapi-
request";

import MapService, {
  Destination,
  Route,
} from "../../../../services/MapService";
import { Box } from "@mui/material";

const addPoint = (
  map: mbgl.Map,
  id: string,
  coordinates: Coordinates,
  color: string
) => {
  map.addLayer({
    id,
    type: "circle",
    source: {
      type: "geojson",
      data: {
        type: "FeatureCollection",
        features: [
          {
            type: "Feature",
            properties: {},
            geometry: {
              type: "Point",
```

```
              coordinates,
            },
          },
        ],
      },
    },
    paint: {
      "circle-radius": 10,
      "circle-color": color,
    },
  });
};

const addRoute = (
  map: mbgl.Map,
  id: string,
  route: Coordinates[],
  color: string
) => {
  map.addLayer({
    id,
    type: "line",
    source: {
      type: "geojson",
      data: {
        type: "Feature",
        properties: {},
        geometry: {
          type: "LineString",
          coordinates: route,
        },
      },
    },
    layout: {
      "line-join": "round",
      "line-cap": "round",
    },
    paint: {
      "line-color": color,
      "line-width": 5,
      "line-opacity": 0.75,
    },
  });
};

mapboxgl.accessToken = MapService.accessToken;

interface MapProps {
  options: Omit<mbgl.MapOptions, "container">;
  showSearch?: boolean;
  defaultColor?: string;
  onClick?: (coordinate: { lng: number; lat: number }) => void;
  route?: Route<GeoJSON.LineString> & { color?: string };
```

```
  waypoints?: (Destination & { color?: string })[];
}

export default function Map({
  options,
  showSearch,
  route,
  waypoints,
  onClick,
  defaultColor = "#3887be",
}: MapProps) {
  const mapContainer = useRef<HTMLDivElement | null>(null);
  const map = useRef<mapboxgl.Map | null>(null);

  useEffect(() => {
    if (mapContainer.current) {
      map.current = new mapboxgl.Map({
        container: mapContainer.current,
        ...options,
      });
    }
    return () => {
      map.current?.remove();
    };
  }, []);

  const seachControl = useMemo(
    () =>
      new MapboxGeocoder({
        accessToken: mapboxgl.accessToken,
        mapboxgl: mapboxgl as unknown as typeof mbgl,
      }),
    []
  );

  useEffect(() => {
    if (showSearch) {
      map.current?.addControl(seachControl);
    } else {
      if (map.current?.hasControl(seachControl)) {
        map.current.removeControl(seachControl);
      }
    }
  }, [seachControl, showSearch]);

  useEffect(() => {
    map.current?.on("click", (e) => {
      onClick?.(e.lngLat);
    });
  }, [onClick]);

  useEffect(() => {
    try {
```

```
      if (map.current?.getLayer("route")) {
        map.current.removeLayer("route");
        map.current.removeSource("route");
      }
      if (route) {
        const routeLine = route.geometry.coordinates as
Coordinates[];
        if (map.current)
          addRoute(
            map.current,
            "route",
            routeLine,
            route.color || defaultColor
          );
      }
    } catch (error) {
      console.error(error);
    }
  }, [route, defaultColor]);

  useEffect(() => {
    try {
      let index = 0;
      while (map.current?.getLayer(`point-${index}`)) {
        map.current.removeLayer(`point-${index}`);
        map.current.removeSource(`point-${index}`);
        ++index;
      }
      if (waypoints) {
        waypoints.forEach((waypoint, index) => {
          if (map.current)
            addPoint(
              map.current,
              `point-${index}`,
              waypoint.location as Coordinates,
              waypoint.color || defaultColor
            );
        });
      }
    } catch (error) {
      console.error(error);
    }
  }, [waypoints, defaultColor]);

  return (
    <Box>
      <Box
        ref={mapContainer}
        className="map-container"
        style={{ height: "500px", width: "100%" }}
      />
      {route && <MapInstructions routeInfo={route} />}
    </Box>
```

```
  );
}
```

MapInstructions.tsx

```
import { Route } from "@mapbox/mapbox-sdk/services/directions";

interface MapInstructionsProps {
  routeInfo: Route<GeoJSON.MultiLineString | GeoJSON.LineString>;
}

const MapInstructions = ({ routeInfo }: MapInstructionsProps) => {
  const steps = routeInfo.legs[0].steps;

  return (
    <div id="instructions" className="instructions">
      <p>
        <strong>
          Trip duration: {Math.floor(routeInfo.duration / 60)} min
🚴{" "}
        </strong>
      </p>
      <ol>
        {steps.map((step, index) => (
          <li key={`step-
${index}`}>{step.maneuver.instruction}</li>
        ))}
      </ol>
    </div>
  );
};

export default MapInstructions;
```

TripPage.tsx

```
import { styled } from "@mui/material/styles";
import ArrowForwardIosSharpIcon from "@mui/icons-
material/ArrowForwardIosSharp";
import MuiAccordion, { AccordionProps } from
"@mui/material/Accordion";
import MuiAccordionSummary, {
  AccordionSummaryProps,
} from "@mui/material/AccordionSummary";
import MuiAccordionDetails from "@mui/material/AccordionDetails";
import Typography from "@mui/material/Typography";
import Grid from "@mui/material/Grid2";
import { Box, Button, CircularProgress } from "@mui/material";
import { LocationsAutocomplete } from
"./components/LocationsAutocomplete/LocationsAutocomplete";
import Map from "./components/Map/Map";
```

```
import { useCallback, useContext, useEffect, useMemo, useState }
from "react";
import MapService, { Destination, Route } from
"../../services/MapService";
import { LocationsList } from
"./components/LocationsList/LocationsList";
import LocationService, { Location } from
"../../services/LocationService";
import { useParams } from "react-router-dom";
import { TripContext } from "../../contexts/TripContext";
import TripService, { Trip } from "../../services/TripService";
import { LocationCreationModal } from
"./components/LocationCreationModal/LocationCreationModal";
import clusterize from "../../algo/clusterize";
import sphericalDistance from "../../algo/sphericalDistance";
import chroma from "chroma-js";
import useLoading from "../../hooks/useLoading";
import dayjs from "dayjs";

const Accordion = styled((props: AccordionProps) => (
  <MuiAccordion disableGutters elevation={0} square {...props} />
))(({ theme }) => ({
  border: `1px solid ${theme.palette.divider}`,
  "&:not(:last-child)": {
    borderBottom: 0,
  },
  "&::before": {
    display: "none",
  },
}));

const AccordionSummary = styled((props: AccordionSummaryProps) =>
(
  <MuiAccordionSummary
    expandIcon={<ArrowForwardIosSharpIcon sx={{ fontSize: "0.9rem"
}} />}
    {...props}
  />
))(({ theme }) => ({
  backgroundColor: "rgba(0, 0, 0, .03)",
  flexDirection: "row-reverse",
  "& .MuiAccordionSummary-expandIconWrapper.Mui-expanded": {
    transform: "rotate(90deg)",
  },
  "& .MuiAccordionSummary-content": {
    marginLeft: theme.spacing(1),
  },
}));

const AccordionDetails = styled(MuiAccordionDetails)(({ theme })
=> ({
  padding: theme.spacing(2),
  borderTop: "1px solid rgba(0, 0, 0, .125)",
```

```
}));

interface Cluster {
  centroid: [number, number];
  elements: [number, number][];
  color?: string;
}

const dayMaxRadius = 3;

const addClusters = (
  locations: {
    longitude: number;
    latitude: number;
    cluster?: Cluster;
    color?: string;
  }[]
) => {
  const coordinates = locations.map((location) => [
    location.longitude,
    location.latitude,
  ]);
  const clusters = clusterize(
    coordinates,
    (point1: number[], point2: number[]) =>
      sphericalDistance(point1, point2) * 6371,
    () => dayMaxRadius
  );
  coordinates.forEach((coordinate, i) => {
    const clusterIndex = clusters.findIndex((cluster) =>
      cluster.elements.includes(coordinate)
    );
    if (clusterIndex !== -1) {
      locations[i].cluster = clusters[clusterIndex] as Cluster;
      locations[i].cluster.color = chroma
        .hsl(clusterIndex * 28.8, 1, 0.5)
        .hex();
      locations[i].color = locations[i].cluster.color;
    }
  });
  return clusters;
};

export const TripPage = () => {
  const [tripLocations, setTripLocations] = useState<
    (Location & { cluster?: Cluster; color?: string })[]
  >([]);
  const { tripId } = useParams();
  const [trip, setTrip] = useState<Trip | null>(null);
  const { activate } = useContext(TripContext);
  const [chosenLocation, setChosenLocation] =
    useState<Partial<Location> | null>(null);
  const getContent = useCallback(
```

```
    async (tripId: string) =>
      setTripLocations(
        (await LocationService.getAllLocationsByTrip(tripId)).map(
          ({ longitude, latitude, ...rest }) => ({
            longitude: +longitude,
            latitude: +latitude,
            ...rest,
          })
        )
      ),
    []
  );
  const { isLoading, execute } = useLoading(
    getContent as (...args: unknown[]) => Promise<void> | void
  );

  const [search, setSearch] = useState<boolean | undefined>();
  const [activeDay, setActiveDay] = useState<number |
undefined>();
  const [routes, setRoutes] = useState<
    (
      | {
          waypoints: (Destination & { color?: string })[];
          route: Route<GeoJSON.LineString> & { color?: string };
        }
      | undefined
    )[]
  >([]);

  const profile = "walking";

  const bounds = useMemo(() => {
    if (tripLocations.length === 0) return;
    const lngs = tripLocations.map((location) =>
location.longitude);
    const lats = tripLocations.map((location) =>
location.latitude);
    return [
      [Math.min(...lngs), Math.min(...lats)],
      [Math.max(...lngs), Math.max(...lats)],
    ] as [[number, number], [number, number]];
  }, [tripLocations]);

  const coordinates = useMemo(
    () =>
      tripLocations.map((location) => [
        +location.longitude,
        +location.latitude,
      ]) as [number, number][],
    [tripLocations]
  );

  const tripWaypoints = useMemo(
```

```
    () =>
      tripLocations.map(({ longitude, latitude, title, ...rest })
=> ({
        name: title,
        location: [+longitude, +latitude] as [number, number],
        ...rest,
      })),
    [tripLocations]
  );

  const clusters = useMemo(
    () => [
      ...new Set(
        tripLocations
          .map((location) => location.cluster)
          .filter((cluster) => !!cluster)
      ),
    ],
    [tripLocations]
  );

  const fetchRoutes = useCallback(async () => {
    const routes = (
      await Promise.allSettled(
        clusters.map((cluster) =>
          MapService.getRoute(profile, cluster.elements).then(
            (json) => ({
              route: { color: cluster.color, ...json.trips[0] },
              waypoints: json.waypoints
                .map((point) => ({ color: cluster.color, ...point
}))
                .sort((l, r) => l.waypoint_index -
r.waypoint_index),
            }),
            console.error
          )
        )
      )
    ).map((route) => (route.status === "fulfilled" ? route.value :
undefined));
    setRoutes(routes);
  }, [clusters]);

  const route = useMemo(
    () => (activeDay === undefined ? undefined :
routes[activeDay]?.route),
    [activeDay, routes]
  );
  const waypoints = useMemo(
    () => (activeDay === undefined ? undefined :
routes[activeDay]?.waypoints),
    [activeDay, routes]
  );
```

