МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА

Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»

Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»

галузь знань 12 Інформаційні технології
спеціальність 122 Комп'ютерні науки

Форма навчання: очна (денна)

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему: «Розроблення інформаційної управляючої системи для туристичної агенції»

здобувача Паплінського Владислава Вікторовича
(ПІБ, підпис)

Науковий керівник: кандидат економічних наук, доцент, Тішков Богдан Олександрович
(науковий ступінь, учене звання, ПІБ)

_____
(підпис)

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти (ЕК)

Завідувач кафедри: кандидат економічних наук, доцент, Тішков Богдан Олександрович
(науковий ступінь, учене звання, ПІБ)

_____
(підпис)

Київ 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»
галузь знань 12 Інформаційні технології
спеціальність 122 Комп'ютерні науки

ПОГОДЖЕНО: Керівник проєктної групи (гарант) освітньо-професійної програми	ЗАТВЕРДЖУЮ: Завідувач кафедри інформаційних систем в економіці
_____Устенко С.В. «____» _____2025 р.	_____ Тішков Б.О. «____» _____ 2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти Паплінського Владислава Вікторовича
(прізвище, ім'я, по батькові)
очної (денної) форми навчання

на підготовку кваліфікаційної магістерської роботи
на тему: *«Розроблення інформаційної управляючої системи для туристичної агенції»*

Тему затверджено наказом ректора Університету від «__» _____ 202_ р. №_____

Кваліфікаційна магістерська робота виконується на матеріалах
___тез з Додатку А_____

План кваліфікаційної магістерської роботи

Розділ І ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПІДХОДІВ ДО СТВОРЕННЯ ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ ПРЕДМЕТНОЇ ОБЛАСТІ
(назва розділу)
Розділ ІІ ХАРАКТЕРИСТИКА ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ ТА МЕТОДИ І МОДЕЛІ
(назва розділу)
Розділ ІІІ РОЗРОБЛЕННЯ ПРОЄКТНИХ РІШЕНЬ ТА ЇХ РЕАЛІЗАЦІЯ
(назва розділу)
Об'єкт дослідження: процеси управління діяльністю туристичної агенції, зокрема обробка замовлень, взаємодія з клієнтами, керування турами, локалізація та персоналізовані рекомендації
Предмет дослідження: методи та засоби автоматизації управлінських процесів у туристичному бізнесі з використанням інформаційно-управляючих систем (ІУС), включаючи застосування технологій штучного інтелекту, кешування та сучасних архітектур програмного забезпечення
Мета кваліфікаційної магістерської роботи: розроблення інформаційної управляючої системи для туристичної агенції, яка забезпечує автоматизацію основних бізнес-процесів з урахуванням сучасних вимог до зручності, масштабованості, продуктивності та інтелектуальної підтримки користувачів

Конкретні завдання, які здобувач повинен виконати для досягнення поставленої мети:

У розділі I _ провести аналіз поточного стану розвитку туристичної галузі в Україні та світі, визначити актуальні проблеми, виклики та тренди. Ознайомитися з існуючими інформаційними управляючими системами у сфері туризму, провести порівняльний аналіз їхніх функціональних можливостей. Визначити особливості впровадження цифрових технологій та інтелектуального аналізу даних у галузі. Обґрунтувати необхідність створення інтелектуальної інформаційної системи підтримки вибору туристичних турів, яка враховує вподобання користувачів та забезпечує адаптивність, масштабованість і простоту в обслуговуванні. Визначити критерії ефективності, які буде застосовано для оцінки розробленого ПЗ.

У розділі II _ розробити реляційну базу даних з урахуванням вимог предметної області, включаючи структуру таблиць, зв'язки та атрибути. Реалізувати підсистему зберігання зображень та мультимедійних матеріалів із використанням хмарного сховища Cloudinary. Впровадити кешування даних локалізацій за допомогою Redis для оптимізації швидкодії. Побудувати модуль інтелектуального аналізу даних, який формує рекомендації турів на основі попереднього досвіду користувача, з використанням API OpenAI. Створити та реалізувати структуру бази знань, яка використовує результати ІАД для підтримки прийняття рішень. Розробити і протестувати логіку інтелектуальної взаємодії між користувачем і системою. Реалізувати схему взаємодії компонентів та забезпечити зручну інтеграцію між модулями.

У розділі III_Обґрунтувати вибір технологій, інструментів і архітектурного підходу для реалізації системи. Реалізувати ПЗ на основі патерну MVC із розділенням проєкту на компоненти Client, Backend, Common та Tests. Інтегрувати компоненти системи з Redis, PostgreSQL, Cloudinary, OpenAI API. Налаштувати сервіси, DTO-моделі та систему маршрутизації запитів. Здійснити модульне тестування основних сервісів із використанням xUnit. Розробити адаптивний інтерфейс користувача з підтримкою кількох мов. Підготувати інструкцію користувача для кожної ролі: туриста, гіда, адміністратора. Оцінити ефективність реалізованого рішення та сформулювати практичні рекомендації щодо подальшого розвитку системи.

Завдання підготував
науковий керівник _____ _____
(підпис) *(ініціали, прізвище)*

«____» _____ 2025 р.

Завдання одержав
здобувач _____ _____
(підпис) *(ініціали, прізвище)*

«____» _____ 2025 р.

Реферат

Кваліфікаційна магістерська робота містить 120 сторінок, 11 таблиць, 38 рисунків, список використаних джерел з 50 найменувань, додатки.

«Розроблення інформаційної управляючої системи для туристичної агенції»

Об'єктом дослідження кваліфікаційної магістерської роботи є процеси управління діяльністю туристичної агенції, зокрема обробка замовлень, взаємодія з клієнтами, керування турами, локалізація та персоналізовані рекомендації.

Предметом дослідження є методи та засоби автоматизації управлінських процесів у туристичному бізнесі з використанням інформаційно-управляючих систем (ІУС), включаючи застосування технологій штучного інтелекту, кешування та сучасних архітектур програмного забезпечення.

Мета і завдання дослідження. Розроблення інформаційної управляючої системи для туристичної агенції, яка забезпечує автоматизацію основних бізнес-процесів з урахуванням сучасних вимог до зручності, масштабованості, продуктивності та інтелектуальної підтримки користувачів.

Відповідно до поставленої мети визначені такі *завдання*:

- проаналізувати існуючі підходи до автоматизації управлінських процесів у туристичних інформаційних системах;
- спроєктувати архітектуру інформаційної управляючої системи на основі сучасних технологій;
- розробити модулі для керування турами, бронюванням, локалізацією, а також систему персоналізованих рекомендацій на основі ШІ;
- реалізувати функціонал для роботи з різними ролями користувачів;
- провести інтеграцію з хмарними сервісами та інструментами кешування для підвищення продуктивності системи;
- забезпечити тестування, демонстрацію результатів роботи інформаційної управляючої системи;
- підготувати документацію для користувачів і розробників.

Теоретична, методична та практична значущість отриманих результатів. Теоретична значущість полягає в застосуванні сучасних підходів до реалізації інтелектуальних інформаційних систем, зокрема інтеграції з LLM-моделями для генерації рекомендацій. Методична значущість полягає у поєднанні ролевої моделі доступу, кешування локалізацій та шаблонів програмування (MVC, DI, Repository) у єдиному архітектурному рішенні. Практична значущість підтверджується можливістю використання розробленої інформаційної управляючої системи у невеликих турагенціях як готового рішення або бази для подальшого вдосконалення.

Рік виконання кваліфікаційної магістерської роботи – 2025.

Рік захисту роботи – 2025.

Ключові слова: інформаційна система, туризм, рекомендації, кешування, локалізація, автоматизація, ролева модель.

В і д г у к

про кваліфікаційну магістерську роботу
здобувача навчально-наукового інституту «Інститут інформаційних технологій в економіці»
освітньо-професійної програми «Інформаційні управляючі системи і технології»
Паплінського Владислава Вікторовича
(прізвище, ім'я, по батькові)

на тему «Розроблення інформаційної управляючої системи для туристичної агенції»

1. Актуальність теми: Тема є надзвичайно актуальною з огляду на потребу в автоматизованому підборі туристичних продуктів, персоналізованих рекомендаціях та ефективному управлінні контентом у сфері туризму. Тема відповідає сучасним тенденціям цифрової трансформації галузі, розвитку штучного інтелекту та хмарних технологій.
2. Позитивні риси кваліфікаційної магістерської роботи: Автор показав високий рівень технічної підготовки, вміло реалізував бізнес-логіку системи, а також забезпечив її масштабованість та адаптивність до розвитку.
3. Наявність самостійних розробок автора Серед помітних самостійних напрацювань автора — реалізація системи рекомендацій турів із використанням зовнішнього ШІ API, продумана архітектура з розділенням на Client, Backend та Common частини, кешування локалізацій у Redis, збереження мультимовного контенту та динамічне формування запитів.
4. Цінність теоретичних висновків та практичних рекомендацій: Теоретичні висновки мають цінність у контексті організації баз знань, інтелектуального аналізу даних і застосування сучасних шаблонів архітектури ПЗ. Практичні рекомендації можуть бути використані для розгортання та масштабування подібних інформаційних систем у галузі туризму або інших сферах
5. Наявність недоліків: Деякі частини реалізації потребують уточнення або подальшого розвитку, зокрема захист API-інтерфейсів, покриття всіх модулів інтеграційними тестами, а також впровадження повноцінної аналітики продуктивності системи
6. Загальна оцінка кваліфікаційної магістерської роботи та її допущення до захисту перед ЕК: Робота відповідає вимогам до випускної магістерської роботи, демонструє глибокі знання автора з предметної області та високу компетентність у сфері розробки ПЗ

Науковий керівник кандидат економічних наук, доцент, Тішков Богдан Олександрович
(посада, учене звання, науковий ступінь)

_____ _____
(підпис) *(прізвище, ініціали)*
«___»_____ 2025 р.

Рецензія
на кваліфікаційну магістерську роботу
здобувача вищої освіти
Паплінського Владислава Вікторовича
(прізвище, ім'я, по батькові)

Тема Розроблення інформаційної управляючої системи для туристичної агенції

Актуальність теми кваліфікаційної магістерської роботи і доцільність її розроблення Тема роботи є актуальною в умовах розвитку цифрових сервісів у туристичній галузі. Доцільність розробки інформаційної управляючої системи з використанням технологій ШІ, хмарних сервісів та персоналізованих рекомендацій цілком обґрунтована.

Якість проведеного дослідження Дослідження проведено якісно, з дотриманням структурованого підходу до проєктування, реалізації та тестування. Автор охопив усі етапи життєвого циклу ПЗ, використав сучасний стек технологій, забезпечив наочність та функціональність розробленої системи

Позитивні риси кваліфікаційної магістерської роботи Вдале поєднання практичного програмування з елементами інтелектуального аналізу даних. Програма реалізована з урахуванням архітектурних шаблонів (MVC, DI), має модульну структуру, містить автоматизовані тести, застосовує кешування та зовнішні API.

Зауваження Деякі частини системи можуть бути розширені — зокрема, інтеграційне тестування, моніторинг, система захисту API. Також можна покращити зберігання результатів ШІ-аналізу для подальшої аналітики.

Практична значимість висновків і рекомендацій Отримані результати можуть бути застосовані при створенні інформаційних систем у сфері туризму та подібних галузях. Система має потенціал до масштабування та адаптації під реальні потреби користувачів.

Місце роботи та посада рецензента
Науковий ступінь, учене звання (*за наявності*) _____
(підпис, ПІБ)

Підпис засвідчую: _____
(посада, підпис)

ЗМІСТ

ВСТУП

У сучасних умовах розвитку інформаційних технологій автоматизація бізнес-процесів стає ключовим фактором підвищення ефективності підприємств. Туристична галузь не є винятком, оскільки забезпечення оперативного управління даними про клієнтів, тури, бронювання та фінансові операції є критично важливим для конкурентоспроможності туристичних агенцій. Традиційні методи обробки інформації часто є неефективними та призводять до втрат часу і ресурсів. Саме тому розроблення інформаційно-управлінської системи (ІУС) для туристичної агенції є актуальною задачею, що сприяє підвищенню продуктивності, покращенню клієнтського обслуговування та оптимізації бізнес-процесів.

Наукова проблема, яка розглядається у даній роботі, полягає у відсутності комплексного підходу до автоматизації управління туристичною агенцією з урахуванням сучасних технологій збереження, обробки та аналізу даних.

Метою даного дослідження є розроблення інформаційно-управлінської системи для туристичної агенції, яка дозволить автоматизувати ключові бізнес-процеси та підвищити ефективність її діяльності. Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати сучасні підходи до автоматизації управлінських процесів у туристичній сфері.
2. Визначити вимоги до інформаційної системи туристичної агенції.
3. Розробити концептуальну модель інформаційно-управлінської системи.
4. Реалізувати та протестувати розроблену систему.
5. Оцінити ефективність впровадженої системи та сформулювати рекомендації щодо її подальшого вдосконалення.

Об’єктом дослідження є процеси управління діяльністю туристичної агенції. Предметом дослідження є методи та засоби автоматизації управління туристичним бізнесом на основі інформаційно-управлінських систем.

У процесі дослідження використовувалися такі методи:

– Аналіз літературних джерел та існуючих рішень у сфері автоматизації туризму.

– Системний аналіз та проектування інформаційних систем.

– Методологія об’єктно-орієнтованого програмування для реалізації системи.

– Методи тестування програмного забезпечення для оцінки його функціональності та продуктивності.

Результати дослідження можуть бути використані в діяльності туристичних агентств для покращення управлінських процесів. Прототип системи був апробований у реальній туристичній агенції, що підтвердило його ефективність у прискоренні обробки заявок, зменшенні помилок та покращенні взаємодії з клієнтами.

Кваліфікаційна магістерська робота складається з трьох розділів. У першому розділі розглянуто теоретичні аспекти автоматизації управління туристичними агентствами. У другому розділі описано процес розробки інформаційно-управлінської системи, включаючи аналіз вимог, проектування та реалізацію. Третій розділ містить результати тестування системи та оцінку її ефективності. Завершується робота висновками та рекомендаціями щодо подальшого вдосконалення розробленої системи.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПІДХОДІВ ДО СТВОРЕННЯ ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз існуючих інформаційних управляючих систем (предметної області)

1.1.1 Дослідження предметної області

Туризм — це комплексна соціально-економічна діяльність, що охоплює організацію подорожей та відпочинку людей, сприяючи культурному обміну, економічному розвитку та зміцненню міжнародних зв'язків. Він включає в себе різноманітні види діяльності: від транспортних перевезень і готельного бізнесу до екскурсійних послуг та гастрономії. Це є видом послуг, який пов’язаний з тимчасовим виїздом особи з місця її постійного проживання з оздоровчою, пізнавальною, діловою або іншою метою без здійснення оплачуваної діяльності у місці перебування.[1]

У глобальному контексті туризм є одним із найбільших секторів економіки, забезпечуючи мільйони робочих місць і значні надходження до бюджетів країн. Він сприяє розвитку інфраструктури, збереженню культурної спадщини та стимулює інвестиції в регіони. Глобальний туризм включає в себе подорожі на найбільшу відстань між країнами та континентами. Він охоплює туризм для відпочинку, бізнесу, культурного обміну, медичного туризму та багатьох інших аспектів. Цей вид діяльності має величезний вплив на світову економіку, суспільство та культуру.[2]

В Україні туристична галузь має значний потенціал, зважаючи на багатство природних ресурсів, культурну спадщину та географічне розташування. До початку війни туризм був важливою складовою економіки, сприяючи розвитку регіонів та наповненню бюджету. Валова додана вартість, що створювалася в туризмі у 2019 році досягла 12,62% ВВП України, у 2020 році дещо знизилась, а за 2021 рік, коли тривали карантинні обмеження, складала вже 11,82% [3]. Початок вторгнення Російської Федерації на

територію України завдало дуже болісного удару по всім жителям країни, та сильно вдарило по українській економіці. Серед економічних секторів однією з набільш постраждалих є туристична галузь [4]. Хоча наразі і зважається, що значного удару по галузі туризму в Україні завдало саме повномасштабне вторгнення Росії у 2022 році (що є правдою), втім не можна забувати, що нестабільна ситуація в Україні та початок російського вторгнення в 2014 році також було одним з важких ударів по туризму в Україні, що видно у порівняльній таблиці з кількості міжнародних туристів в Україні по рокам (див. рис. 1.1) [5].

Рисунок 1.1 – Порівняльний аналіз туристичних потоків та економічного розвитку в Україні (1995–2023) [5][6][7]

На основі рисунку 1.1 можна бачити, як негативно вплинули на міжнародних туристів в Україні такі події, як Світова криза у 2008 році, Анексія Криму та Війна на Донбасі в 2014 році, Світова пандемія Коронавірусу у 2020 році, та, врешті-решт, Повномасштабне вторгнення Росії в Україну у 2022 році. Війна суттєво вплинула на туристичну галузь. За оцінками ЮНЕСКО, втрати внаслідок руйнування культурних об'єктів та

зниження туристичних потоків перевищують $19 мільярдів. Зруйновано понад 300 культурних об'єктів, включаючи історичні пам'ятки в Києві, Львові та Одесі [8].

Втім, незважаючи на виклики, пов'язані з війною та пандемією, українська туристична галузь демонструє ознаки відновлення. Порівняно з 2022 роком, податкові надходження від тургалузі зросли на 13%, що свідчить про поступове відновлення галузі. Також спостерігається збільшення надходжень від готелів: найбільша частка податкових надходжень в туристичні галузі (63%) за 9 місяців надійшла від готелів - майже 916,65 млн грн. Це на 19,8% більше, ніж у 2022 році, і на 10,4% більше, ніж у 2021 році. [9]. У 2024 році туристична сфера України продемонструвала позитивну динаміку, принісши до державного бюджету майже 3 мільярди гривень податків [10]. Окрім того, хоча і відбувалися падіння в туризмі по багатьом напрямкам після початку повномасштабного вторгнення, по деяким напрямкам, наприклад Греція, спостерігається навпаки приплив українських туристів [19]. Це свідчить про можливості відновлення галузі при покращенні ситуаї та її здатність адаптуватися до нових умов

Управлінські рішення в ІУС для туристичної агенції охоплюють:

– формування туристичного продукту, а саме розробка турів, визначення цільових аудиторій, ціноутворення;

– маркетинг та просування, що включає в себе вибір каналів комунікації, рекламні кампанії, участь у виставках;

– логістика та обслуговування клієнтів, це включає організацію перевезень, бронювання, забезпечення якості послуг, які можуть бути описані та керовані через ІУС;

– фінансове планування, а саме бюджетування, аналіз витрат та доходів, оптимізація ресурсів;

– управління персоналом: набір, навчання та мотивація працівників.

Об'єктами дослідження в контексті ІУС для туристичної агенції є:

– бізнес-процеси агенції, включаючи продаж турів, обробку замовлень, взаємодію з постачальниками;

– інформаційні потоки, обмін даними між відділами, з клієнтами та партнерами;

– клієнтська база: збір, зберігання та аналіз інформації про клієнтів для персоналізації послуг;

– фінансові операції: облік платежів, витрат, прибутків;

– маркетингові дані: аналіз ефективності рекламних кампаній, вивчення ринку.

В вищеперелічених умовах, в яких наразі перебуває країна, та буде перебувати ще достатньо тривалий час, ефективність роботи підприємства туристичної галузі є не просто приємним бонусом для користувачів ІУС, включаючи й сам персонал туристичної агенції, а є справжньою необхідністю. Ефективне управління цими об'єктами забезпечує конкурентоспроможність агенції, підвищує рівень обслуговування клієнтів та сприяє зростанню прибутковості. Своєчасне та обґрунтоване прийняття управлінських рішень є критично важливим. Інформаційні управлінські системи дозволяють автоматизувати процеси, забезпечити доступ до актуальних даних та підтримати стратегічне планування.

Виклики, перед якими стоїть туристична галузь в Україні, є значними. Війна, що зараз відбувається, завдає значних збитків по багатьом параметрам, а саме: відтік людей за кордон, зменшення економічної привабливості галузі, страх потенційних туристів відвідувати країну, що перебуває у стані гарячої війни, зменшення економічної купівельної спроможності громадян України, тощо. Водночас, війна в країні створює ризики для інфраструктури та пам’яток України через ризик їх пошкодження та знищення ворожими снарядами, а це зменшує туристичну привабливість пам’яток архітектури і природи та туристичних місць країни.

Втім, у контексті повоєнного відновлення туристичної галузі України важливо враховувати принципи сталого розвитку. Значення туризму для суспільства, зокрема в період війни, підкреслює його роль у досягненні цілей сталого розвитку завдяки економічній, соціально-культурній та екологічній ролі.

Можливими напрямками повоєнного відновлення сфери туризму в Україні є:

– розвиток ділового, освітнього, спортивного, медичного, зеленого та воєнного туризму;

– розробка та реалізація програм (як державних, так і міжнародних) підтримки суб'єктів туристичної діяльності;

– забезпечення тісної співпраці між країнами у напрямку взаємопідтримки туристичної діяльності;

– підвищення інтересу іноземних туристів до пам'ятних маршрутів та повоєнних символічних місць;

– пошук нових методів та способів реалізації туристичних послуг тощо.[11]

Таким чином, туристична галузь України, незважаючи на виклики, пов'язані з війною, має потенціал для відновлення та подальшого розвитку. Це вимагає стратегічного планування, впровадження інноваційних підходів та активної співпраці між державними органами, бізнесом та громадськістю.

1.1.2 Дослідження інформаційних управляючих систем

Сучасні інформаційні управляючі системи (ІУС), що використовуються туристичними агентствами, забезпечують комплексну автоматизацію бізнес-процесів — від обробки запитів клієнтів до аналізу продажів, управління платежами й формування маршрутів. У цьому підрозділі проведено аналіз актуальних рішень, що вже застосовуються на практиці в галузі, з метою виявлення їх сильних і слабких сторін, що дозволить обґрунтувати архітектурні, функціональні та технологічні рішення при розробленні власної ІУС у межах проєкту

Для дослідження ІУС для туристичних агенцій ми маємо визначити, які саме функції має виконувати програма, аби вважатися повноцінною інформаційною системою управління. Функції інформаційних систем охоплюють різноманітні аспекти діяльності підприємств, зокрема в туристичній індустрії, де ефективне управління інформацією є ключовим чинником успіху, зокрема:

1. Збір даних – одна з основних функцій інформаційних систем, яка включає акумуляцію даних із різних джерел. У туристичному бізнесі це можуть бути дані про бронювання, клієнтів, туристичні маршрути, погодні умови та інші важливі чинники.

2. Зберігання даних. У туризмі, наприклад, це можуть бути дані про клієнтів, історію їхнього бронювання, відгуки, інформація про відвідані місця та пройдені маршрути.

3. Обробка даних – ключова функція інформаційних систем, яка полягає в тому, що зібрані дані перетворюються в корисну інформацію. У туристичній галузі вона може включати обробку даних про попит на туристичні послуги, аналіз тенденцій у поведінці клієнтів та прогнозування сезонних коливань.

4. Аналіз даних. Для туристичних компаній аналітичні функції ІС можуть використовуватися для вивчення ринкових тенденцій, поведінки

клієнтів, ефективності маркетингових кампаній або прогнозування попиту на послуги.

5. Передача даних та обмін ними. Для туристичних компаній це означає можливість миттєвого оновлення інформації про послуги, готелі, тури та спеціальні пропозиції для клієнтів.

6. Моніторинг і контроль. У туристичній індустрії це може бути моніторинг статусу бронювання, поточного завантаження готелів, контроль за станом транспортних засобів або інфраструктури.

7. Автоматизація рутинних процесів – це одна із основних функцій інформаційних систем, яка включає обробку бронювань, облік фінансових операцій, підготовку звітів, управління ресурсами тощо. Ця функція дозволяє зменшити кількість помилок, зумовлених "людським чинником", скоротити витрати на обробку інформації та підвищити ефективність роботи.

8. Підтримка прийняття рішень. Даний інтструментарій допомагає менеджерам ухвалювати обґрунтовані рішення, базуючись на точних даних та прогнозах, що надзвичайно важливо в туристичному бізнесі, де швидке реагування на зміни ринку може бути ключем до успіху.[12]

Маючи на увазі функціонал, що має бути присутнім у системі ми можемо тепер проаналізувати існуючі рішення.

TravelOperations

Перший в списку існуючих ІУС є TravelOperations (див. рис.1.2).

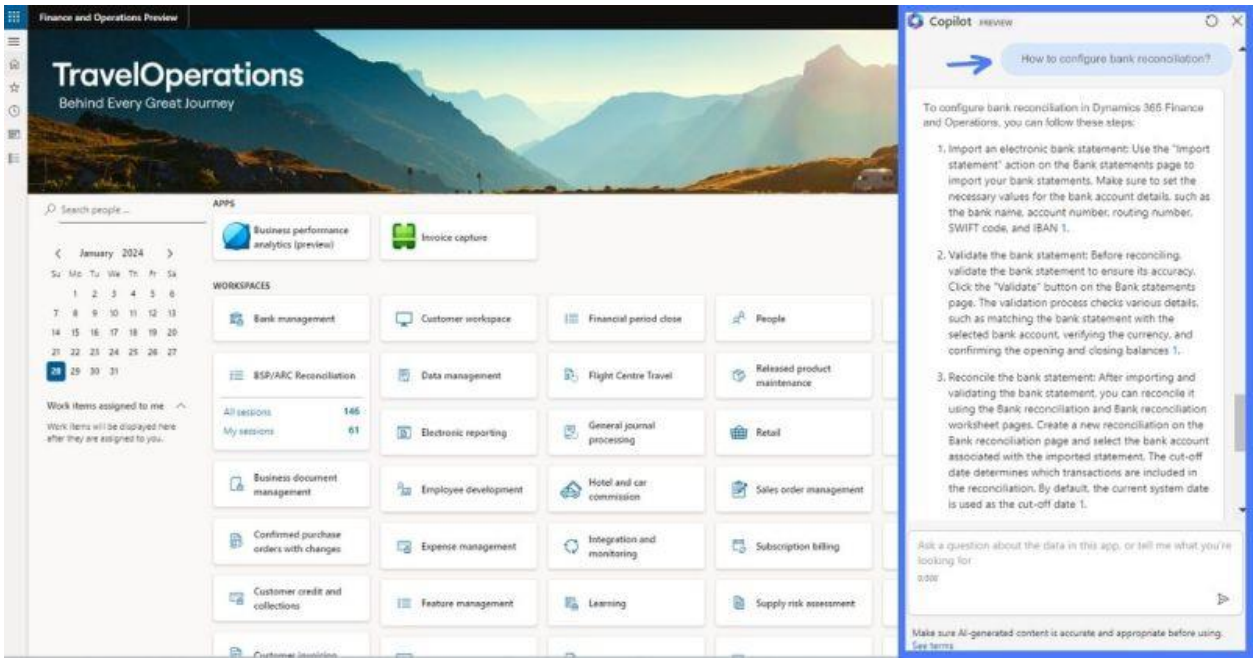


Рисунок 1.2 – ІУС «TravelOperations» [47]

TravelOperations — це комплексне рішення для туристичних агентств, яке поєднує функціональність CRM (англ. Customer Relationship Management, управління відносинами з клієнтами) та ERP (англ. Enterprise Resource Planning, планування ресурсів підприємства). Система розроблена на базі Microsoft Dynamics 365 і забезпечує автоматизацію фінансових операцій, управління клієнтами, бронюваннями та іншими бізнес-процесами. [13]

Ключовими функціями системи є:

– CRM: управління клієнтськими даними, історією взаємодій, персоналізація послуг;

– ERP: автоматизація фінансових процесів, облік, звітність;

– інтеграція: Сумісність з GDS та іншими сторонніми системами;

– аналітика: інструменти для аналізу продажів, продуктивності та інших показників.