```
  const handleLocationSelect = (selectedLocation:
Partial<Location>) => {
    setChosenLocation({ trip_id: tripId, ...selectedLocation });
  };

  const handleLocationSubmit = (newLocation: Location) =>
    setTripLocations((prev) => prev.concat(newLocation));

  useEffect(() => {
    (async () => {
      const trip = tripId ? await TripService.getOne(tripId) :
null;
      activate(trip);
      setTrip(trip);
    })();
  }, [activate, tripId]);

  useEffect(() => {
    execute(tripId);
  }, [execute, tripId]);

  const options = useMemo(
    () => ({
      style: "mapbox://styles/mapbox/streets-v12",
      bounds,
    }),
    [bounds]
  );

  if (isLoading) {
    return (
      <Box
        sx={{
          display: "flex",
          justifyContent: "center",
          alignItems: "center",
          height: "100%",
        }}
      >
        <CircularProgress />
      </Box>
    );
  }

  return (
    <Grid container spacing={1}>
      <Grid size={4}>
        <Typography variant="h6">All Trip Locations to
Visit</Typography>
        <Box sx={{ mb: 2 }}>
          <Accordion>
            <AccordionSummary
```

```
              aria-controls="panelPlaces-content"
              id="panelPlaces-header"
            >
              <Typography>Locations</Typography>
            </AccordionSummary>
            <AccordionDetails>
              <LocationsList
                tripLocations={tripLocations}
                setTripLocations={setTripLocations}
              />
              <LocationsAutocomplete
                label={"Add place"}
                onLocationSelect={handleLocationSelect}
              />
              <LocationCreationModal
                modalTitle={"Add new location"}
                open={!!chosenLocation}
                onClose={() => setChosenLocation(null)}
                onSubmit={handleLocationSubmit}
                item={chosenLocation || {}}
              />
              <Button
                variant="contained"
                sx={{ borderRadius: 25, mt: 3, width: "100%" }}
                onClick={() =>
                  setTripLocations((prev) => {
                    addClusters(prev);
                    return prev.slice();
                  })
                }
              >
                {"Split by days"}
              </Button>
              <Button
                variant="contained"
                sx={{ borderRadius: 25, mt: 3, width: "100%" }}
                disabled={clusters.length === 0}
                onClick={fetchRoutes}
              >
                {"Create routes"}
              </Button>
            </AccordionDetails>
          </Accordion>
        </Box>
        {clusters.length !== 0 && (
          <>
            <Typography variant="h6">Trip Locations by
Day</Typography>
            {tripLocations
              .reduce(
                (acum, location) => {
                  const day = acum.find(
```

```
                    (item) => item[0]?.cluster ===
location.cluster
                  );

                  // eslint-disable-next-line @typescript-
eslint/no-unused-expressions
                  day ? day.push(location) :
acum.push([location]);
                  return acum;
                },
                [] as (Location & { cluster?: Cluster })[][]
              )
              .map((day, index) => (
                <Accordion key={index}>
                  <AccordionSummary
                    aria-controls={`panel${index}bh-content`}
                    id={`panel${index}bh-header`}
                  >
                    <Typography>
                      {dayjs(trip?.start_date)
                        .add(index, "day")
                        .format("DD/MM/YY")}
                    </Typography>
                  </AccordionSummary>
                  <AccordionDetails>
                    <Button onClick={() => setActiveDay(index)}>
                      {"Show route"}
                    </Button>
                    <LocationsList tripLocations={day} />
                  </AccordionDetails>
                </Accordion>
              ))}
            <Button onClick={() => setActiveDay(undefined)}>
              {"Show all location"}
            </Button>
          </>
        )}
      </Grid>
      <Grid size={8}>
        <Box sx={{ flex: 1, border: 0 }}>
          <Map
            route={route}
            waypoints={waypoints || tripWaypoints}
            options={options}
            showSearch={search}
          />
        </Box>
      </Grid>
    </Grid>
  );
};
```

TripCard.tsx

```
import Card from "@mui/material/Card";
import CardContent from "@mui/material/CardContent";
import CardMedia from "@mui/material/CardMedia";
import Typography from "@mui/material/Typography";
import CardActionArea from "@mui/material/CardActionArea";
import { Box, CardActions, IconButton, Tooltip } from
"@mui/material";
import { Link } from "react-router-dom";
import TripService, { Trip } from
"../../../../services/TripService";
import { Delete, Edit } from "@mui/icons-material";
import { useState } from "react";
import { DeleteModal } from
"../../../Shared/components/DeleteModal/DeleteModal";
import { TripCreationModal } from
"../TripCreationModal/TripCreationModal";

interface TripCardProps {
  trip: Trip;
}

export const TripCard = ({ trip }: TripCardProps) => {
  const imageUrl = trip?.image
    ? import.meta.env.VITE_MEDIA_URL + "/" + trip.image
    : "/defaultImg.svg";
  const [isEditOpen, setIsEditOpen] = useState(false);
  const [isDeleteOpen, setIsDeleteOpen] = useState(false);

  const onEditOpenHandler = (e) => {
    e.stopPropagation();
    e.preventDefault();
    setIsEditOpen(() => true);
  };

  const onEditCloseHandler = () => {
    setIsEditOpen(() => false);
  };

  const onDeleteOpenHandler = (e) => {
    e.stopPropagation();
    e.preventDefault();
    setIsDeleteOpen(() => true);
  };

  const onDeleteCloseHandler = () => {
    setIsDeleteOpen(() => false);
  };

  const onDeleteHandler = async () => {
    await TripService.delete(trip.id);
  };
```

```
  return (
    <>
      <Card
        sx={{
          maxWidth: 345,
          borderRadius: 5,
          height: "100%",
          display: "flex",
          flexDirection: "column",
          justifyContent: "space-between",
        }}
      >
        <CardActionArea
          component={Link}
          to={`/trips/${trip.id}`}
          sx={{ height: "100%" }}
        >
          <Box sx={{ height: "180px", m: 1 }}>
            <CardMedia
              component="img"
              image={imageUrl}
              alt={trip.title}
              sx={{
                objectFit: "cover",
                width: "100%",
                height: "100%",
                borderRadius: 5,
              }}
            />
          </Box>
          <CardContent>
            <Typography gutterBottom variant="h6" component="div">
              {trip.title}
            </Typography>
            <Typography
              gutterBottom
              variant="body2"
              sx={{ color: "text.secondary" }}
            >
              {trip.description}
            </Typography>
            <Typography variant="subtitle2">
              Date: {new
Date(trip.start_date).toLocaleDateString()} -{" "}
              {new Date(trip.end_date).toLocaleDateString()}
            </Typography>
          </CardContent>
        </CardActionArea>
        <CardActions
          sx={{
            justifyContent: "space-between",
            bottom: 0,
```

```
            width: `100%`,
          }}
        >
          <Tooltip title="Delete" placement="top">
            <IconButton onClick={onDeleteOpenHandler}>
              <Delete />
            </IconButton>
          </Tooltip>
          <Tooltip title="Edit" placement="top">
            <IconButton onClick={onEditOpenHandler}>
              <Edit />
            </IconButton>
          </Tooltip>
        </CardActions>
      </Card>
      <TripCreationModal
        modalTitle="Edit trip"
        open={isEditOpen}
        onClose={onEditCloseHandler}
        item={trip}
      />
      <DeleteModal
        open={isDeleteOpen}
        onClose={onDeleteCloseHandler}
        onDelete={onDeleteHandler}
        title={`Delete "${trip.title}"?`}
        description={`Do you want to delete this trip?`}
      />
    </>
  );
};
```

TripCreationModal.tsx

```
import * as yup from "yup";
import { TextField } from "@mui/material";
import { DateField } from
"../../../Shared/components/Fields/DateField/DateField";
import TripService, { Trip } from
"../../../../services/TripService";
import { Modal } from "../../../Shared/components/Modal/Modal";
import { FieldRendererProps, Form } from
"../../../Shared/components/Form/Form";
import { FileField } from
"../../../Shared/components/Fields/FileField/FileField";
import { Dayjs } from "dayjs";
import { FormikHelpers } from "formik";
import config from "../../../../config";

interface TripFormValues {
  title: string;
  description: string;
  image?: File | null;
```

```
  start_date: string | Date | Dayjs;
  end_date: string | Date | Dayjs;
}

const validationSchema = yup.object({
  title: yup
    .string()
    .required("Name is required")
    .max(255, "Name must be less than 255 characters"),
  description: yup
    .string()
    .required("Description is required")
    .max(255, "Description must be less than 255 characters"),
  start_date: yup.date().required("The start date is required"),
  end_date: yup
    .date()
    .required("The end date is required")
    .min(
      yup.ref("start_date"),
      "The end date must be later than the start date"
    ),
  image: yup
    .mixed<File>()
    .nullable()
    .test("fileType", "Only image files are allowed", (file) => {
      if (!file) return true;
      return file.type.startsWith("image/");
    })
    .test(
      "fileSize",
      `File size must be less than ${config.MAX_FILE_SIZE / (1 <<
20)}MB`,
      (file) => {
        if (!file) return true;
        return file.size <= config.MAX_FILE_SIZE;
      }
    ),
});

const fieldsRenderers = {
  title: ({key, ...props}: FieldRendererProps<string>) => (
    <TextField key={key} label="Title" {...props} />
  ),
  description: ({key, ...props}: FieldRendererProps<string>) => (
    <TextField key={key} label="Description" {...props} />
  ),
  image: ({key, ...props}: FieldRendererProps<File | null |
undefined>) => (
    <FileField key={key} {...props} />
  ),
  start_date: ({key, ...props}: FieldRendererProps<string | Date |
Dayjs>) => (
    <DateField key={key} label="Choose start date" {...props} />
```

```
  ),
  end_date: ({key, ...props}: FieldRendererProps<string | Date |
Dayjs>) => (
    <DateField key={key} label="Choose end date" {...props} />
  ),
};

interface TripCreationModalProps {
  modalTitle: string;
  open: boolean;
  onClose: (trip: Trip | null) => void;
  item?: Trip;
}

export const TripCreationModal = ({
  modalTitle,
  open,
  onClose,
  item,
}: TripCreationModalProps) => {
  const initialValues: TripFormValues = {
    title: item?.title || "",
    description: item?.description || "",
    image: null,
    start_date: item?.start_date || "",
    end_date: item?.end_date || "",
  };

  const onCreateTripHandler = async (
    values: TripFormValues,
    { resetForm }: FormikHelpers<TripFormValues>
  ) => {
    const formData = new FormData();
    for (let value in values) {
      formData.append(value, values[value]);
    }
    const trip = item?.id
      ? await TripService.update(item.id, formData)
      : await TripService.create(formData);

    resetForm();
    onClose(trip);
  };

  return (
    <Modal title={modalTitle} open={open} onClose={onClose}>
      <Form<TripFormValues>
        id="tripCreateUpdate"
        initialValues={initialValues}
        validationSchema={validationSchema}
        fieldsRenderers={fieldsRenderers}
        onSubmit={onCreateTripHandler}
      />
```

```
    </Modal>
  );
};
```

TripsPage.tsx

```
import Grid from "@mui/material/Grid2";
import { TripCard } from "./components/TripCard/TripCard";
import { useEffect, useState } from "react";
import TripService, { Trip } from "../../services/TripService";
import { Add } from "@mui/icons-material";
import { Fab } from "@mui/material";
import { TripCreationModal } from
"./components/TripCreationModal/TripCreationModal";

export const TripsPage = () => {
  const [trips, setTrips] = useState<Trip[]>([]);
  const [loading, setLoading] = useState<boolean>(true);
  const [isTripCreateModalOpen, setIsTripCreateModalOpen] =
useState<boolean>(false);

  useEffect(() => {
    TripService.getAll().then(
      (trips) => {
        setTrips(trips);
        setLoading(false);
      },
      (error) => {
        console.error(error);
      }
    );
  }, []);

  const onTripCreateModalOpen = () => {
    setIsTripCreateModalOpen(true);
  };

  const onTripCreateModalClose = (newTrip: Trip | null) => {
    setIsTripCreateModalOpen(false);
    if (newTrip) {
      setTrips((prev) => prev.concat(newTrip));
    }
  };

  return (
    <Grid container spacing={4} columns={16}>
      {!loading ? (
        trips.map((trip) => (
          <Grid size={4} key={trip.id}>
            <TripCard trip={trip} />
          </Grid>
        ))
      ) : (
```

```
        <></>
      )}
      <TripCreationModal
        modalTitle="Create new trip"
        open={isTripCreateModalOpen}
        onClose={onTripCreateModalClose}
      />
      <Fab
        color="primary"
        sx={{
          position: "fixed",
          bottom: 30,
          right: 30,
        }}
        onClick={onTripCreateModalOpen}
      >
        <Add />
      </Fab>
    </Grid>
  );
};
```

AuthService.ts

```
import $api from '../http/Api';

export interface LoginValues {
  email: string;
  password: string;
}

export interface RegistrationValues {
  fullName: string,
  email: string,
  password: string,
  avatar?: File | null,
};

export default class AuthService {
  static async login(values: LoginValues) {
    let response;
    try {
      response = await $api.post('auth/login', values);
    } catch (error) {
      throw new Error(
        error.response.status === 401
          ? 'Wrong email or password'
          : 'Oops, something is wrong'
      );
    }
    return response.data;
  }
```

```
  static async registration(values: RegistrationValues) {
    let response;
    const formData = new FormData();
    Object.entries(values).forEach(([key, value]) =>
formData.append(key, value));
    try {
      response = await $api.post('auth/registration', formData);
    } catch (error) {
      throw new Error(
        error.response?.status === 409
          ? 'An account is already registered with your email'
          : 'Oops, something is wrong'
      );
    }
    return response.data;
  }

  static async logout() {
    return $api.post('auth/logout');
  }
}
```

BudgetCategoryService.ts

```
import $api from "../http/Api";

export interface BudgetCategory {
  id?: number;
  title: string;
}

export default class BudgetCategoryService {
  static async getAllBudgetCategoryByLocation(locationId) {
    const response = await $api.get(`budget-
category/${locationId}/all`);
    return response.data;
  }

  static async update(
    id: number,
    category: Partial<BudgetCategory>
  ): Promise<BudgetCategory> {
    const response = await $api.put<BudgetCategory>(`budget-
category/${id}`, category);
    return response.data;
  }

  static async create(category: BudgetCategory):
Promise<BudgetCategory> {
    return $api.post("budget-category/", category);
  }

  static async delete(id: number): Promise<void> {
```

```
    await $api.delete(`budget-category/${id}`);
  }
}
```

HotelService.ts

```
import axios from "axios";

export interface Hotel {
  id: number;
  title: string;
  location: string;
  guests: string;
  amenities: string;
  pricePerNight: number;
  rating: number;
  reviews: number;
  image: string;
}

const $api = axios.create({
  withCredentials: true,
  baseURL: import.meta.env.VITE_HOTELS_API_URL + "/api/",
});

export default class HotelService {
  static async getAllHotels(): Promise<Hotel[]> {
    const response = await $api.get<Hotel[]>("hotel/");
    return response.data;
  }

  static async getHotel(id: number): Promise<Hotel> {
    const response = await $api.get<Hotel>(`hotel/${id}`);
    return response.data;
  }

  static async createHotel(hotel: Hotel): Promise<Hotel> {
    const response = await $api.post<Hotel>("hotel/", hotel);
    return response.data;
  }

  static async updateHotel(id: number, hotel: Hotel):
Promise<Hotel> {
    const response = await $api.put<Hotel>(`hotel/${id}`, hotel);
    return response.data;
  }

  static async deleteHotel(id: number): Promise<void> {
    await $api.delete(`hotel/${id}`);
  }
}
```

LocationService.ts

```
import $api from "../http/Api";

export interface Location {
  id?: number;
  trip_id?: string;
  title: string;
  latitude: number;
  longitude: number;
}

export default class LocationService {
  static async getAll(): Promise<Location[]> {
    const response = await $api.get<Location[]>("location/");
    return response.data;
  }

  static async getOne(id: number): Promise<Location> {
    const response = await $api.get<Location>(`location/${id}`);
    return response.data;
  }

  static async update(
    id: number,
    location: Partial<Location>
  ): Promise<Location> {
    const response = await $api.put<Location>(`location/${id}`,
location);
    return response.data;
  }

  static async create(location: Location): Promise<Location> {
    const result = await $api.post<Location>("location/",
location);
    return result.data;
  }

  static async delete(id: number): Promise<void> {
    await $api.delete(`location/${id}`);
  }

  static async geocode(query: string): Promise<any[]> {
    const response = await fetch(

`https://api.mapbox.com/geocoding/v5/mapbox.places/${encodeURIComp
onent(
        query
      )}.json?access_token=${import.meta.env.VITE_MAPBOX_TOKEN}`
    );
    const data = await response.json();
    return data.features || [];
  }
```

```
  static async createCategoryForLocation(locationId, category) {
    await $api.post(`location/${locationId}/category`, category);
  }

  static async getCategoriesForLocation(locationId: string) {
    const response = await
$api.get(`location/${locationId}/categories`);
    return response.data;
  }

  static async updateCategoryAmount(locationId: string) {
    await $api.post(`location/${locationId}`);
  }

  static async deleteCategoryFromLocation(locationId: number,
categoryId: number) {
    return await
$api.delete(`location/${locationId}/category/${categoryId}`);
  }

  static async getAllLocationsByTrip(tripId) {
    const response = await $api.get(`location/${tripId}/all`);
    return response.data;
  }
}
```

MapService.ts

```
import mapboxSdk from "@mapbox/mapbox-sdk";
import {
  Coordinates,
  MapboxProfile,
} from "@mapbox/mapbox-sdk/lib/classes/mapi-request";
import directionsSdk, { Route } from "@mapbox/mapbox-
sdk/services/directions";
import matrixSdk, { MatrixResponse } from "@mapbox/mapbox-
sdk/services/matrix";
import optimizationSdk from "@mapbox/mapbox-
sdk/services/optimization";
import { Destination } from "@mapbox/mapbox-sdk/services/matrix";

export type { Route };
export type { Destination };

export interface DirectionsResponse {
  routes: Route<GeoJSON.LineString>[];
  waypoints: Destination[];
}

export interface RouteResponse {
  code: string;
  waypoints: (Destination & {
```

```
    waypoint_index: number;
    trips_index: number;
  })[];
  trips: Route<GeoJSON.LineString>[];
}

class MapService {
  static readonly accessToken = import.meta.env.VITE_MAPBOX_TOKEN;

  static readonly mapboxClient = mapboxSdk({
    accessToken: this.accessToken,
  });

  static readonly directionsClient =
directionsSdk(this.mapboxClient);
  static readonly matrixClient = matrixSdk(this.mapboxClient);
  static readonly optimizationClient =
optimizationSdk(this.mapboxClient);

  static async getDirections(
    profile: MapboxProfile,
    coordinates: Coordinates[]
  ): Promise<DirectionsResponse> {
    return (
      await this.directionsClient
        .getDirections({
          profile,
          waypoints: coordinates.map((coord) => ({ coordinates:
coord })),
          steps: true,
          geometries: "geojson",
        })
        .send()
    ).body as unknown as DirectionsResponse;
  }

  static async getMatrix(
    profile: MapboxProfile,
    coordinates: Coordinates[]
  ): Promise<MatrixResponse> {
    return (
      await this.matrixClient
        .getMatrix({
          profile,
          points: coordinates.map((coord) => ({ coordinates: coord
})),
        })
        .send()
    ).body;
  }

  static async getRoute(
    profile: MapboxProfile,
```

```
    coordinates: Coordinates[]
  ): Promise<RouteResponse> {
    return (
      await this.optimizationClient
        .getOptimization({
          profile,
          waypoints: coordinates.map((coord) => ({ coordinates:
coord })),
          steps: true,
          geometries: "geojson",
          roundtrip: false,
          source: "first",
          destination: "last",
        })
        .send()
    ).body;
  }
}

export default MapService;
```

ReservationService.ts

```
import axios from "axios";

export interface Reservation {
  id: number;
  tripId: number;
  startDate: string;
  endDate: string;
  status: string;
  fullName: string;
  email: string;
  phone: string;
  country: string;
}

const $api = axios.create({
  withCredentials: true,
  baseURL: import.meta.env.VITE_HOTELS_API_URL + "/api/",
});

export default class ReservationService {
  static async getAllReservations(): Promise<Reservation[]> {
    const response = await
$api.get<Reservation[]>("reservation/");
    return response.data;
  }

  static async getReservation(id: number): Promise<Reservation> {
    const response = await
$api.get<Reservation>(`reservation/${id}`);
    return response.data;
```