До переваг можна віднести гнучкість та масштабованість завдяки платформі Microsoft Dynamics 365, та можливість адаптації під потреби різних агентств — від малих до великих. [14]

Серед недоліків ІУС є її складність впровадження та необхідність навчання персоналу.

WeTravel

Наступною ІУС є WeTravel (див.рис. 1.3).

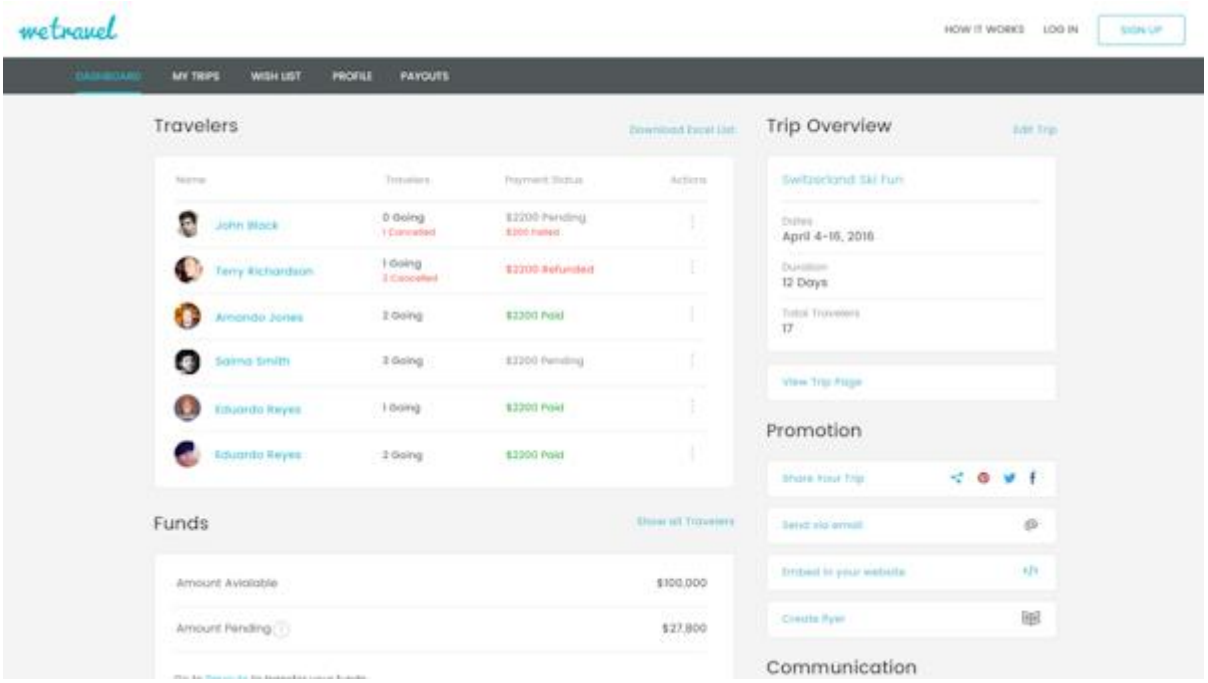


Рисунок 1.3 – ІУС «WeTravel» [48]

WeTravel — це платформа для туристичних агентств, яка забезпечує управління бронюваннями, платежами та створенням маршрутів. Система орієнтована на групові подорожі, ретріти та індивідуальні тури. [15]

Ключовими функціями системи є:

– бронювання: управління груповими та індивідуальними бронюваннями;

– платежі: прийом міжнародних платежів, автоматичні нагадування, планування платежів;

– маршрути: створення маршрутів з можливістю спільного редагування;

– інтеграція: інтеграція з веб-сайтами агентств через віджети.

Серед переваг системи є простота використання та швидке впровадження, а також можливість обробки міжнародних платежів з мінімальними комісіями.

Недоліки системи полягають у обмеженій функціональності для великих агентств з комплексними потребами.

Tern

Третьою системою є Tern (див. рис. 1.4).

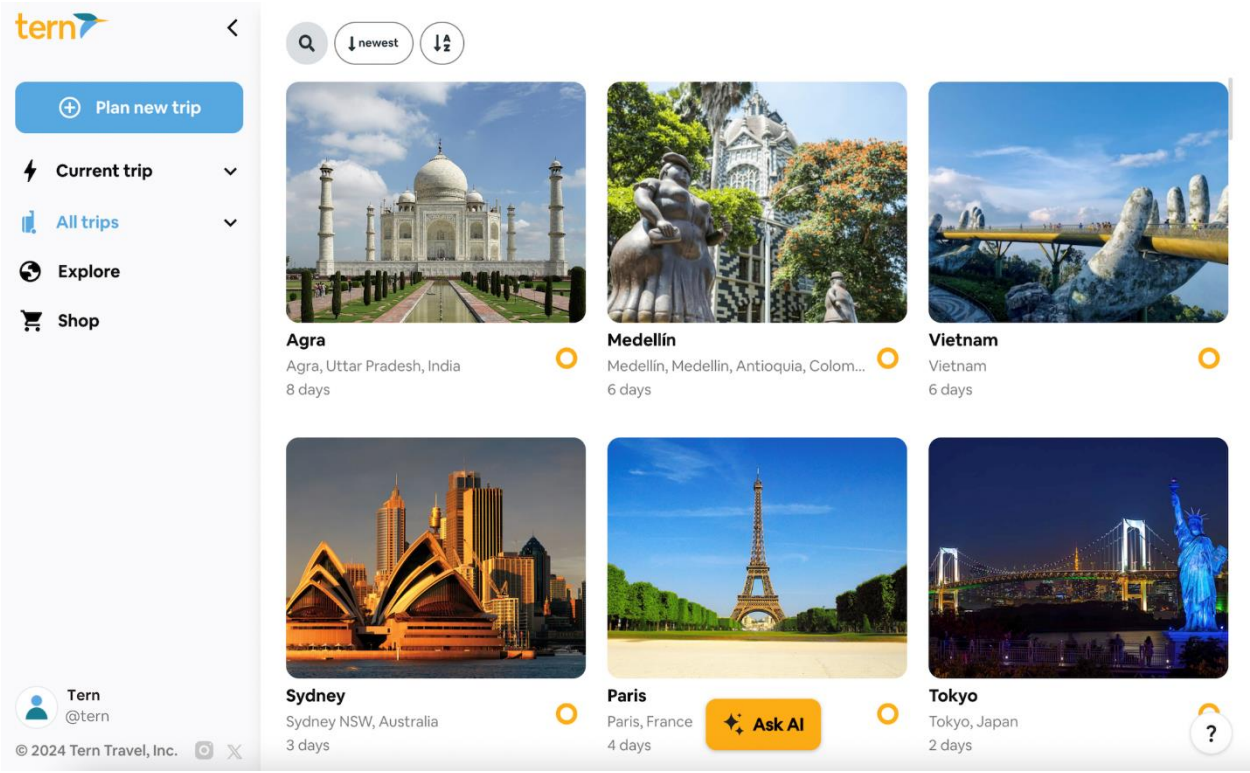


Рисунок 1.4 – ІУС «Tern» [49]

Tern — це сучасна платформа для туристичних агентів, яка об'єднує CRM, інструменти для створення маршрутів, управління комісіями та автоматизацію завдань. Система орієнтована на підвищення ефективності роботи агентів та покращення взаємодії з клієнтами. [16]

Ключові функції:

– CRM: управління контактами, історією подорожей, вподобаннями клієнтів;

– маршрути: створення маршрутів з можливістю інтерактивного вибору опцій клієнтами;

– фінанси: відстеження комісій, прогнозування доходів, управління платежами;

– автоматизація: інтеграція з календарями, автоматичне створення завдань, AI-асистент для обробки документів.

Серед переваг системи є інтуїтивно зрозумілий інтерфейс та сучасний дизайн та можливість масштабування та адаптації під потреби агентства.

Недолік можна виділити у тому, що це відносно новий продукт на ринку, що може впливати на стабільність та підтримку.

ClientBase

Четвертою ІУС в черзі є ClientBase (див. рис. 1.5).

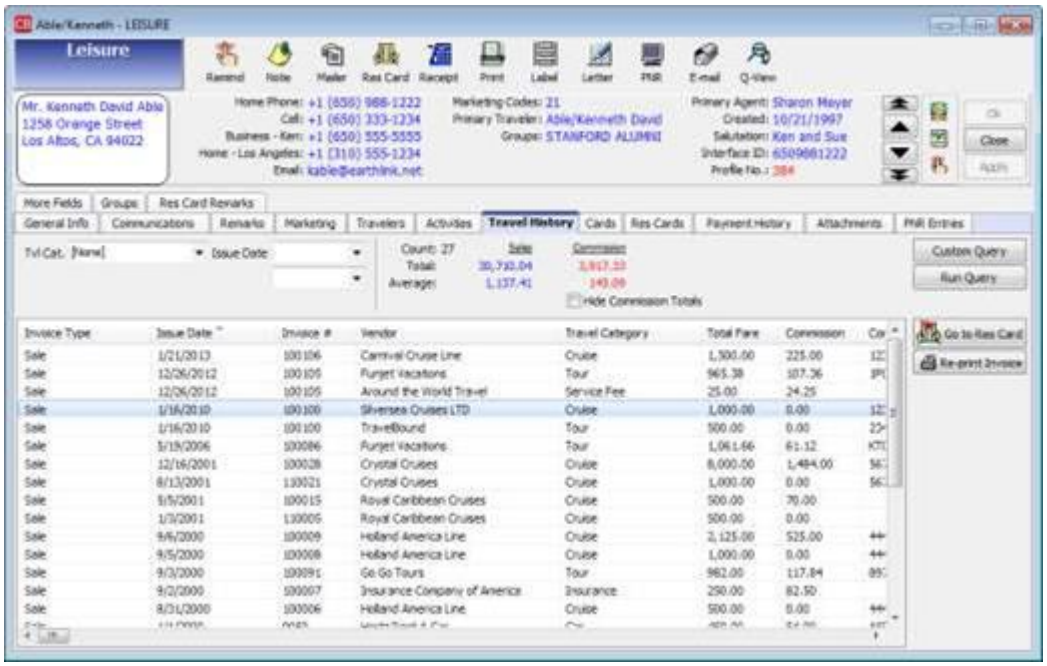


Рисунок 1.5 – ІУС « ClientBase» [50]

ClientBase — це спеціалізована CRM-система для туристичних агентств, яка забезпечує управління клієнтськими даними, бронюваннями та комунікацією. Система є частиною Sabre Red Revenue Suite і орієнтована на покращення взаємодії з клієнтами. [17]

Ключові функції:

– CRM: зберігання детальної інформації про клієнтів, історії подорожей, вподобання;

– бронювання: інтеграція з GDS для управління бронюваннями;

– маркетинг: інструменти для створення маркетингових кампаній та аналізу ефективності;

– звіти: генерація звітів для аналізу продажів, продуктивності агентів.

Перевагами системи є глибока інтеграція з системами бронювання та можливість персоналізації обслуговування клієнтів.

Недоліком системи є її застарілість у порівнянні з більш сучасними аналогами, інтерфейс може здаватися застарілим, а також тут відсутні багато сучасних рішень, які є в інших ІУС.

Travefy

Останньою в цьому списку ІУС є Travefy (див. рис. 1.6).