```
  }

  static async getReservationsByUser(userId) {
    const response = await $api.get(`reservation/${userId}`);
    return response.data;
  }

  static async createReservation(
    reservation: Reservation
  ): Promise<Reservation> {
    const response = await $api.post<Reservation>("reservation/",
reservation);
    return response.data;
  }

  static async updateReservation(
    id: number,
    reservation: Reservation
  ): Promise<Reservation> {
    const response = await $api.put<Reservation>(
      `reservation/${id}`,
      reservation
    );
    return response.data;
  }

  static async deleteReservation(id: number): Promise<void> {
    await $api.delete(`reservation/${id}`);
  }
}
```

TripService.ts

```
import $api from "../http/Api";
import { Dayjs } from "dayjs";

export interface Trip {
  id: number;
  title: string;
  description?: string;
  image: string;
  start_date: string | Date | Dayjs;
  end_date: string | Date | Dayjs;
}

export interface TripItem {
  id: number;
  tripId: number;
  title: string;
}

export default class TripService {
  static async getAll(): Promise<Trip[]> {
```

```
    const response = await $api.get<Trip[]>("trip/");
    return response.data;
  }

  static async getOne(id: number): Promise<Trip> {
    const response = await $api.get<Trip>(`trip/${id}`);
    return response.data;
  }

  static async update(id: number, trip: Trip): Promise<Trip> {
    const response = await $api.put(`trip/${id}`, trip);
    return response.data;
  }

  static async delete(id: number): Promise<void> {
    await $api.delete(`trip/${id}`);
  }

  static async create(trip: Trip): Promise<Trip> {
    return $api.post("trip/", trip);
  }

  static async addTripItem(tripId: number, title: string):
Promise<TripItem> {
    const response = await $api.post<TripItem>("trip/item", {
tripId, title });
    return response.data;
  }

  static async updateTripItem(id: number, data:
Partial<TripItem>): Promise<TripItem> {
    const response = await $api.put<TripItem>(`trip/item/${id}`,
data);
    return response.data;
  }

  static async getTripItems(tripId: number): Promise<TripItem[]> {
    const response = await
$api.get<TripItem[]>(`trip/${tripId}/items`);
    return response.data;
  }

  static async deleteTripItem(id: number): Promise<void> {
    await $api.delete(`trip/item/${id}`);
  }
}
```

UserService.ts

```
import { User } from "../contexts/UserContext";
import $api from "../http/Api";

export default class UserService {
```

```
  static async getAllUsers(): Promise<User[]> {
    const response = await $api.get<User[]>("user/");
    return response.data;
  }

  static async getUser(id: number): Promise<User> {
    const response = await $api.get<User>(`user/${id}`);
    return response.data;
  }

  static async updateUser(id: number, user: FormData):
Promise<User> {
    const response = await $api.put<User>(`user/${id}`, user);
    return response.data;
  }
}
```

WishlistService.ts

```
import axios from "axios";

export interface WishlistItem {
  id: number;
  userId: number;
  hotelId: number;
  hotel?: {
    id: number;
    title: string;
    location: string;
    pricePerNight: number;
    image: string;
  };
}

const $api = axios.create({
  withCredentials: true,
  baseURL: import.meta.env.VITE_HOTELS_API_URL + "/api/",
});

export default class WishlistService {
  static async getWishlist(userId: number):
Promise<WishlistItem[]> {
    const response = await
$api.get<WishlistItem[]>(`wishlist/${userId}`);
    return response.data;
  }

  static async addToWishlist(userId: number, hotelId: number):
Promise<WishlistItem> {
    const response = await $api.post<WishlistItem>("wishlist/add",
{ userId, hotelId });
    return response.data;
  }
```

```
  static async removeFromWishlist(userId: number, hotelId:
number): Promise<{ message: string }> {
    const response = await $api.delete<{ message: string
}>("wishlist/remove", {
      data: { userId, hotelId },
    });
    return response.data;
  }
}
```

vite-env.d.ts

```
/// <reference types="vite/client" />
```

vite.config.ts

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vite.dev/config/
export default defineConfig({
  plugins: [react()],
})
```

babel.config.js

```
module.exports = {
  presets: [['@babel/preset-env', { targets: { node: 'current' }
}]],
};
```

FileController.js

```
export default class FileController {
  constructor() {
    this.getFile = this.getFile.bind(this);
  }

  async getFile(req, res) {
    try {
      return res.download('static/' + req.params.file);
    } catch (error) {
      res.status(500).json(error);
    }
  }
}
```

HotelController.js

```
import HotelService from "../services/HotelService";

export default class HotelController {
  constructor() {
    this.getAllHotels = this.getAllHotels.bind(this);
    this.getHotel = this.getHotel.bind(this);
    this.createHotel = this.createHotel.bind(this);
    this.updateHotel = this.updateHotel.bind(this);
    this.deleteHotel = this.deleteHotel.bind(this);
  }

  async getAllHotels(req, res, next) {
    try {
      const hotels = await HotelService.getAllHotels();
      return res.json(hotels);
    } catch (error) {
      next(error);
    }
  }

  async getHotel(req, res, next) {
    try {
      const hotel = await HotelService.getHotel(req.params.id);
      return res.json(hotel);
    } catch (error) {
      next(error);
    }
  }

  async createHotel(req, res) {
    try {
      const hotel = await HotelService.createHotel(req.body,
req.files?.image);
      res.json(hotel);
    } catch (error) {
      res.status(500).json(error);
    }
  }

  async updateHotel(req, res, next) {
    try {
      const updatedHotel = await HotelService.updateHotel(
        req.params.id,
        req.body,
        req.files?.image
      );
      return res.json(updatedHotel);
    } catch (error) {
      next(error);
    }
  }

  async deleteHotel(req, res, next) {
```

```
    try {
      const hotel = await HotelService.deleteHotel(req.params.id);
      return res.json(hotel);
    } catch (error) {
      next(error);
    }
  }
}
```

ReservationController.js

```
import ReservationService from "../services/ReservationService";

export default class ReservationController {
  constructor() {
    this.getAllReservations = this.getAllReservations.bind(this);
    this.getReservation = this.getReservation.bind(this);
    this.getReservationsByUser =
this.getReservationsByUser.bind(this);
    this.createReservation = this.createReservation.bind(this);
    this.updateReservation = this.updateReservation.bind(this);
    this.deleteReservation = this.deleteReservation.bind(this);
  }

  async getAllReservations(req, res, next) {
    try {
      const reservations = await
ReservationService.getAllReservations();
      return res.json(reservations);
    } catch (error) {
      next(error);
    }
  }

  async getReservation(req, res, next) {
    try {
      const reservation = await
ReservationService.getReservation(req.params.id);
      return res.json(reservation);
    } catch (error) {
      next(error);
    }
  }

  async getReservationsByUser(req, res, next) {
    try {
      const reservation = await
ReservationService.getReservationsByUser(req.params.userId);
      return res.json(reservation);
    } catch (error) {
      next(error);
    }
  }
```

```
  async createReservation(req, res, next) {
    try {
      const reservation = await
ReservationService.createReservation(req.body);
      res.json(reservation);
    } catch (error) {
      next(error);
    }
  }

  async updateReservation(req, res, next) {
    try {
      const updatedReservation = await
ReservationService.updateReservation(
        req.params.id,
        req.body
      );
      return res.json(updatedReservation);
    } catch (error) {
      next(error);
    }
  }

  async deleteReservation(req, res, next) {
    try {
      const reservation = await
ReservationService.deleteReservation(req.params.id);
      return res.json(reservation);
    } catch (error) {
      next(error);
    }
  }
}
```

WishlistController.js

```
import WishlistService from "../services/WishlistService";

export default class WishlistController {
  constructor() {
    this.addToWishlist = this.addToWishlist.bind(this);
    this.getWishlist = this.getWishlist.bind(this);
    this.removeFromWishlist = this.removeFromWishlist.bind(this);
  }

  async addToWishlist(req, res, next) {
    try {
      const { userId, hotelId } = req.body;
      const wishlistItem = await
WishlistService.addToWishlist(userId, hotelId);
      return res.json(wishlistItem);
    } catch (error) {
```

```
      next(error);
    }
  }

  async getWishlist(req, res, next) {
    try {
      const userId = req.params.userId;
      const wishlist = await WishlistService.getWishlist(userId);
      return res.json(wishlist);
    } catch (error) {
      next(error);
    }
  }

  async removeFromWishlist(req, res, next) {
    try {
      const { userId, hotelId } = req.body;
      const result = await
WishlistService.removeFromWishlist(userId, hotelId);
      return res.json(result);
    } catch (error) {
      next(error);
    }
  }
}
```

db.js

```
import { Sequelize } from 'sequelize';
import env from 'dotenv'
env.config();

const sequelize = new Sequelize(
  process.env.DB_NAME,
  process.env.DB_USER,
  process.env.DB_PASSWORD,
  {
    dialect: 'postgres',
    host: process.env.DB_HOST,
    port: process.env.DB_PORT,
  }
);

export default sequelize;
```

ApiError.js

```
export default class ApiError extends Error {
  constructor(status, message, errors = []) {
    super(message);
    this.status = status;
    this.errors = errors;
```

```
    Object.setPrototypeOf(this, ApiError.prototype);
  }

  static UnauthorizedError() {
    return new ApiError(401, "User not authorized");
  }

  static BadRequest(message, errors = []) {
    return new ApiError(400, message, errors);
  }
}
```

ErrorMiddleware.js

```
import ApiError from '../exceptions/ApiError';

export default function (err, req, res, next) {
  if (err instanceof ApiError) {
    return res.status(err.status).json({ message: err.message,
errors: err.errors });
  }
  return res.status(500).json({ message: err.message ||
'Unexpected error' });
}
```