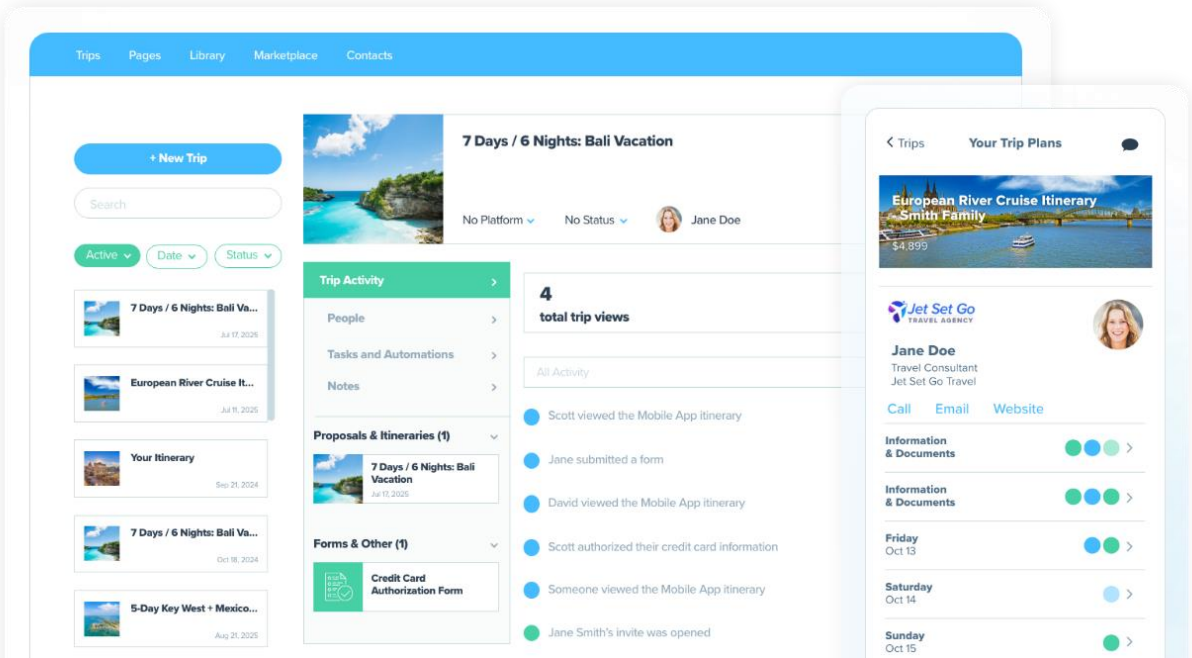


Рисунок 1.6 – ІУС « Travefy» [18]

Travefy — це веб-платформа, яка дозволяє агентам швидко створювати індивідуальні туристичні маршрути, інтерактивні подорожні плани, ділитися

ними з клієнтами та співпрацювати над їх редагуванням. Вона не є повноцінною CRM чи ERP-системою, однак може використовуватись у зв’язці з такими для покращення клієнтського досвіду. [18]

Ключові функції:

- планування подорожей: швидке створення детальних маршрутів з інтеграцією фото, мап, описів і посилань;
- інтерактивність: можливість клієнтам переглядати та взаємодіяти з маршрутами на своїх пристроях;
- брендинг: підтримка фірмового стилю агентства у кожній презентації;
- шаблони: доступ до готових шаблонів для різних типів подорожей;
- звіти та аналітика: базові інструменти для перегляду статистики по переглядах маршрутів.

Перевагами є естетична візуальна привабливість, інтеграція з іншими системами CRM, email-розсилками та вебсайтами, а також повна підтримка мобільних пристроїв.

Серед недоліків є обмежений набір управлінських функцій, відсутність CRM, аналітики, роботи з бронюванням та платежами, а також дана система не підходить як автономне рішення для комплексного управління туристичним агентством.

Аналіз сучасних інформаційних управляючих систем для туристичних агентств показав, що кожна з них має свою спеціалізацію: одні зосереджені на візуалізації маршрутів (Travefy), інші — на управлінні клієнтами (ClientBase), або повній бізнес-автоматизації (TravelOperations). Проте універсального рішення, яке б повністю закривало потреби невеликого туристичного агентства, не виявлено.

Переважна більшість систем побудовані за веб-архітектурою, підтримують інтеграцію з зовнішніми сервісами, мають функції автоматизації, CRM і базової аналітики. Це визначає загальні технологічні вимоги та функціональну модель, яку варто врахувати при проєктуванні власної ІУС.

Таким чином, доцільно створити систему, яка поєднує сильні сторони існуючих рішень, адаптована до українського ринку, із простою інтеграцією, доступною аналітикою та можливістю гнучкого масштабування.

1.2 Обґрунтування вибору підходів і технологій для створення інформаційної управляючої системи

У сучасних умовах цифрової трансформації та швидкого розвитку інформаційних технологій створення ефективної інформаційної управляючої системи (ІУС) для туристичної агенції потребує ретельного вибору концептуальних підходів, які забезпечать не лише автоматизацію рутинних процесів, а й створять можливості для персоналізації обслуговування, адаптації до мінливих умов ринку та ефективного прийняття управлінських рішень.

З урахуванням специфіки предметної області було визначено такі базові вимоги до ІУС:

– доступність для користувача будь-якого рівня цифрової підготовки (через веб-інтерфейс);

– підтримка роботи з кількома типами користувачів: переглядач, клієнт, гід, керівник;

– автоматизація основних бізнес-процесів: перегляд, фільтрація, бронювання турів, управління контентом;

– наявність модуля персоналізованих рекомендацій;

– гнучка підтримка багатомовності (локалізації);

– просте адміністрування даних (локації, категорії турів, користувачі).

Виходячи з цих вимог, було прийнято низку концептуальних рішень, які охоплюють методологічний, технологічний і організаційний рівні.

На методологічному рівні система орієнтована на модульну архітектуру, що дозволяє ізольовано розробляти, тестувати й масштабувати окремі компоненти — перегляд турів, управління заявками, рекомендації, управління локалізацією тощо. Такий підхід дозволяє не лише спростити підтримку та оновлення системи, а й забезпечити її адаптивність до майбутніх змін. Крім того, модель взаємодії між компонентами побудована з

урахуванням принципів MVC (Model-View-Controller), що покращує розділення відповідальності та забезпечує логічну структуризацію коду.

Особливу увагу приділено підтримці прийняття рішень користувачем. У систему інтегровано AI-модуль, який аналізує історію замовлень та популярність турів і формує персоналізовані рекомендації. Таким чином реалізовано гібридний підхід до підтримки прийняття рішень: rule-based фільтрація для обробки структурованих параметрів та AI-допомога для слабоструктурованих запитів.

У ролі платформи для реалізації веб-системи було обрано стек технологій на основі .NET, зокрема ASP.NET Core та Blazor Server, що дозволяє створювати сучасні інтерактивні веб-застосунки з урахуванням високої продуктивності та безпеки.

Дані зберігаються у базі PostgreSQL з урахуванням її надійності, продуктивності та широкої підтримки у спільноті. Для реалізації кешування та оптимізації швидкодії планується використання Redis.

Штучний інтелект інтегровано за допомогою зовнішнього API — OpenAI GPT, який обробляє текстові запити користувача та пропонує відповідні тури на основі історичних даних.

Інтерфейс користувача реалізований з урахуванням принципів UX/UI-дизайну: система підтримує зміну мови, адаптивна для мобільних пристроїв, має інтуїтивну структуру і мінімальну кількість кроків для виконання основних дій (пошук, перегляд, бронювання).

З точки зору організації розробки, було обрано гнучкий підхід на основі інкрементної моделі. Спочатку реалізовано MVP (мінімально життєздатний продукт), який включає перегляд, фільтрацію, бронювання турів та управління ними, після чого було додано модулі локалізації та AI-рекомендацій. Таким чином забезпечується поетапне розширення функціональності без порушення цілісності системи.

Вибір засобів розробки здійснювався за критеріями відкритості, популярності у спільноті, наявності документації та підтримки

масштабування. Усі компоненти мають відкриту ліцензію або безкоштовні умови для використання в освітньо-дослідницьких цілях.

Запропоновані рішення оцінювалися за такими критеріями: відповідність функціональним вимогам, простота реалізації, підтримка масштабованості, продуктивність, сумісність з AI-компонентами.

Таким чином, обрана концепція ІУС дозволяє ефективно вирішувати завдання цифровізації бізнес-процесів туристичної агенції, забезпечує персоналізацію сервісу, підтримує масштабованість, адаптацію до різних ролей користувачів та гнучку інтеграцію інтелектуальних технологій. У наступних розділах буде деталізовано логіку роботи системи, інформаційні потоки та структуру бази даних.

РОЗДІЛ 2 ХАРАКТЕРИСТИКА ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ ТА МЕТОДИ І МОДЕЛІ

2.1 Структура і характеристика системи

Розроблювана інформаційна управляюча система (ІУС) для туристичної агенції є сучасним веб-застосунком, побудованим на принципах модульності, розмежування доступу за ролями та інтеграції інтелектуальної підтримки користувача. Основна мета системи — цифрове забезпечення процесів підбору, замовлення та адміністрування туристичних турів, з урахуванням специфіки взаємодії між туристами, гідами та керівництвом агенції.

У системі передбачено чотири типи користувачів, кожен з яких має чітко визначений обсяг функціональності:

- переглядач — неавторизований користувач, який має доступ до загального переліку турів, може ознайомитися з деталями та скористатися базовою фільтрацією;
- клієнт — зареєстрований користувач, що має розширений функціонал: можливість подавати заявки на участь у турах, отримувати персоналізовані рекомендації, переглядати та скасовувати свої замовлення;
- гід — фахівець, який створює тури, керує їхнім вмістом, а також переглядає заявки від клієнтів на свої тури й може скасувати сам тур;
- керівник — особа, відповідальна за редагування довідників, які впливають на функціонування системи, зокрема список міст, країн та мовні локалізації інтерфейсу.

Функціонально система поділена на кілька ключових модулів. Перший з них відповідає за взаємодію з клієнтами, забезпечуючи зручний інтерфейс для пошуку турів, їх попереднього перегляду, а також обробки заявок. Впроваджено фільтрацію за критеріями (місце, вартість, період, гід) і модуль відображення детальної інформації про тур.

Другий модуль реалізує персоналізовані рекомендації. Застосовуючи алгоритми штучного інтелекту, система аналізує вподобання, історію заявок і переглядів конкретного користувача, і на цій основі пропонує тури, які найбільше відповідають його інтересам. Цей модуль активно взаємодіє з даними, що зберігаються в особистому кабінеті клієнта.

Третій модуль — менеджмент турів, реалізований для гідів. Він включає можливість створення нових турів із зазначенням усіх необхідних атрибутів (назва, опис, маршрут, ціна, дати, готелі, умови участі), редагування вже наявних, перегляд заявок, які подали клієнти, та скасування туру у разі потреби. Це дозволяє гідам мати повний контроль над організованими ними подорожами.

Окремий модуль доступний керівнику агенції, який відповідає за підтримку актуального стану системних довідників: списку країн, міст та локалізацій. Завдяки цьому забезпечується можливість обслуговування клієнтів із різними мовними налаштуваннями та коректна робота фільтрації турів. Кожен користувач, незалежно від ролі, має можливість самостійно обрати мову інтерфейсу, що є важливим чинником інклюзивності та масштабованості системи.

Основні об’єкти даних, що обробляються системою, включають:

– тури (ідентифікатор, назва, гід, дата, ціна, маршрут, опис, статус);

– заявки (ідентифікатор клієнта, тур, статус, дата створення, дата скасування);

– користувачі (ролі, профіль, історія дій);

– локалізація (словники, доступні мови);

– довідники міст і країн (для структуризації маршрутів та географічної прив’язки).

Уся система працює за принципом клієнт-серверної взаємодії, де запити користувача надсилаються через веб-інтерфейс, обробляються серверною логікою і повертаються у вигляді сформованих сторінок або JSON-

відповідей. Комунікація між модулями відбувається на рівні API, що забезпечує як масштабованість, так і можливість подальшого розвитку функціоналу (наприклад, інтеграцію з глобальними системами бронювання, CRM-модулями або системами аналітики).

Діаграма прецедентів описаної системи представлена на наступному рисунку (див. рис. 2.1).

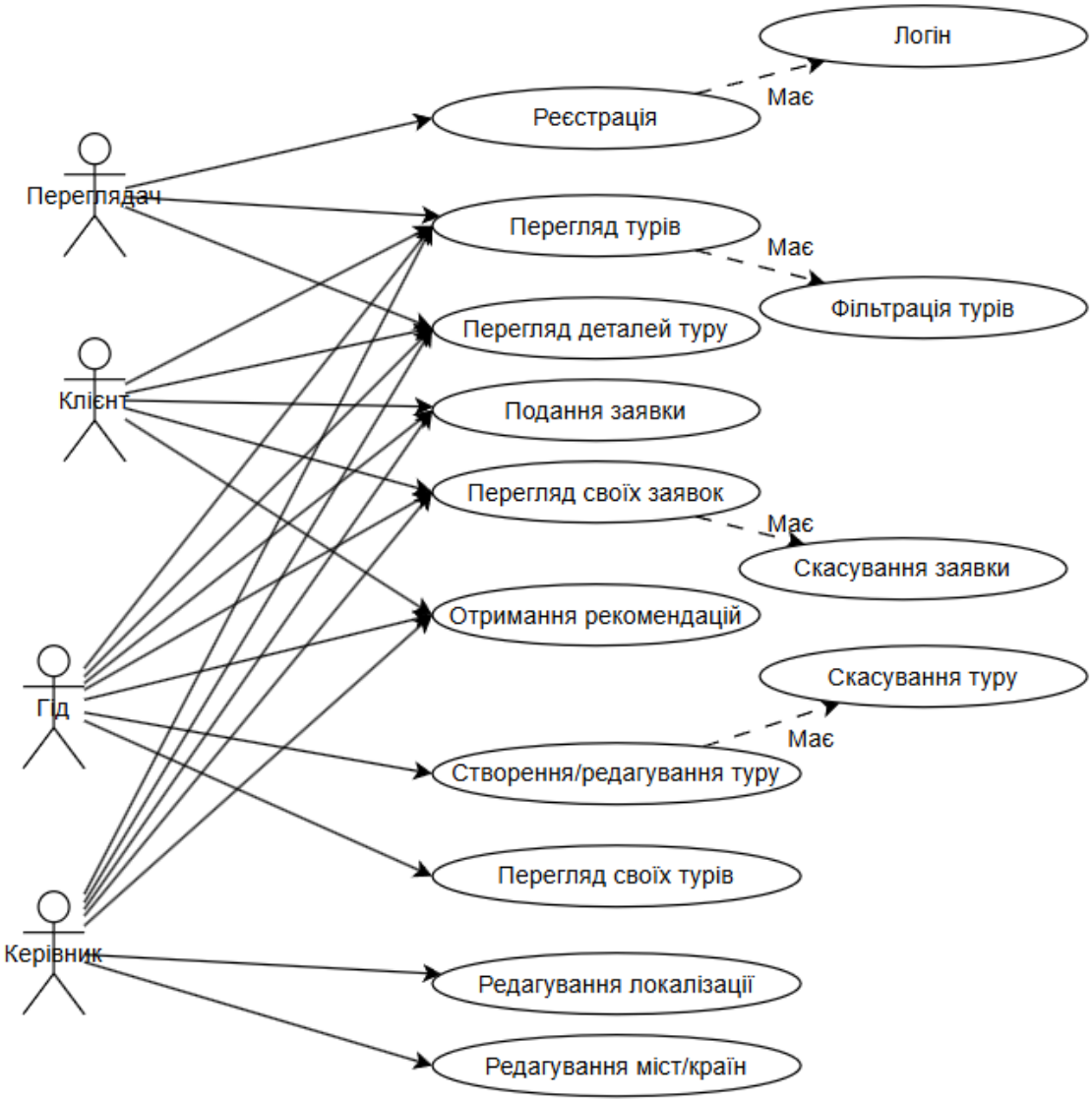


Рисунок 2.1 – Діаграма прецедентів

Джерело: розроблено автором на основі виконаної розробки

Діаграма описана за допомогою інструменту draw.io [20]. Вказана діаграма прецедентів показує 4 актори (Переглядач, клієнт, гід та керівник),

кожен з яких має свої набори активностей, які можуть бути виконані системою. Перелік кожної з головних активностей показана на наступних рисунках (див. рис. 2.2 – 2.9).

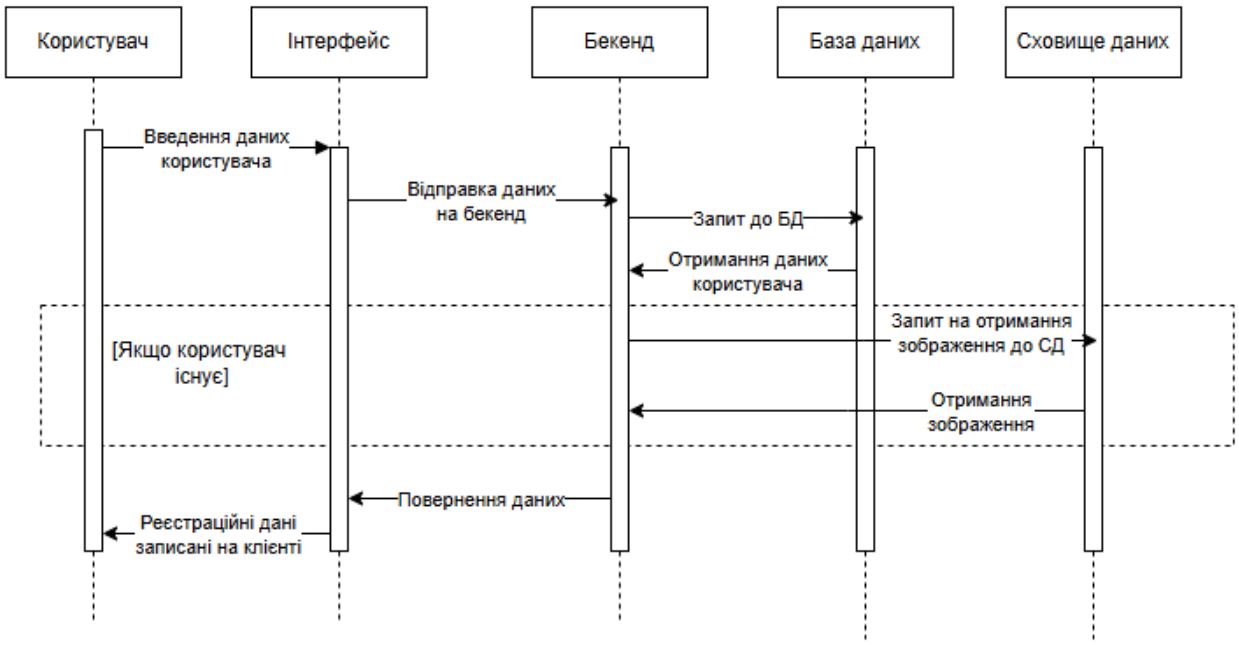


Рисунок 2.2 – Діаграма послідовностей Реєстрації

Джерело: розроблено автором на основі виконаної розробки

На діаграмі на рисунку 2.2 користувач вводить реєстраційні дані через інтерфейс. Інтерфейс передає ці дані на бекенд, який виконує перевірку в базі даних. Якщо користувача ще не існує, створюється новий запис. Якщо додано зображення (наприклад, аватар), воно зберігається у сховищі файлів. Після успішної реєстрації система повертає підтвердження та зберігає токен (JWT) та інші дані акаунта на клієнті, щоб уникнути повторного входу після перезавантаження сторінки.

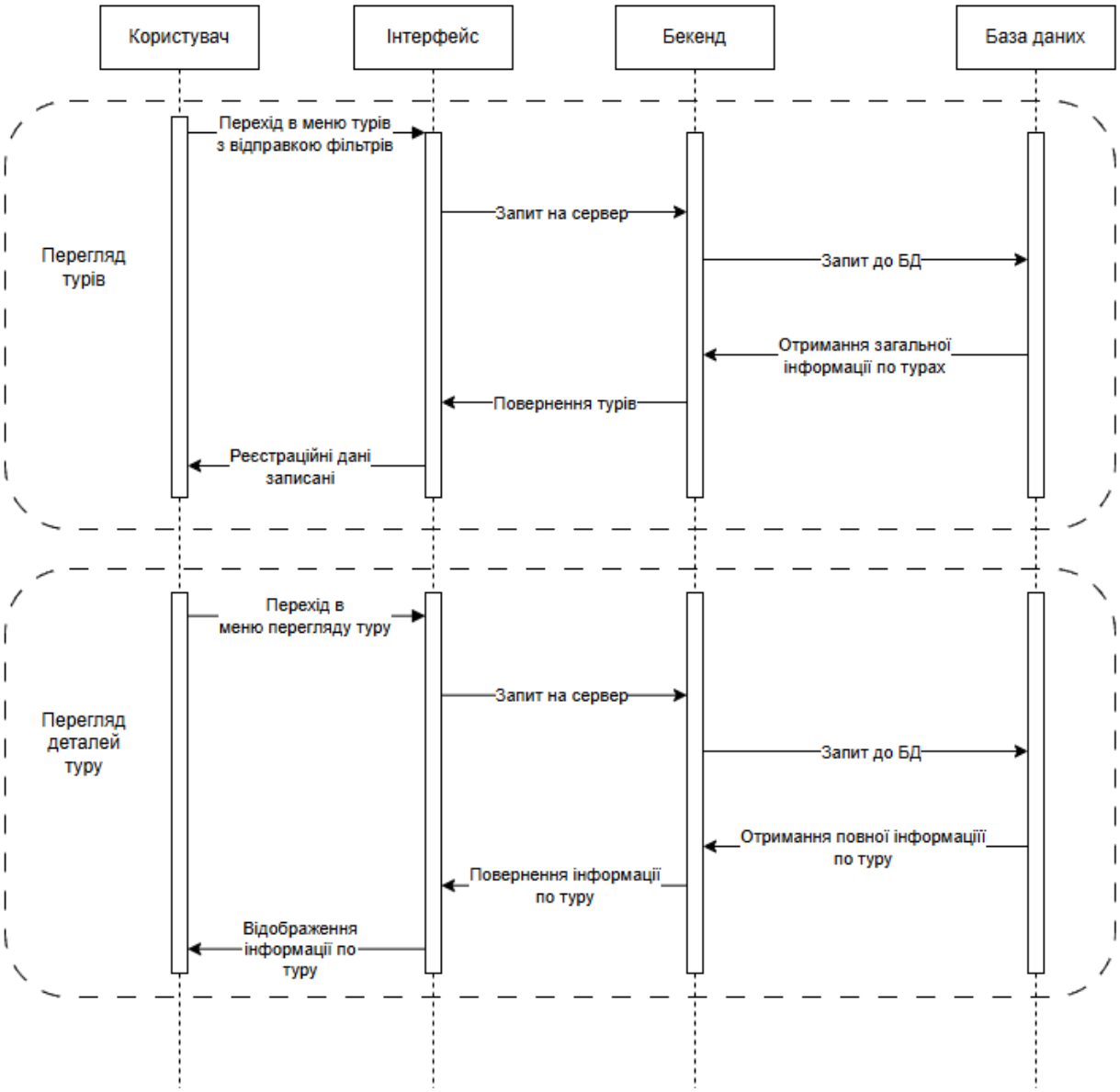


Рисунок 2.3 – Діаграма послідовностей перегляду турів та деталей туру

Джерело: розроблено автором на основі виконаної розробки

На зображеній діаграмі послідовності на рисунку 2.3 представлено два ключові сценарії взаємодії користувача з інформаційною управляючою системою туристичної агенції — перегляд списку турів та перегляд детальної інформації про окремий тур.

Перший сценарій — перегляд турів — починається з ініціативи користувача, який переходить до розділу перегляду турів. У процесі переходу, за потреби, він може застосовувати фільтри (наприклад, за країною, містом,

датами чи ціною). Інтерфейс, у свою чергу, формує запит до серверної частини системи (бекенду) з урахуванням зазначених параметрів фільтрації. Бекенд звертається до бази даних, щоб отримати відповідний список турів, з якого повертається лише загальна інформація — назва, напрямок, короткий опис, ціна та ключові дати. Після цього ці дані передаються назад на інтерфейс і виводяться користувачу для перегляду. Додатково система записує локальні реєстраційні дані, щоб зберегти фільтри або сесію користувача на стороні клієнта.

Другий сценарій — перегляд деталей туру — активується, коли користувач натискає на конкретний тур у списку. Це призводить до переходу в окремий розділ перегляду, де інтерфейс формує новий запит до бекенду, тепер уже із зазначенням унікального ідентифікатора обраного туру. Бекенд передає запит до бази даних, де відбувається вибірка повної інформації про тур: детальний маршрут, умови проживання, наявність харчування, супровід гіда, включені послуги та вартість у розрізі. Отримані дані повертаються назад через сервер на інтерфейс, де виводяться користувачу у зручному вигляді — як сторінка опису туру.

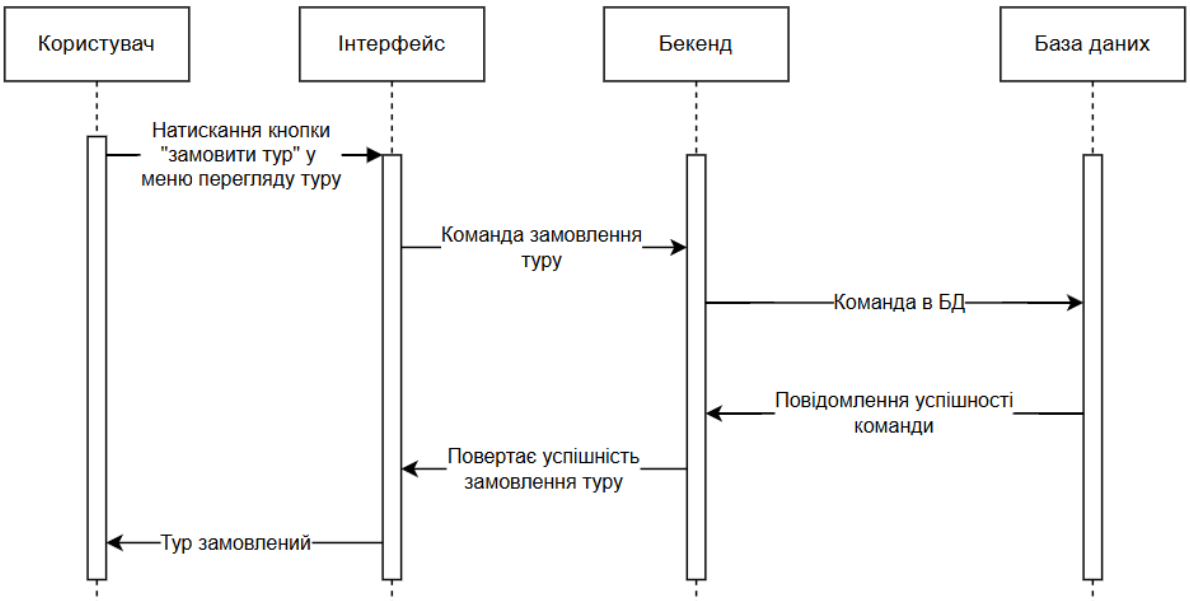


Рисунок 2.4 – Діаграма послідовностей подання заявки

Джерело: розроблено автором на основі виконаної розробки

На діаграмі на рисунку 2.4 зображено сценарій оформлення замовлення на тур, який відбувається після перегляду його детальної інформації. Сценарій починається з того, що користувач натискає кнопку «замовити тур» на сторінці перегляду детальної інформації про обраний тур. Ця подія ініціює дію на рівні інтерфейсу — система фіксує запит користувача та формує відповідну команду для серверної частини (бекенду), яка означає бажання користувача створити замовлення на цей тур.