Models.js

```
import sequelize from "../db";
import { DataTypes } from "sequelize";

export const Hotel = sequelize.define("hotel", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
  image: { type: DataTypes.STRING, allowNull: true },
  title: { type: DataTypes.STRING, allowNull: false },
  location: { type: DataTypes.STRING, allowNull: false },
  guests: { type: DataTypes.INTEGER, allowNull: false },
  description: { type: DataTypes.STRING, allowNull: true },
  pricePerNight: { type: DataTypes.FLOAT, allowNull: false },
  rating: { type: DataTypes.FLOAT, allowNull: true, defaultValue:
0 },
  reviews: { type: DataTypes.INTEGER, allowNull: true,
defaultValue: 0 },
});

export const Reservation = sequelize.define("reservation", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
  tripId: { type: DataTypes.INTEGER, allowNull: false },
  hotelId: { type: DataTypes.INTEGER, allowNull: false },
  userId: { type: DataTypes.INTEGER, allowNull: false },
```

```
  startDate: { type: DataTypes.DATEONLY, allowNull: false },
  endDate: { type: DataTypes.DATEONLY, allowNull: false },
  status: {
    type: DataTypes.STRING,
    allowNull: false,
    defaultValue: "pending",
  },
  fullName: { type: DataTypes.STRING, allowNull: false },
  email: { type: DataTypes.STRING, allowNull: false },
  phone: { type: DataTypes.STRING, allowNull: false },
  country: { type: DataTypes.STRING, allowNull: false },
});

export const Wishlist = sequelize.define("wishlist", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
  userId: { type: DataTypes.INTEGER, allowNull: false },
  hotelId: { type: DataTypes.INTEGER, allowNull: false },
});

Wishlist.hasMany(Hotel);
Hotel.belongsTo(Wishlist);

Hotel.hasMany(Reservation);
Reservation.belongsTo(Hotel);
```

fileRoutes.js

```
import { Router } from 'express';
import FileController from '../controllers/FileController';

const fileRoutes = new Router();
const fileController = new FileController();

fileRoutes.get('/:file', fileController.getFile);

export default fileRoutes;
```

hotelRoutes.js

```
import { Router } from "express";
import HotelController from "../controllers/HotelController";

const hotelRoutes = new Router();
const hotelController = new HotelController();

hotelRoutes.get("/:id", hotelController.getHotel);
hotelRoutes.post("/", hotelController.createHotel);
hotelRoutes.get("/", hotelController.getAllHotels);
hotelRoutes.put("/:id", hotelController.updateHotel);
hotelRoutes.delete("/:id", hotelController.deleteHotel);
```

```
export default hotelRoutes;
```

mainRoutes.js

```
import { Router } from 'express';
import fileRoutes from './fileRoutes';
import hotelRoutes from './hotelRoutes';
import wishlistRoutes from './wishlistRoutes';
import reservationRoutes from './reservationRoutes';

const mainRoutes = new Router();

mainRoutes.use('/hotel', hotelRoutes);
mainRoutes.use('/wishlist', wishlistRoutes);
mainRoutes.use('/reservation', reservationRoutes);
mainRoutes.use('/file', fileRoutes);

export default mainRoutes;
```

reservationRoutes.js

```
import { Router } from "express";
import ReservationController from
"../controllers/ReservationController";

const reservationRoutes = new Router();
const reservationController = new ReservationController();

reservationRoutes.get("/",
reservationController.getAllReservations);
reservationRoutes.get("/:userId",
reservationController.getReservationsByUser);
reservationRoutes.get("/:id",
reservationController.getReservation);
reservationRoutes.post("/",
reservationController.createReservation);
reservationRoutes.put("/:id",
reservationController.updateReservation);
reservationRoutes.delete("/:id",
reservationController.deleteReservation);

export default reservationRoutes;
```

wishlistRoutes.js

```
import { Router } from "express";
import WishlistController from
"../controllers/WishlistController";

const wishlistRoutes = new Router();
const wishlistController = new WishlistController();
```

```
wishlistRoutes.post("/add", wishlistController.addToWishlist);
wishlistRoutes.get("/:userId", wishlistController.getWishlist);
wishlistRoutes.delete("/remove",
wishlistController.removeFromWishlist);

export default wishlistRoutes;
```

server.js

```
import express from 'express';
import fileUpload from 'express-fileupload';
import env from 'dotenv'
import cors from 'cors';
import * as path from 'path';
import sequelize from './db';
import mainRoutes from './routes/mainRoutes';
import cookieParser from 'cookie-parser';
import ErrorMiddleware from './middlewares/ErrorMiddleware';

env.config();
const app = express();

app.use(express.json());
app.use(cookieParser());
app.use(
  cors({
    origin: process.env.CLIENT_URL,
    credentials: true,
  }),
);
app.use(express.static(path.resolve('static')));
app.use(fileUpload({}));
app.use('/api', mainRoutes);
app.use(ErrorMiddleware);

const start = async () => {
  try {
    await sequelize.authenticate();
    await sequelize.sync();
    app.listen(process.env.PORT, () => {
      console.log(`App listening on port ${process.env.PORT}`);
    });
  } catch (error) {
    console.error(error);
  }
};

start();
```

FileService.js

```
import { File } from '../models/Models';
import * as uuid from 'uuid';
import * as path from 'path';

function checkID(id) {
  if (!id) throw new Error('No id specified');
}

class FileService {
  async getAllFiles() {
    return await File.findAll();
  }

  async getFile(id) {
    checkID(id);
    return await File.findOne({
      where: { id: id },
    });
  }

  async createFile(file) {
    return await File.create(file);
  }

  async updateFile(file) {
    checkID(file.id);
    return await File.update(file, {
      where: {
        id: file.id,
      },
    });
  }

  async deleteFile(id) {
    checkID(id);
    return await File.destroy({
      where: {
        id: id,
      },
    });
  }

  saveFile(file) {
    try {
      const fileName = uuid.v4() + path.extname(file.name);
      file.mv(path.resolve('static', fileName));
      return fileName;
    } catch (error) {
      console.error(error);
    }
  }
}
```

```
export default new FileService();
```

HotelService.js

```
import { Hotel } from "../models/Models";
import FileService from "./FileService";
import ApiError from "../exceptions/ApiError";

function checkID(id) {
  if (!id) throw ApiError.BadRequest("No id specified");
}

class HotelService {
  async getAllHotels() {
    return await Hotel.findAll();
  }


  async getHotel(id) {
    checkID(id);
    return await Hotel.findOne({
      where: { id: id },
    });
  }

  async createHotel(hotelData, image) {
    const fileName = image ? FileService.saveFile(image) : null;
    return await Hotel.create({ ...hotelData, image: fileName });
  }

  async updateHotel(id, hotelData, image) {
    checkID(id);
    delete hotelData.id;

    const updateData = { ...hotelData };
    if (image) updateData.image = FileService.saveFile(image);

    await Hotel.update(updateData, {
      where: { id: id },
    });

    return await Hotel.findOne({ where: { id: id } });
  }

  async deleteHotel(id) {
    checkID(id);
    return await Hotel.destroy({
      where: { id: id },
    });
  }
}

export default new HotelService();
```

ReservationService.js

```
import { Reservation } from "../models/Models";
import ApiError from "../exceptions/ApiError";
import { QueryTypes } from 'sequelize';
import sequelize from '../db';

function checkID(id) {
  if (!id) throw ApiError.BadRequest("No id specified");
}

class ReservationService {
  async getAllReservations() {
    return await Reservation.findAll();
  }

  async getReservation(id) {
    checkID(id);
    return await Reservation.findOne({
      where: { id },
    });
  }

  async getReservationsByUser(userId) {
    checkID(userId);
    return await sequelize.query(
      `SELECT id, "tripId", "startDate", "endDate", status,
"fullName", email, phone, country, "createdAt", "updatedAt",
"hotelId", "userId"
FROM public.reservations
WHERE "userId" = ${userId};`,
      { type: QueryTypes.SELECT }
    );
  }

  async createReservation(reservationData) {
    return await Reservation.create(reservationData);
  }

  async updateReservation(id, reservationData) {
    checkID(id);
    delete reservationData.id;

    await Reservation.update(reservationData, {
      where: { id },
    });

    return await Reservation.findOne({ where: { id } });
  }

  async deleteReservation(id) {
    checkID(id);
```

```
    return await Reservation.destroy({
      where: { id },
    });
  }
}

export default new ReservationService();
```

WishlistService.js

```
import { Wishlist, Hotel } from "../models/Models";
import ApiError from "../exceptions/ApiError";

class WishlistService {
  async addToWishlist(userId, hotelId) {
    if (!userId || !hotelId) {
      throw ApiError.BadRequest("User ID and Hotel ID are
required");
    }

    const exists = await Wishlist.findOne({ where: { userId,
hotelId } });
    if (exists) {
      throw ApiError.BadRequest("This hotel is already in your
wishlist");
    }

    return await Wishlist.create({ userId, hotelId });
  }

  async getWishlist(userId) {
    if (!userId) {
      throw ApiError.BadRequest("User ID is required");
    }

    return await Wishlist.findAll({
      where: { userId },
      include: [
        {
          model: Hotel,
          attributes: ["id", "title", "location", "pricePerNight",
"image"],
        },
      ],
    });
  }

  async removeFromWishlist(userId, hotelId) {
    if (!userId || !hotelId) {
      throw ApiError.BadRequest("User ID and Hotel ID are
required");
    }
```

```
    const removed = await Wishlist.destroy({ where: { userId,
hotelId } });
    if (!removed) {
      throw ApiError.BadRequest("Hotel not found in your
wishlist");
    }

    return { message: "Hotel removed from wishlist" };
  }
}

export default new WishlistService();
```

babel.config.js

```
module.exports = {
  presets: [['@babel/preset-env', { targets: { node: 'current' }
}]],
};
```

jwtConsts.js

```
export const JWT_ACCESS_SECRET_KEY = 'jwt-access-secret-key';
export const JWT_REFRESH_SECRET_KEY = 'jwt-refresh-secret-key';
```

AuthController.js

```
import { validationResult } from 'express-validator';
import ApiError from '../exceptions/ApiError';
import AuthService from '../services/AuthService';
import env from 'dotenv'
env.config();

export default class AuthController {
  constructor() {
    this.registration = this.registration.bind(this);
    this.activate = this.activate.bind(this);
    this.login = this.login.bind(this);
    this.logout = this.logout.bind(this);
    this.refresh = this.refresh.bind(this);
  }

  async registration(req, res, next) {
    try {
      const errors = validationResult(req);
      if (!errors.isEmpty()) {
        return next(ApiError.BadRequest('Validation error',
errors.array()));
      }
      const userData = await AuthService.registration(req.body,
req.files?.avatar);
```

```
    this.__addRefreshCookies(res, userData.refreshToken);
    return res.json(userData);
  } catch (error) {
    next(error);
  }
}

async activate(req, res, next) {
  try {
    const activationLink = req.params.link;
    await AuthService.activate(activationLink);
    return res.redirect(process.env.CLIENT_URL);
  } catch (error) {
    console.error(error);
    next(error);
  }
}

async login(req, res, next) {
  try {
    const { email, password } = req.body;
    const userData = await AuthService.login(email, password);
    this.__addRefreshCookies(res, userData.refreshToken);
    return res.json(userData);
  } catch (error) {
    next(error);
  }
}

async logout(req, res, next) {
  try {
    const { refreshToken } = req.cookies;
    const token = await AuthService.logout(refreshToken);
    res.clearCookie('refreshToken');
    return res.json(token);
  } catch (error) {
    next(error);
  }
}

async refresh(req, res, next) {
  try {
    const { refreshToken } = req.cookies;
    const userData = await AuthService.refresh(refreshToken);
    this.__addRefreshCookies(res, userData.refreshToken);
    return res.json(userData);
  } catch (error) {
    next(error);
  }
}

__addRefreshCookies(res, refreshToken) {
  res.cookie('refreshToken', refreshToken, {
```

```
      maxAge: 30 * 24 * 60 * 60 * 1000,
      httpOnly: true,
    });
  }
}
```