Інтерфейс передає сформовану команду на бекенд, де вона обробляється: перевіряється наявність користувача, валідність туру та поточний стан замовлень. Далі сервер формує відповідну команду запису до бази даних. На цьому етапі в БД створюється новий запис у таблиці замовлень, який фіксує обраний тур, дату замовлення, користувача, статус та інші параметри.

Після успішного створення запису в базі даних система повертає на бекенд підтвердження про виконання операції. Сервер, у свою чергу, передає цю інформацію назад на інтерфейс. Користувачу повідомляється, що тур успішно замовлено — зазвичай у вигляді повідомлення або оновлення інтерфейсу, наприклад, зміни кнопки на «замовлено» або перенаправлення до історії замовлень.

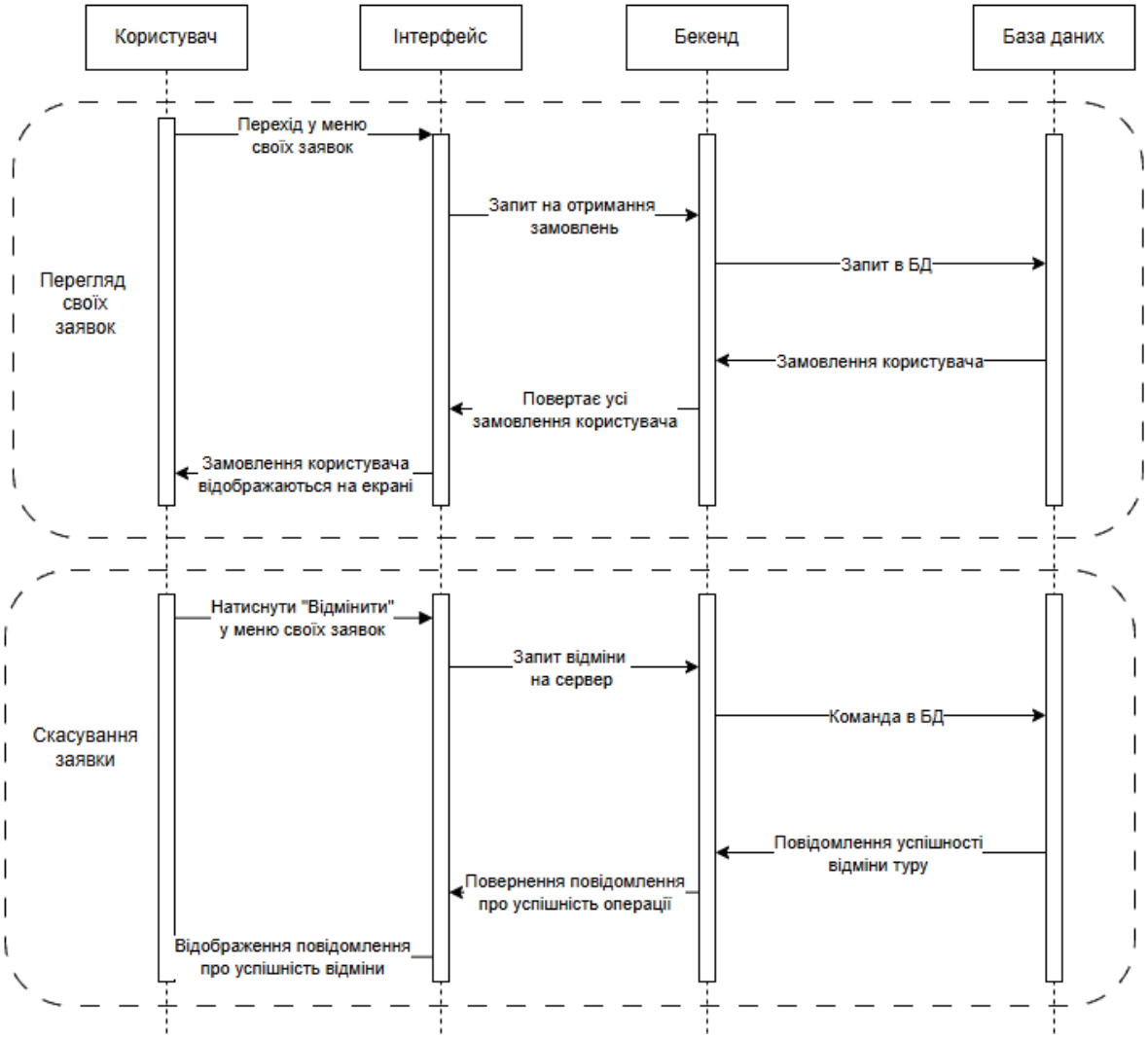


Рисунок 2.5 – Діаграма послідовностей взаємодії з заявками

Джерело: розроблено автором на основі виконаної розробки

Ця діаграма послідовності на рисунку 2.5 описує два пов’язані між собою сценарії взаємодії користувача з інформаційною управляючою системою: перегляд власних замовлень (заявок на тури) та скасування однієї з таких заявок.

У першому сценарії, «Перегляд своїх заявок», користувач ініціює запит, переходячи в розділ перегляду власних замовлень. Інтерфейс генерує відповідний запит до серверної частини (бекенду), яка у свою чергу звертається до бази даних для отримання всіх записів, пов’язаних із поточним користувачем. Бекенд опрацьовує отриману відповідь і повертає список

замовлень на інтерфейс, де дані відображаються на екрані користувача у зрозумілому вигляді. Таким чином, користувач може ознайомитися з переліком своїх активних чи історичних заявок на тури.

Другий сценарій, «Скасування заявки», починається з дії користувача у тому ж самому розділі — він натискає кнопку «Відмінити» біля певного туру, який раніше було замовлено. Інтерфейс фіксує цю дію та передає запит на сервер, де обробляється команда на скасування заявки. Бекенд звертається до бази даних з відповідною командою: змінити статус заявки або повністю видалити її. Після успішного виконання цієї операції база даних надсилає підтвердження назад на сервер, який у свою чергу повідомляє інтерфейс про завершення операції. Інтерфейс оновлює відображення, інформуючи користувача про успішне скасування заявки.

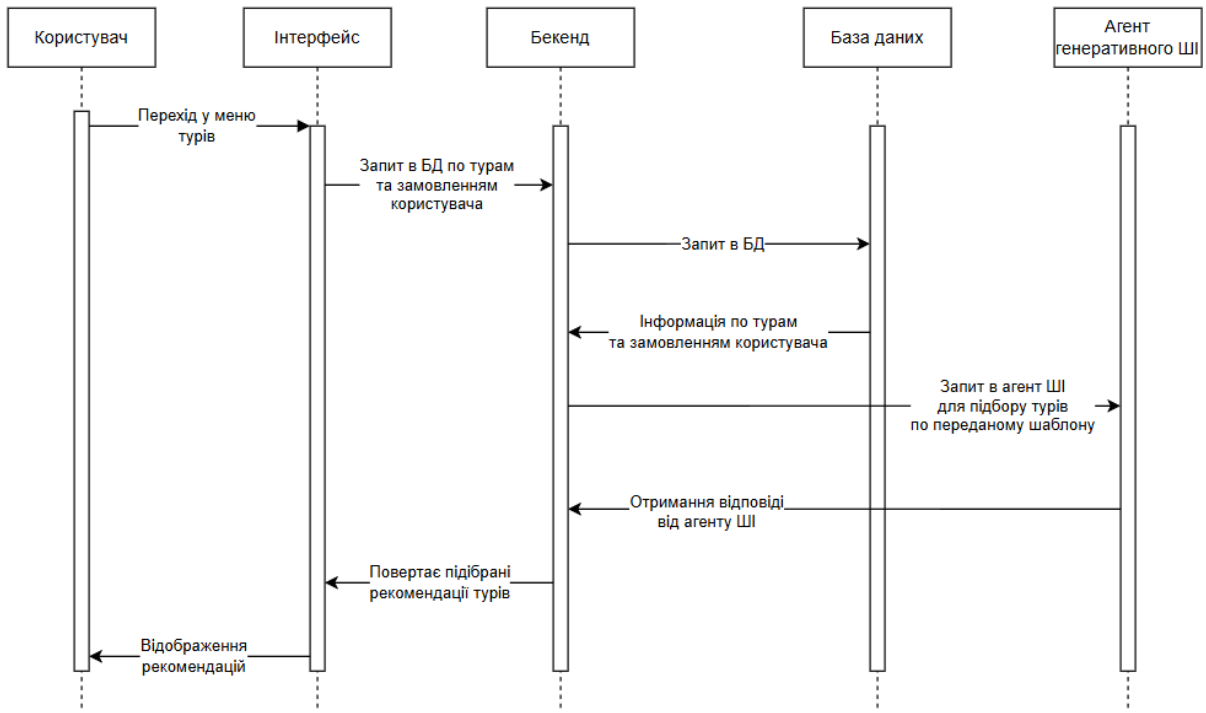


Рисунок 2.6 – Діаграма послідовностей отримання рекомендацій

Джерело: розроблено автором на основі виконаної розробки

Ця діаграма на рисунку 2.6 ілюструє сценарій отримання користувачем персоналізованих рекомендацій турів із використанням агента генеративного штучного інтелекту.

Процес починається з того, що користувач переходить у розділ меню «Тури». Інтерфейс, реагуючи на цю дію, ініціює запит до серверної частини (бекенду), в якому запитується інформація про всі тури, що є доступними, а також історія замовлень поточного користувача. Бекенд, у свою чергу, звертається до бази даних із відповідним SQL-запитом і отримує всі необхідні дані: список доступних турів та ті, що вже були замовлені користувачем раніше.

Наступним кроком є звернення до агента генеративного ШІ. Бекенд формує запит, у якому передає отриману з бази даних інформацію та шаблон для генерації рекомендацій. Цей шаблон може містити інструкції щодо врахування переваг користувача, тематики турів, географії, бюджету, тривалості подорожей або інших релевантних характеристик.

Агент ШІ обробляє запит, аналізуючи як нові тури, так і історію користувача, та повертає згенеровану відповідь — набір персоналізованих туристичних пропозицій. Бекенд отримує цю відповідь, інтерпретує її та передає назад на інтерфейс.

Інтерфейс, отримавши дані з рекомендаціями, виводить їх на екран користувача у вигляді карток, списку або окремого блоку «Рекомендовані тури». Таким чином, користувач має можливість ознайомитися з турами, які з високою ймовірністю його зацікавлять, без необхідності ручного пошуку або застосування фільтрів.