BudgetCategoryController.js

```
import BudgetCategoryService from
'../services/BudgetCategoryService';

export default class LocationController {
  constructor() {
    this.getAllBudgetCategoriesByLocation =
this.getAllBudgetCategoriesByLocation.bind(this);
    this.createBudgetCategory =
this.createBudgetCategory.bind(this);
    this.updateBudgetCategory =
this.updateBudgetCategory.bind(this);
    this.deleteBudgetCategory =
this.deleteBudgetCategory.bind(this);
  }

  async getAllBudgetCategoriesByLocation(req, res) {
    try {
      const categories = await
BudgetCategoryService.getAllBudgetCategoriesByLocation(req.params.
locationId);
      return res.json(categories);
    } catch (error) {
      res.status(500).json(error);
    }
  }

  async createBudgetCategory(req, res) {
    try {
      const category = await
BudgetCategoryService.createBudgetCategory(req.body);
      res.json(category);
    } catch (error) {
      res.status(500).json(error);
    }
  }

  async updateBudgetCategory(req, res, next) {
    try {
      const category = await
BudgetCategoryService.updateBudgetCategory(req.params.id,
req.body);
      return res.json(category);
    } catch (error) {
      next(error);
    }
```

```
  }

  async deleteBudgetCategory(req, res, next) {
    try {
      const category = await
BudgetCategoryService.deleteBudgetCategory(req.params.id);
      return res.json(category);
    } catch (error) {
      next(error);
    }
  }
}
```

FileController.js

```
export default class FileController {
  constructor() {
    this.getFile = this.getFile.bind(this);
  }

  async getFile(req, res) {
    try {
      return res.download('static/' + req.params.file);
    } catch (error) {
      res.status(500).json(error);
    }
  }
}
```

LocationController.js

```
import LocationService from '../services/LocationService';

export default class LocationController {
  constructor() {
    this.getAllLocations = this.getAllLocations.bind(this);
    this.getLocation = this.getLocation.bind(this);
    this.updateLocation = this.updateLocation.bind(this);
    this.deleteLocation = this.deleteLocation.bind(this);
    this.getCategoriesForLocation =
this.getCategoriesForLocation.bind(this);
    this.createCategoryForLocation =
this.createCategoryForLocation.bind(this);
    this.updateCategoryAmount =
this.updateCategoryAmount.bind(this);
    this.deleteCategoryFromLocation =
this.deleteCategoryFromLocation.bind(this);
  }

  async getAllLocations(req, res, next) {
    try {
      const locations = await LocationService.getAllLocations();
```

```
      return res.json(locations);
    } catch (error) {
      next(error);
    }
  }

  async getAllLocationsByTrip(req, res, next) {
    try {
      const locations = await
LocationService.getAllLocationsByTrip(req.params.tripId);
      return res.json(locations);
    } catch (error) {
      next(error);
    }
  }

  async getLocation(req, res, next) {
    try {
      const location = await
LocationService.getLocation(req.params.id);
      return res.json(location);
    } catch (error) {
      next(error);
    }
  }

  async createLocation(req, res, next) {
    try {
      const location = await
LocationService.createLocation(req.body);

      return res.json(location);
    } catch (error) {
      next(error);
    }
  }

  async getCategoriesForLocation(req, res, next) {
    try {
      const categories = await
LocationService.getCategoriesForLocation(req.params.locationId);
      return res.json(categories);
    } catch (error) {
      next(error);
    }
  }

  async createCategoryForLocation(req, res, next) {
    try {
      const category = await
LocationService.createCategoryForLocation(req.params.locationId,
req.body);
      return res.json(category);
```

```
    } catch (error) {
      next(error);
    }
  }

  async updateCategoryAmount(req, res, next) {
    try {
      const { locationId } = req.params;
      const { categoryId, newAmount } = req.body;

      if (!categoryId || newAmount === undefined) {
        return res.status(400).json({ error: 'categoryId и
newAmount обязательны' });
      }

      const updatedCategory = await
LocationService.updateCategoryAmount(locationId, categoryId,
newAmount);

      return res.json(updatedCategory);
    } catch (error) {
      next(error);
    }
  }

  async deleteCategoryFromLocation(req, res, next) {
    try {
      const { locationId, categoryId } = req.params;

      const category = await
LocationService.deleteCategoryFromLocation(locationId,
categoryId);

      return res.json(category);
    } catch (error) {
      next(error);
    }
  }

  async updateLocation(req, res, next) {
    try {
      const updatedLocation = await
LocationService.updateLocation(req.params.id, req.body,
req.files?.avatar);
      return res.json(updatedLocation);
    } catch (error) {
      next(error);
    }
  }

  async deleteLocation(req, res, next) {
    try {
```

```
      const location = await
LocationService.deleteLocation(req.params.id);
      return res.json(location);
    } catch (error) {
      next(error);
    }
  }
}
```

TripController.js

```
import TripService from "../services/TripService";

export default class TripController {
  constructor() {
    this.getAllTrips = this.getAllTrips.bind(this);
    this.getTrip = this.getTrip.bind(this);
    this.createTrip = this.createTrip.bind(this);
    this.updateTrip = this.updateTrip.bind(this);
    this.deleteTrip = this.deleteTrip.bind(this);
    this.addTripItem = this.addTripItem.bind(this);
    this.getTripItems = this.getTripItems.bind(this);
    this.deleteTripItem = this.deleteTripItem.bind(this);
    this.updateTripItem = this.updateTripItem.bind(this);
  }

  async getAllTrips(req, res, next) {
    try {
      const trips = await TripService.getAllTrips();
      return res.json(trips);
    } catch (error) {
      next(error);
    }
  }

  async getTrip(req, res, next) {
    try {
      const trip = await TripService.getTrip(req.params.id);
      return res.json(trip);
    } catch (error) {
      next(error);
    }
  }

  async createTrip(req, res) {
    try {
      const trip = await TripService.createTrip(req.body,
req.files?.image);
      res.json(trip);
    } catch (error) {
      console.error(error);

      res.status(500).json(error);
```

```
    }
  }

  async updateTrip(req, res, next) {
    try {
      const updatedTrip = await TripService.updateTrip(
        req.params.id,
        req.body,
        req.files?.avatar
      );
      return res.json(updatedTrip);
    } catch (error) {
      next(error);
    }
  }

  async deleteTrip(req, res, next) {
    try {
      const trip = await TripService.deleteTrip(req.params.id);
      return res.json(trip);
    } catch (error) {
      next(error);
    }
  }

  async addTripItem(req, res) {
    try {
      const { tripId, title } = req.body;
      const tripItem = await TripService.addTripItem(tripId,
title);
      return res.json(tripItem);
    } catch (error) {
      res.status(500).json(error);
    }
  }

  async updateTripItem(req, res, next) {
    try {
      const { id } = req.params;
      const updatedTripItem = await TripService.updateTripItem(id,
req.body);
      return res.json(updatedTripItem);
    } catch (error) {
      next(error);
    }
  }

  async getTripItems(req, res) {
    try {
      const tripItems = await
TripService.getTripItems(req.params.tripId);
      return res.json(tripItems);
    } catch (error) {
```

```
      res.status(500).json(error);
    }
  }

  async deleteTripItem(req, res) {
    try {
      const tripItem = await
TripService.deleteTripItem(req.params.id);
      return res.json(tripItem);
    } catch (error) {
      res.status(500).json(error);
    }
  }
}
```

UserController.js

```
import UserService from '../services/UserService';

export default class UserController {
  constructor() {
    this.getAllUsers = this.getAllUsers.bind(this);
    this.getUser = this.getUser.bind(this);
    this.updateUser = this.updateUser.bind(this);
    this.deleteUser = this.deleteUser.bind(this);
  }

  async getAllUsers(req, res, next) {
    try {
      const users = await UserService.getAllUsers();
      return res.json(users);
    } catch (error) {
      next(error);
    }
  }

  async getUser(req, res, next) {
    try {
      const user = await UserService.getUser(req.params.id);
      return res.json(user);
    } catch (error) {
      next(error);
    }
  }

  async updateUser(req, res, next) {
    try {
      const updatedUser = await
UserService.updateUser(req.params.id, req.body,
req.files?.avatar);
      return res.json(updatedUser);
    } catch (error) {
      next(error);
```

```
    }
  }

  async deleteUser(req, res, next) {
    try {
      const user = await UserService.deleteUser(req.params.id);
      return res.json(user);
    } catch (error) {
      next(error);
    }
  }
}
```

db.js

```
import { Sequelize } from "sequelize";
import env from 'dotenv'
env.config();

const sequelize = new Sequelize(
  process.env.DB_NAME,
  process.env.DB_USER,
  process.env.DB_PASSWORD,
  {
    dialect: "postgres",
    host: process.env.DB_HOST,
    port: process.env.DB_PORT,
  }
);

export default sequelize;
```

ApiError.js

```
export default class ApiError extends Error {
  constructor(status, message, errors = []) {
    super(message);
    this.status = status;
    this.errors = errors;

    Object.setPrototypeOf(this, ApiError.prototype);
  }

  static UnauthorizedError() {
    return new ApiError(401, "User not authorized");
  }

  static BadRequest(message, errors = []) {
    return new ApiError(400, message, errors);
  }
}
```

AuthMiddleware.js

```
import ApiError from '../exceptions/ApiError';
import TokenService from '../services/TokenService';

export default function (req, res, next) {
  try {
    const authoHeader = req.headers.authorization;
    if (!authoHeader) {
      return next(ApiError.UnauthorizedError());
    }
    const accessToken = authoHeader.split(' ')[1];
    if (!accessToken) {
      return next(ApiError.UnauthorizedError());
    }
    const userData =
TokenService.validateAccessToken(accessToken);

    if (!userData) {
      return next(ApiError.UnauthorizedError());
    }
    req.user = userData;
    next();
  } catch (error) {
    return next(ApiError.UnauthorizedError());
  }
}
```

ErrorMiddleware.js

```
import ApiError from '../exceptions/ApiError';

export default function (err, req, res, next) {
  if (err instanceof ApiError) {
    return res.status(err.status).json({ message: err.message,
errors: err.errors });
  }
  return res.status(500).json({ message: err.message ||
'Unexpected error' });
}
```