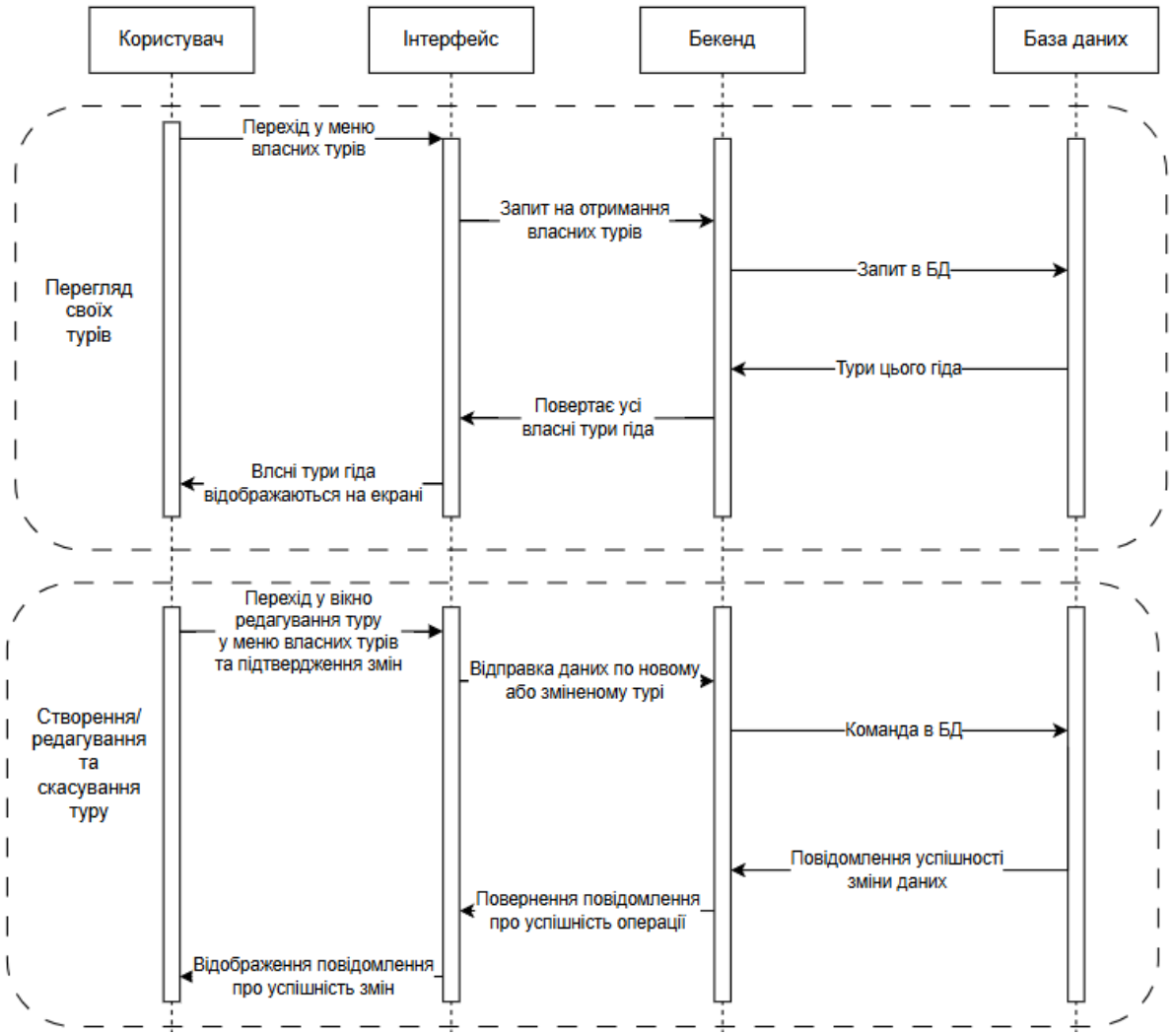


Рисунок 2.7 – Діаграма послідовностей меню гіда

Джерело: розроблено автором на основі виконаної розробки

Діаграма послідовності на рисунку 2.7 описує два взаємопов’язані процеси, характерні для гіда як користувача інформаційної управляючої системи туристичної агенції: перегляд власних турів і редагування конкретного туру. Обидва сценарії дозволяють гідам керувати створеним ними контентом і підтримувати його в актуальному стані.

У першому сценарії — «Перегляд власних турів» — гід ініціює дію, переходячи до розділу перегляду своїх турів. Інтерфейс у відповідь формує запит до серверної частини (бекенду) з ідентифікатором поточного користувача. Бекенд звертається до бази даних із відповідним запитом, щоб отримати всі тури, створені цим гідом. Після отримання результатів ці дані

повертаються на інтерфейс, де відображаються у зручному форматі — у вигляді списку або панелі з картками турів. Це дозволяє гіду швидко ознайомитися з наявним контентом і підготуватись до його оновлення чи перевірки.

Другий сценарій — «Редагування туру» — розпочинається, коли гід обирає певний тур зі списку та переходить до режиму редагування. У цьому вікні він змінює наявну інформацію або створює новий тур, заповнюючи всі необхідні поля: назву, опис, маршрут, тривалість, ціну тощо. Після підтвердження змін інтерфейс надсилає відповідні дані на бекенд. Серверна частина обробляє отриману інформацію та формує команду для бази даних — створити новий запис або оновити наявний. База даних виконує відповідну операцію та надсилає повідомлення про успішність змін. Це повідомлення проходить через бекенд назад на інтерфейс, де користувачу повідомляється про результат дії (наприклад, спливаючим вікном або зміною статусу на кнопці).

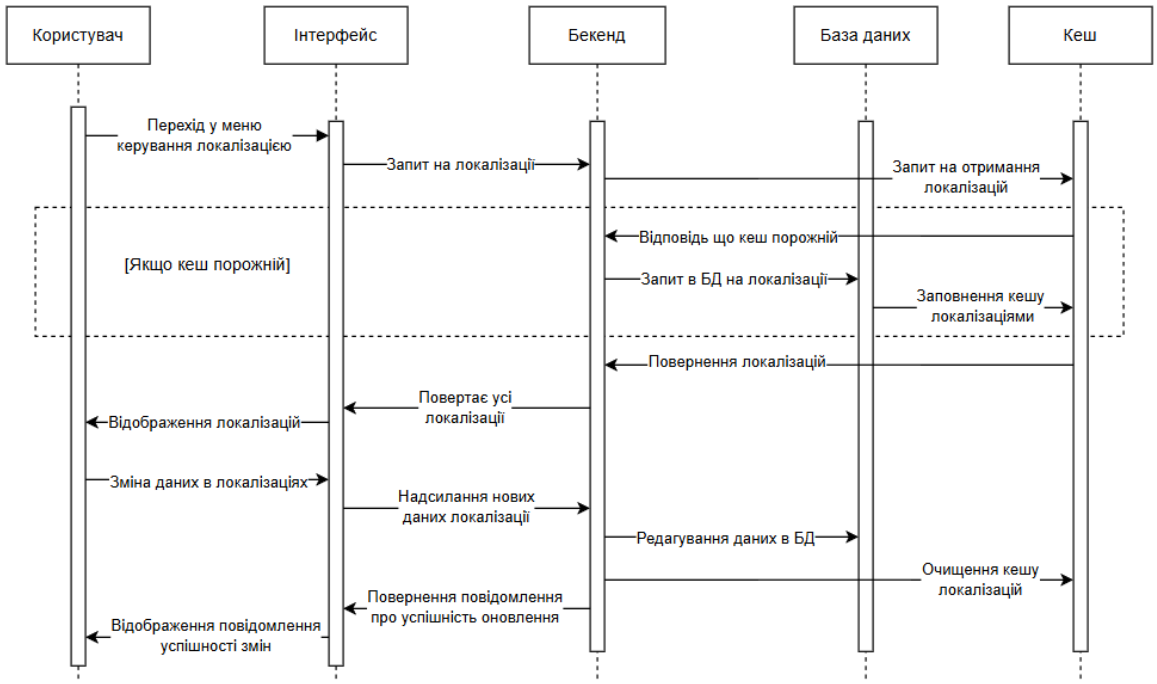


Рисунок 2.8 – Діаграма послідовностей редагування локалізації

Джерело: розроблено автором на основі виконаної розробки

Діаграма на рисунку 2.8 описує сценарій керування локалізаціями в інформаційній управляючій системі туристичної агенції, який є доступним для користувача з роллю керівника. Сценарій охоплює два основні процеси: отримання локалізаційних даних із використанням кешування та оновлення (редагування) локалізаційної інформації.

Сценарій починається з того, що користувач переходить до розділу керування локалізацією. Інтерфейс формує відповідний запит на отримання локалізацій (перекладів текстів інтерфейсу) і передає його до бекенду. Бекенд, у свою чергу, спершу звертається до кешу з метою отримати локалізації без звернення до бази даних.

Якщо кеш порожній (або застарілий), бекенд отримує відповідь, що відповідні дані відсутні, і тоді виконує запит до бази даних на отримання актуальних локалізацій. Отримані дані з БД повертаються до бекенду, де відбувається оновлення кешу — щоби прискорити подальші запити. Після цього повний набір локалізацій повертається через інтерфейс і відображається користувачу.

У другій частині сценарію керівник змінює один або кілька локалізованих текстів і підтверджує редагування. Інтерфейс надсилає змінені дані на бекенд, який, у свою чергу, виконує запит на оновлення даних у базі даних. Після успішного оновлення записів відбувається очищення кешу, щоб при наступному запиті система не використовувала застарілі дані. Далі користувач отримує підтвердження про успішне оновлення, яке відображається на інтерфейсі.

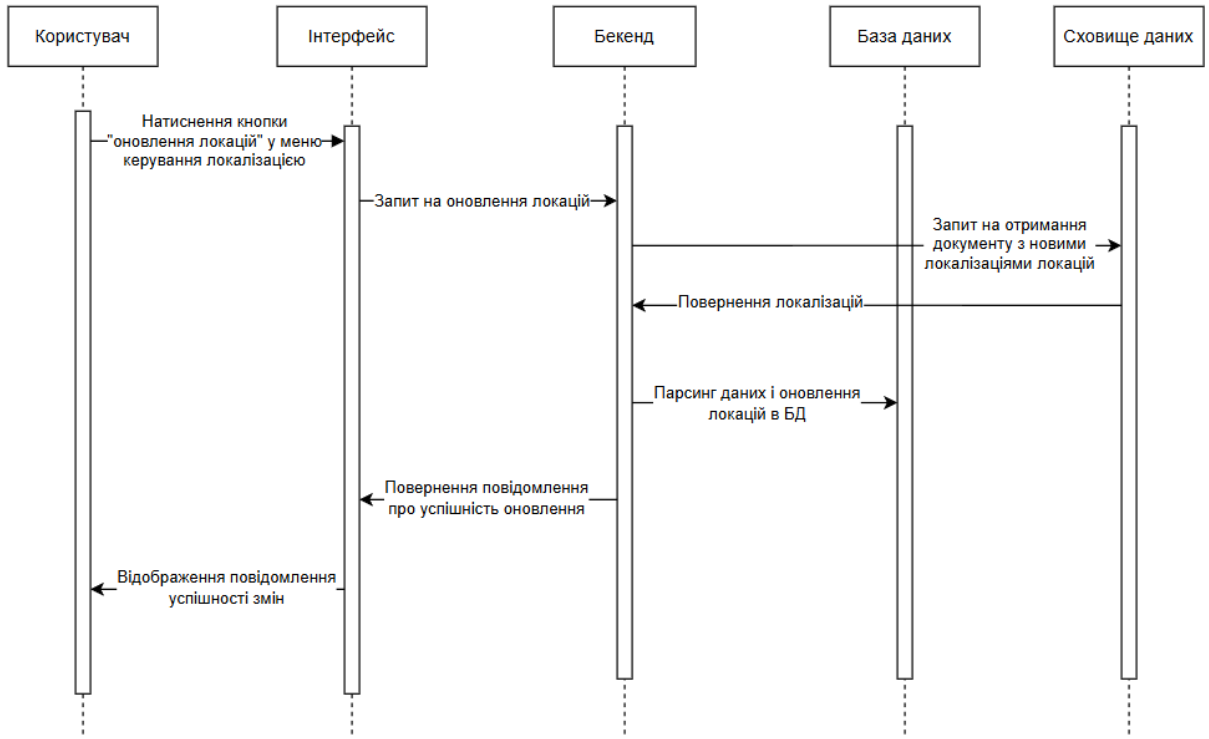


Рисунок 2.9 – Діаграма послідовностей редагування міст/країн

Джерело: розроблено автором на основі виконаної розробки

На діаграмі на рисунку 2.9 зображено сценарій оформлення замовлення на тур, який відбувається після перегляду його детальної інформації.

Сценарій починається з того, що користувач натискає кнопку «замовити тур» на сторінці перегляду детальної інформації про обраний тур. Ця подія ініціює дію на рівні інтерфейсу — система фіксує запит користувача та формує відповідну команду для серверної частини (бекенду), яка означає бажання користувача створити замовлення на цей тур.

Інтерфейс передає сформовану команду на бекенд, де вона обробляється: перевіряється наявність користувача, валідність туру та поточний стан замовлень. Далі сервер формує відповідну команду запису до бази даних. На цьому етапі в БД створюється новий запис у таблиці замовлень, який фіксує обраний тур, дату замовлення, користувача, статус та інші параметри.

Після успішного створення запису в базі даних система повертає на бекенд підтвердження про виконання операції. Сервер, у свою чергу, передає цю інформацію назад на інтерфейс. Користувачу повідомляється, що тур успішно замовлено — зазвичай у вигляді повідомлення або оновлення інтерфейсу, наприклад, зміни кнопки на «замовлено» або перенаправлення до історії замовлень.

Окрім діаграми прецедентів та діаграм послідовностей це існує необхідність описати систему через UML діаграму. Сама UML даіграма сутностей представлена нижче (див. рис. 2.10).