Models.js

```
import sequelize from "../db";
import { DataTypes } from "sequelize";

export const User = sequelize.define("user", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
  email: { type: DataTypes.STRING, unique: true, allowNull: false
},
  password: { type: DataTypes.STRING, allowNull: false },
```

```
  isActivated: {
    type: DataTypes.BOOLEAN,
    defaultValue: false,
  },
  activationLink: { type: DataTypes.STRING },
  fullName: { type: DataTypes.STRING, allowNull: false },
  avatar: { type: DataTypes.STRING, allowNull: true },
});

export const Token = sequelize.define("token", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
  refreshToken: { type: DataTypes.STRING, allowNull: false },
});

export const Trip = sequelize.define("trip", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
  start_date: { type: DataTypes.DATEONLY, allowNull: false },
  end_date: { type: DataTypes.DATEONLY, allowNull: false },
  title: { type: DataTypes.STRING, allowNull: false },
  image: { type: DataTypes.STRING, allowNull: true },
  description: { type: DataTypes.STRING, allowNull: true },
});

export const Location = sequelize.define("location", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
  title: { type: DataTypes.STRING, allowNull: false },
  latitude: { type: DataTypes.DECIMAL(10, 8), allowNull: false },
  longitude: { type: DataTypes.DECIMAL(11, 8), allowNull: false },
});

export const Route = sequelize.define("route", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
});

export const BudgetCategory = sequelize.define("budget_category",
{
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
  title: { type: DataTypes.STRING, allowNull: false },
});

export const PermissionType = sequelize.define("permission_type",
{
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
  title: { type: DataTypes.STRING, allowNull: false },
});

export const RouteLocation = sequelize.define("route_location", {
```

```
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
  number_by_order: { type: DataTypes.STRING, allowNull: true },
});

export const Permission = sequelize.define("permission", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
});

export const TripItem = sequelize.define("trip_item", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
  title: { type: DataTypes.STRING, allowNull: false },
});

export const TripDay = sequelize.define("trip_day", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
  number_by_order: { type: DataTypes.STRING, allowNull: false },
  description: { type: DataTypes.STRING, allowNull: true },
});

export const Reservation = sequelize.define("reservation", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
});

export const BudgetInLocation =
sequelize.define("budget_in_location", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
  amount: { type: DataTypes.FLOAT, allowNull: false },
});

export const LocationInRoute =
sequelize.define("location_in_route", {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
});

export const LocationInTrip = sequelize.define("location_in_trip",
{
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true },
});

User.hasMany(Permission);
Permission.belongsTo(User);

Trip.hasMany(TripItem);
TripItem.belongsTo(Trip);
```

```
Trip.hasMany(Permission);
Permission.belongsTo(Trip);

Trip.hasMany(TripDay);
TripDay.belongsTo(Trip);

Trip.hasMany(Reservation);
Reservation.belongsTo(Trip);

Permission.hasMany(PermissionType);
PermissionType.belongsTo(Permission);

BudgetCategory.belongsToMany(Location, { through: BudgetInLocation
});
Location.belongsToMany(BudgetCategory, { through: BudgetInLocation
});

Route.belongsToMany(Location, { through: LocationInRoute });
Location.belongsToMany(Route, { through: LocationInRoute });

Trip.belongsToMany(Location, { through: LocationInTrip });
Location.belongsToMany(Trip, { through: LocationInTrip });

TripDay.hasOne(Route);
Route.belongsTo(TripDay, {
  foreignKey: {
    allowNull: true,
  },
});

User.hasMany(Token);
Token.belongsTo(User, {
  foreignKey: {
    allowNull: false,
  },
});
```

authRoutes.js

```
import { Router } from 'express';
import { body } from 'express-validator';
import AuthController from '../controllers/AuthController';

const authRoutes = new Router();
const authController = new AuthController();

authRoutes.post(
  '/registration',
  body('email').isEmail(),
  body('password').isLength({ min: 5, max: 15 }),
  authController.registration,
);
authRoutes.post('/login', authController.login);
```

```
authRoutes.post('/logout', authController.logout);
authRoutes.get('/refresh', authController.refresh);
authRoutes.get('/activate/:link', authController.activate);

export default authRoutes;
```

budgetCategoryRoutes.js

```
import { Router } from 'express';
import BudgetCategoryController from
'../controllers/BudgetCategoryController';

const budgetCategoryRoutes = new Router();
const budgetCategoryController = new BudgetCategoryController();

budgetCategoryRoutes.get('/:tripId/all',
budgetCategoryController.getAllBudgetCategoriesByLocation);
budgetCategoryRoutes.post('/',
budgetCategoryController.createBudgetCategory);
budgetCategoryRoutes.put('/:id',
budgetCategoryController.updateBudgetCategory);
budgetCategoryRoutes.delete('/:id',
budgetCategoryController.deleteBudgetCategory);

export default budgetCategoryRoutes;
```

fileRoutes.js

```
import { Router } from 'express';
import FileController from '../controllers/FileController';

const fileRoutes = new Router();
const fileController = new FileController();

fileRoutes.get('/:file', fileController.getFile);

export default fileRoutes;
```

locationRoutes.js

```
import { Router } from 'express';
import LocationController from
'../controllers/LocationController';

const locationRoutes = new Router();
const locationController = new LocationController();

locationRoutes.get('/:id', locationController.getLocation);
locationRoutes.get('/:locationId/categories',
locationController.getCategoriesForLocation);
```

```
locationRoutes.post('/:locationId',
locationController.updateCategoryAmount);
locationRoutes.get('/:tripId/all',
locationController.getAllLocationsByTrip);
locationRoutes.post('/', locationController.createLocation);
locationRoutes.post('/:locationId/category',
locationController.createCategoryForLocation);
locationRoutes.get('/', locationController.getAllLocations);
locationRoutes.put('/:id', locationController.updateLocation);
locationRoutes.delete('/:id', locationController.deleteLocation);
locationRoutes.delete('/:locationId/category/:categoryId',
locationController.deleteCategoryFromLocation);

export default locationRoutes;
```

mainRoutes.js

```
import { Router } from 'express';
import fileRoutes from './fileRoutes';
import authRoutes from './authRoutes';
import AuthMiddleware from '../middlewares/AuthMiddleware';
import userRoutes from './userRoutes';
import tripRoutes from './tripRoutes';
import locationRoutes from './locationRoutes';
import budgetCategoryRoutes from './budgetCategoryRoutes';

const mainRoutes = new Router();

mainRoutes.use('/auth', authRoutes);
mainRoutes.use('/user', AuthMiddleware, userRoutes);
mainRoutes.use('/trip', AuthMiddleware, tripRoutes);
mainRoutes.use('/location', AuthMiddleware, locationRoutes);
mainRoutes.use('/budget-category', AuthMiddleware,
budgetCategoryRoutes);
mainRoutes.use('/file', AuthMiddleware, fileRoutes);

export default mainRoutes;
```

tripRoutes.js

```
import { Router } from 'express';
import TripController from '../controllers/TripController';

const tripRoutes = new Router();
const tripController = new TripController();

tripRoutes.get('/:id', tripController.getTrip);
tripRoutes.post('/', tripController.createTrip);
tripRoutes.get('/', tripController.getAllTrips);
tripRoutes.put('/:id', tripController.updateTrip);
tripRoutes.delete('/:id', tripController.deleteTrip);
tripRoutes.post("/item", tripController.addTripItem);
```

```
tripRoutes.put("/item/:id", tripController.updateTripItem);
tripRoutes.get("/:tripId/items", tripController.getTripItems);
tripRoutes.delete("/item/:id", tripController.deleteTripItem);


export default tripRoutes;
```

userRoutes.js

```
import { Router } from 'express';
import UserController from '../controllers/UserController';

const userRoutes = new Router();
const userController = new UserController();

userRoutes.get('/', userController.getAllUsers);
userRoutes.get('/:id', userController.getUser);
userRoutes.put('/:id', userController.updateUser);
userRoutes.delete('/:id', userController.deleteUser);

export default userRoutes;
```

server.js

```
import express from 'express';
import fileUpload from 'express-fileupload';
import env from 'dotenv'
import cors from 'cors';
import * as path from 'path';
import sequelize from './db';
import mainRoutes from './routes/mainRoutes';
import cookieParser from 'cookie-parser';
import ErrorMiddleware from './middlewares/ErrorMiddleware';

env.config();
const app = express();

app.use(express.json());
app.use(cookieParser());
app.use(
  cors({
    origin: process.env.CLIENT_URL,
    credentials: true,
  }),
);
app.use(express.static(path.resolve('static')));
app.use(fileUpload({}));
app.use('/api', mainRoutes);
app.use(ErrorMiddleware);

const start = async () => {
  try {
```

```
    await sequelize.authenticate();
    await sequelize.sync();
    app.listen(process.env.PORT, () => {
      console.log(`App listening on port ${process.env.PORT}`);
    });
  } catch (error) {
    console.error(error);
  }
};

start();
```

AuthService.js

```
import bcrypt from 'bcrypt';
import * as uuid from 'uuid';
import { User } from '../models/Models';
import FileService from './FileService';
import TokenService from './TokenService';
import ApiError from '../exceptions/ApiError';
import UserService from './UserService';

class AuthService {
  async registration(user, avatar) {
    const personAlreadyExists = await User.findOne({
      where: { email: user.email },
    });
    if (personAlreadyExists) {
      throw ApiError.BadRequest(`The user with email
"${user.email}" already exists in the database`);
    }

    const hashPassword = await bcrypt.hash(user.password, 3);
    const activationLink = uuid.v4();
    const fileName = avatar ? FileService.saveFile(avatar) : null;
    user.password = hashPassword;

    const newUser = await User.create({
      ...user,
      avatar: fileName,
      activationLink,
    });

    return this.createTokens(newUser);
  }

  async activate(activationLink) {
    const user = await User.findOne({
      where: { activationLink: activationLink },
    });
    if (!user) {
      throw ApiError.BadRequest('Incorrect link for account
activation');
```

```
    }
    user.isActivated = true;
    await user.save();
  }

  async login(email, password) {
    const user = await User.findOne({
      where: { email: email },
    });
    if (!user) {
      throw ApiError.BadRequest(`User with this ${email} not
found`);
    }
    const isPassEquals = await bcrypt.compare(password,
user.password);
    if (!isPassEquals) {
      throw ApiError.BadRequest('Incorrect password');
    }

    return this.createTokens(user);
  }

  async logout(refreshToken) {
    const token = await TokenService.removeToken(refreshToken);
    return token;
  }

  async refresh(refreshToken) {
    if (!refreshToken) {
      throw ApiError.UnauthorizedError();
    }
    const userData =
TokenService.validateRefreshToken(refreshToken);
    const tokenFromDB = await
TokenService.findToken(refreshToken);
    if (!userData || !tokenFromDB) {
      throw ApiError.UnauthorizedError();
    }
    const user = await User.findOne({
      where: { id: userData.id },
    });

    return this.createTokens(user);
  }

  async createTokens(user) {
    const { id, email, isActivated } = user;
    const tokens = TokenService.generateTokens({ id, email,
isActivated });
    await TokenService.saveToken(id, tokens.refreshToken);
    return {
      ...tokens,
      user: UserService.makeUserDto(user),
```

```
    };
  }
}

export default new AuthService();
```

BudgetCategoryService.js

```
import { BudgetCategory } from "../models/Models";
import ApiError from "../exceptions/ApiError";
import sequelize from "../db";
import { QueryTypes } from "sequelize";

function checkID(id) {
  if (!id) throw ApiError.BadRequest("No id specified");
}

class BudgetCategoryService {
  async getAllBudgetCategoriesByLocation(locationId) {
    return await sequelize.query(
      `SELECT
  bc.id AS "categoryId",
  bc.title AS "categoryTitle",
  bil."locationId",
  bil."createdAt",
  bil."updatedAt"
FROM
  public.budget_categories bc
JOIN
  public.budget_in_locations bil
ON
  bc.id = bil."budgetCategoryId"
WHERE
  bil."locationId" = ${locationId}`,
      { type: QueryTypes.SELECT }
    );
  }

  async createBudgetCategory(category) {
    return await BudgetCategory.create(category);
  }

  async updateBudgetCategory(id, category) {
    checkID(id);
    delete category.id;
    await BudgetCategory.update({
      where: {
        id: id,
      },
    });

    return await BudgetCategory.findOne({
      where: { id: id },
```

```
    });
  }

  async deleteBudgetCategory(id) {
    checkID(id);
    return await BudgetCategory.destroy({
      where: {
        id: id,
      },
    });
  }
}

export default new BudgetCategoryService();
```

FileService.js

```
import { File } from '../models/Models';
import * as uuid from 'uuid';
import * as path from 'path';

function checkID(id) {
  if (!id) throw new Error('No id specified');
}

class FileService {
  async getAllFiles() {
    return await File.findAll();
  }

  async getFile(id) {
    checkID(id);
    return await File.findOne({
      where: { id: id },
    });
  }

  async createFile(file) {
    return await File.create(file);
  }

  async updateFile(file) {
    checkID(file.id);
    return await File.update(file, {
      where: {
        id: file.id,
      },
    });
  }

  async deleteFile(id) {
    checkID(id);
    return await File.destroy({
```

```
      where: {
        id: id,
      },
    });
  }

  saveFile(file) {
    try {
      const fileName = uuid.v4() + path.extname(file.name);
      file.mv(path.resolve('static', fileName));
      return fileName;
    } catch (error) {
      console.error(error);
    }
  }
}

export default new FileService();
```

LocationService.js

```
import { BudgetCategory, BudgetInLocation, Location } from
"../models/Models";
import FileService from "./FileService";
import ApiError from "../exceptions/ApiError";
import sequelize from "../db";
import { QueryTypes } from "sequelize";

function checkID(id) {
  if (!id) throw ApiError.BadRequest("No id specified");
}

class LocationService {
  async getAllLocations() {
    return await Location.findAll();
  }

  async getAllLocationsByTrip(tripId) {
    return await sequelize.query(
      `SELECT l.id, l.title, l.latitude, l.longitude,
l."createdAt", l."updatedAt"
      FROM public.locations l
      JOIN public.location_in_trips lit ON l.id = lit."locationId"
      WHERE lit."tripId" = ${tripId}`,
      {
        replacements: { tripId },
        type: QueryTypes.SELECT,
      }
    );
  }

  async createCategoryForLocation(locationId, categoryData) {
    return await sequelize.query(
```

```
      `WITH new_category AS (
         INSERT INTO public.budget_categories (title, "createdAt",
"updatedAt")
         VALUES ('${categoryData.title}', NOW(), NOW())
         RETURNING id
       )
       INSERT INTO public.budget_in_locations (amount,
"createdAt", "updatedAt", "budgetCategoryId", "locationId")
       VALUES (
         ${categoryData.amount},
         NOW(),
         NOW(),
         (SELECT id FROM new_category),
         ${locationId}
       )`,
      { type: QueryTypes.INSERT }
    );
  }

  async getCategoriesForLocation(locationId) {
    return await sequelize.query(
      `SELECT
      bc.id AS category_id,
      bc.title AS category_title,
      bc."createdAt" AS category_created_at,
      bc."updatedAt" AS category_updated_at,
      bil.amount AS category_amount
    FROM public.locations l
    JOIN public.budget_in_locations bil ON l.id = bil."locationId"
    JOIN public.budget_categories bc ON bil."budgetCategoryId" =
bc.id
    WHERE l.id = ${locationId}`,
      { type: QueryTypes.SELECT }
    );
  }

  async updateCategoryAmount(locationId, categoryId, newAmount) {
    const result = await BudgetInLocation.update(
      { amount: newAmount },
      {
        where: {
          location_id: locationId,
          budget_category_id: categoryId,
        },
      }
    );

    if (result[0] === 0) {
      throw new Error(
        "Обновление не произошло. Возможно, категория не найдена."
      );
    }
```

```
    return { locationId, categoryId, newAmount };
  }

  async deleteCategoryFromLocation(locationId, categoryId) {
    return await sequelize.query(
      `BEGIN;

DELETE FROM public.budget_in_locations
WHERE "locationId" = ${locationId}
  AND "budgetCategoryId" = ${categoryId};

DO $$
BEGIN
  IF (SELECT COUNT(*)
      FROM public.budget_in_locations
      WHERE "budgetCategoryId" = ${categoryId}) = 0 THEN
    DELETE FROM public.budget_categories
    WHERE id = ${categoryId};
  END IF;
END $$;

COMMIT;`,
      { type: QueryTypes.DELETE }
    );
  }

  async getLocation(id) {
    checkID(id);
    return await Location.findOne({
      where: { id: id },
    });
  }

  async createLocation(location) {
    const result = await Location.create(location);
    await sequelize.query(
      `INSERT INTO location_in_trips("createdAt", "updatedAt",
"tripId", "locationId") VALUES(NOW(), NOW(), ${location.trip_id},
${result.id})`,
      { type: QueryTypes.INSERT }
    );
    return result;
  }

  async updateLocation(id, location, file) {
    checkID(id);
    delete location.id;
    await Location.update(
      { file: file ? FileService.saveFile(file) : null,
...location },
      { where: { id: id } }
    );
```

```
    return await Location.findOne({ where: { id: id } });
  }

  async deleteLocation(id) {
    checkID(id);
    return await Location.destroy({ where: { id: id } });
  }
}

export default new LocationService();
```

TokenService.js

```
import jwt from 'jsonwebtoken';
const { sign, verify } = jwt;
import { Token } from '../models/Models';
import { JWT_ACCESS_SECRET_KEY, JWT_REFRESH_SECRET_KEY } from
'../consts/jwtConsts';

class TokenService {
  generateTokens(payload) {
    const accessToken = sign(payload, JWT_ACCESS_SECRET_KEY, {
      expiresIn: '10m',
    });
    const refreshToken = sign(payload, JWT_REFRESH_SECRET_KEY, {
      expiresIn: '30d',
    });
    return {
      accessToken,
      refreshToken,
    };
  }

  validateAccessToken(token) {
    try {
      const userData = verify(token, JWT_ACCESS_SECRET_KEY);
      return userData;
    } catch (error) {
      return null;
    }
  }

  validateRefreshToken(token) {
    try {
      const userData = verify(token, JWT_REFRESH_SECRET_KEY);
      return userData;
    } catch (error) {
      return null;
    }
  }

  async saveToken(userId, refreshToken) {
    const tokenData = await Token.findOne({
```

```
      where: { userId: userId },
    });
    if (tokenData) {
      tokenData.refreshToken = refreshToken;
      return tokenData.save();
    }
    return await Token.create({ userId: userId, refreshToken });
  }

  async removeToken(refreshToken) {
    return await Token.destroy({
      where: {
        refreshToken: refreshToken,
      },
    });
  }

  async findToken(refreshToken) {
    return await Token.findOne({
      where: {
        refreshToken: refreshToken,
      },
    });
  }
}

export default new TokenService();
```

TripService.js

```
import { Trip, TripItem } from "../models/Models";
import FileService from "./FileService";
import ApiError from "../exceptions/ApiError";

function checkID(id) {
  if (!id) throw ApiError.BadRequest("No id specified");
}

class TripService {
  async getAllTrips() {
    return await Trip.findAll();
  }

  async getTrip(id) {
    checkID(id);
    return await Trip.findOne({
      where: { id: id },
    });
  }

  async createTrip(trip, image) {
    const fileName = image && FileService.saveFile(image);
    return await Trip.create({ ...trip, image: fileName });
```

```
  }

  async updateTrip(id, trip, file) {
    checkID(id);
    delete trip.id;
    await Trip.update(
      { file: file ? FileService.saveFile(file) : null, ...trip },
      {
        where: {
          id: id,
        },
      }
    );

    return await Trip.findOne({
      where: { id: id },
    });
  }

  async deleteTrip(id) {
    checkID(id);
    return await Trip.destroy({
      where: {
        id: id,
      },
    });
  }

  async addTripItem(tripId, title) {
    if (!tripId) throw ApiError.BadRequest("Trip ID is required");
    return await TripItem.create({ tripId, title });
  }

  async updateTripItem(id, data) {
    if (!id) throw ApiError.BadRequest("No Trip Item ID
specified");
    delete data.id;
    await TripItem.update(data, { where: { id } });
    return await TripItem.findOne({ where: { id } });
  }

  async getTripItems(tripId) {
    if (!tripId) throw ApiError.BadRequest("Trip ID is required");
    return await TripItem.findAll({
      where: { tripId },
    });
  }

  async deleteTripItem(id) {
    if (!id) throw ApiError.BadRequest("No Trip Item ID
specified");
    return await TripItem.destroy({
      where: { id },
```

```
    });
  }
}

export default new TripService();
```

UserService.js

```
import { User } from '../models/Models';
import FileService from './FileService';
import ApiError from '../exceptions/ApiError';

function checkID(id) {
  if (!id) throw ApiError.BadRequest('No id specified');
}

class UserService {
  makeUserDto({ id, fullName, email, avatar, isActivated,
createdAt }) {
    return { id, fullName, email, avatar, isActivated, createdAt
};
  }

  async getAllUsers() {
    return (await User.findAll()).map((user) =>
this.makeUserDto(user));
  }

  async getUser(id) {
    checkID(id);
    return this.makeUserDto(
      await User.findOne({
        where: { id: id },
      }),
    );
  }

  async updateUser(id, user, avatar) {
    checkID(id);
    delete user.id;
    delete user.avatar;

    if (avatar) {
      user.avatar = FileService.saveFile(avatar);
    }

    await User.update(user, {
      where: {
        id: id,
      },
    });

    return this.makeUserDto(
```

```
      await User.findOne({
        where: { id: id },
      }),
    );
  }

  async deleteUser(id) {
    checkID(id);
    return await User.destroy({
      where: {
        id: id,
      },
    });
  }
}

export default new UserService();
```