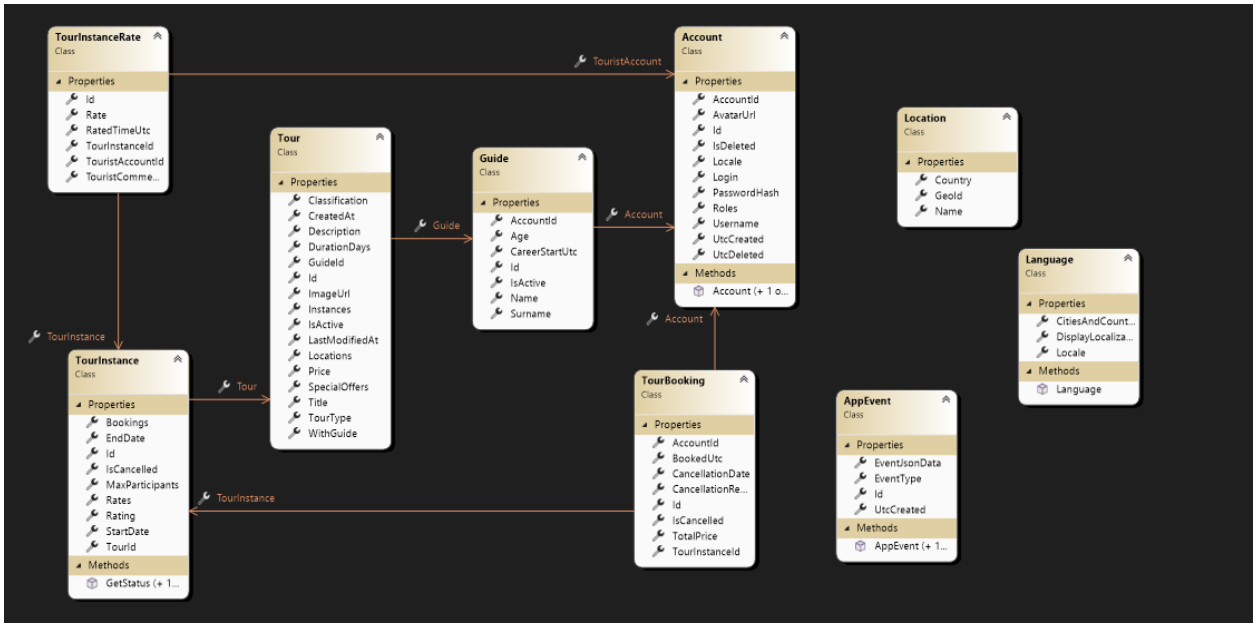


Рисунок 2.10 – UML діаграма системи

Джерело: розроблено автором на основі виконаної розробки

Дана UML-діаграма класів, що зображена на рисунку 2.10, охоплює основні компоненти системи, їх атрибути та зв’язки, забезпечуючи цілісне уявлення про логіку зберігання та обробки даних. Вона охоплює процеси управління користувачами, турами, бронюванням, оцінюванням, локалізацією інтерфейсу та обробкою подій.

Центральне місце в діаграмі посідає клас Account, який відповідає за облікові записи користувачів, незалежно від їхньої ролі в системі. Об’єкт

Account включає такі ключові характеристики, як логін, ім’я користувача, хеш пароля, перелік ролей (наприклад, "Клієнт", "Гід", "Керівник"), мову інтерфейсу (locale), а також прапорці видалення та дату створення облікового запису. Завдяки універсальності цієї сутності, система легко масштабується — для додавання нових типів користувачів достатньо лише розширити ролі.

З класом Account тісно пов’язаний клас Guide, що представляє гідів. Кожен гід має зв’язок із конкретним обліковим записом і володіє власними характеристиками: ім’ям, прізвищем, віком, датою початку кар’єри та статусом активності. Guide асоціюється з класом Tour, адже кожен тур може мати закріпленого за ним гіда.

Клас Tour є центральним з точки зору предметної області. Він моделює загальну інформацію про туристичні маршрути. Його атрибути включають назву, опис, тривалість, вартість, тип, наявність гіда, категорію, дату створення та редагування, посилання на зображення й спеціальні пропозиції. До одного туру може бути прив’язано декілька реалізацій (TourInstance), що дозволяє багаторазове використання одного маршруту у різні дати.

TourInstance описує конкретне проведення туру в заданий час. Він містить інформацію про дати початку та завершення, максимальну кількість учасників, статус скасування, рейтинг, а також перелік пов’язаних бронювань. Завдяки цьому класу реалізується можливість планування та керування турами у часі.

Бронювання турів реалізовано через клас TourBooking. Ця сутність зберігає дані про замовлення користувачів: хто замовив тур, на який тур було зроблено бронювання, коли це відбулося, яка загальна вартість, а також чи було скасовано замовлення. Це дозволяє системі вести облік взаємодії клієнтів із турами та аналізувати історію замовлень.

Користувачі можуть залишати свої оцінки та відгуки про пройдені тури. Для цього передбачено клас TourInstanceRate, що зберігає інформацію про рейтинг, коментар, дату оцінювання, а також ідентифікатори користувача

та туру. Це дозволяє не лише формувати середній рейтинг кожного туру, а й збирати зворотний зв’язок для подальшого аналізу.

Географічна інформація представлена у вигляді класу Location, який містить країну, місто та унікальний географічний ідентифікатор. Цей клас використовується для визначення маршрутів турів, а також може бути локалізованим для різних мов інтерфейсу.

Підтримка багатомовності реалізується через клас Language, який дозволяє зберігати локалізовані назви міст, країн та інтерфейсних компонентів. Це дозволяє користувачам обирати зручну мову системи, що є особливо актуальним для туризму як міжнародної сфери.

Для обліку внутрішніх подій системи застосовується клас AppEvent. Він слугує для зберігання службових подій, таких як зміни в базі даних, взаємодії з користувачами або аналітичні тригери. Події зберігаються у вигляді JSON-структури з позначенням типу події та часу її виникнення.

2.2 Методи та моделі в інформаційних управляючих системах і технологіях

Сучасні задачі у сфері туризму характеризуються високою складністю та неоднорідною структурованістю. Зокрема, багато проблем мають багатокритеріальний характер (наприклад, фільтрація доступних турів за численними параметрами одночасно), можуть бути слабоструктурованими (вибір на основі індивідуальних вподобань клієнта) або частково неструктурованими (генерація рекомендацій за допомогою AI-алгоритмів). При виборі туристичного продукту часто доводиться одночасно враховувати декілька критеріїв – вартість, напрямок, сезонність, рівень готелю, тип дозвілля тощо. Більше того, значну роль відіграють суб'єктивні чинники: клієнти формулюють нечіткі побажання, які важко звести до чітких числових показників. Частина інформації про турпродукт та клієнта може бути неструктурованою (описи, відгуки, фото), тому традиційні формальні моделі не здатні її безпосередньо опрацювати. Відповідно, у процесі планування індивідуальних маршрутів виникає потреба формалізації слабо структурованих показників «профілю» туриста та розробки гнучких систем, здатних максимально врахувати унікальні побажання клієнтів [21]

Класичні кількісні методи (економіко-математичні моделі оптимізації, жорсткі правила вибору тощо), на які традиційно спираються системи підтримки рішень, виявляються недостатніми у згаданих умовах. Вони припускають наявність повної формалізованої інформації та чітких критеріїв ефективності, чого в туризмі досягти важко. Невизначеність параметрів і неможливість точно описати процеси обслуговування туристів призводять до того, що використання строгих математичних моделей часто ускладнене або й неможливе. Наприклад, методи жорсткої багатокритеріальної оптимізації вимагають задання точних ваг важливості для кожного критерію, тоді як реальні вподобання клієнтів можуть бути нечіткими і змінюватись залежно від контексту. У результаті традиційні алгоритми можуть давати неоптимальні,

спрощені рекомендації або зовсім не знаходити рішення. Натомість використання методів штучного інтелекту та м’яких обчислень здатне врахувати суб’єктивні та невизначені фактори: зокрема, дослідження показують, що застосування гібридного нечіткого моделювання дозволяє отримувати більш адекватні результати порівняно з традиційними аналітичними моделями

Окремо варто підкреслити обмеженість rule-based підходів у реальних сценаріях. Фіксовані фільтри та правила (навіть якщо їх багато) допомагають лише в тому разі, коли користувач точно знає свої критерії. На практиці ж клієнти часто висувають нестандартні запити, які неможливо передбачити набором наперед визначених параметрів. Як зазначають у компанії Booking, традиційні моделі та системи правил не здатні вловити тонкощі намірів користувача – наприклад, побажання здійснити подорож із специфічними атрибутами (серця на ліжках, стилізація під Елвіса тощо), для яких не існує окремого фільтра пошуку [22]. Іншими словами, класичний пошук не був розрахований на подібні запити, тому для обробки таких неформальних побажань потрібні інтелектуальні методи нового покоління. Це підкреслює, чому суто кількісних моделей недостатньо і чому виникає потреба залучити штучний інтелект у систему підтримки прийняття рішень.

Отже, для створення ефективної інформаційно-управлінської системи туристичного агентства доцільно поєднати кілька підходів: класичну алгоритмічну обробку структурованих критеріїв і сучасні методи штучного інтелекту для слабоструктурованих та неструктурованих аспектів. У розроблюваній системі запропоновано використання rule-based фільтрації для формалізованих параметрів (таких як межі бюджету, країна чи дата поїздки, наявність візових вимог тощо) – ці чіткі критерії легко реалізувати у вигляді набору правил або фільтрів в базі даних. Водночас для рекомендацій та рішень, що залежать від уподобань клієнта, залучаються методи штучного інтелекту: машинне навчання і нейромережі, здатні навчатися на попередньому досвіді, а також обробка природної мови. Зокрема, інтеграція

OpenAI API (модель GPT) у систему дозволяє інтерпретувати неструктуровані текстові запити користувача та генерувати рекомендації у діалоговому режимі. Це дає змогу «зрозуміти» побажання, сформульовані природною мовою, і зіставити їх з наявними турпродуктами. Подібні AI-компоненти розширюють можливості системи, додаючи до класичних фільтрів ще й інтелектуальний рівень аналізу даних. З часом система може накопичувати інформацію про вподобання клієнтів і вдосконалювати точність порад.

Методи машинного навчання відіграють ключову роль у підтримці прийняття рішень за умов невизначеності. Алгоритми навчання на даних (як-от рекомендаційні системи, побудовані на основі колаборативної фільтрації або нейронних мереж) здатні автоматично виявляти приховані закономірності у великих масивах інформації про клієнтів і тури. Система може навчатися на історичних даних – переглядах турів, бронюваннях, оцінках якості – щоб прогнозувати, які варіанти найбільше сподобаються тому чи іншому користувачеві. Наприклад, використання методів спільної фільтрації дозволяє врахувати зворотний зв’язок від багатьох користувачів для покращення рекомендацій навіть за умови неповних або неточних початкових відомостей. Якщо певний клієнт або сегмент клієнтів демонструє схожі уподобання, модель машинного навчання виявить цей патерн і запропонує відповідні тури (те, що купували чи високо оцінювали «схожі» мандрівники). На відміну від жорстких правил, такі моделі самостійно підлаштовуються під дані, що особливо цінно в умовах динамічного ринку туристичних послуг.

Таким чином, було обрано низку сучасних методів і моделей, що належать до класу інтелектуальних економіко-математичних методів підтримки рішень. Ключові з них перераховані нижче із коротким обґрунтуванням застосування в задачах туризму:

Rule-based фільтрація (експертні системи на основі правил) – підхід, який передбачає застосування наперед визначених правил якщо–то для відбору альтернатив. Використовується для накладення жорстких обмежень і виконання простих логічних перевірок. У туристичній ІС це дає змогу швидко

відсіяти варіанти, що не відповідають базовим вимогам клієнта (наприклад, виключити тури, дорожчі за задану ціну, або такі, що не відповідають обраному напрямку чи датам). Правила зрозумілі для експертів і легко пояснюються користувачеві. Водночас, rule-based методи обмежені тими знаннями, що явно закладені в систему; вони не можуть врахувати нові або нечітко сформульовані критерії, через що їх доповнюють іншими моделями.

Методи машинного навчання та нейронні мережі – це алгоритми, що будуються на даних і здатні навчатися знаходити функціональні залежності між вхідними параметрами і рішенням. У контексті туристичної агенції машинне навчання охоплює, з одного боку, класичні підходи аналізу даних (кластеризація клієнтів, регресійні моделі прогнозування попиту тощо), а з іншого – сучасні нейромережеві моделі (глибокі нейронні мережі, здатні розпізнавати образи, обробляти тексти і робити рекомендації). Нейронні мережі можуть враховувати десятки параметрів туру та профілю туриста одночасно, автоматично визначаючи нелінійні взаємозв’язки між ними. Практичні приклади – побудова рекомендаційних систем, які пропонують користувачу ті напрямки і тури, що найімовірніше йому сподобаються, або прогнозування рейтингів задоволеності подорожжю на основі характеристик поїздки. Сильна сторона ML-методів у тому, що вони поліпшуються з накопиченням даних: що більше система знає про вибори та відгуки клієнтів, то точніше вона зможе передбачити наступний оптимальний варіант. У поєднанні з нечіткою логікою (для інтерпретації вхідних нечітких запитів) та з правилами (для забезпечення обмежень) нейромережі виступають як «двигун» персоналізації і прогнозування у СППР туристичного бізнесу.

Отже, чому саме ці методи були обрані? Враховуючи багатокритеріальність і часткову невизначеність туристичних задач, класичні математичні моделі потребують розширення інструментарію. Застосування штучного інтелекту (нечітких моделей, машинного навчання, еволюційних алгоритмів тощо) обґрунтоване тим, що вони дозволяють будувати гнучкі системи підтримки прийняття рішень. Така система зможе адаптивно

підлаштовуватися під індивідуальні вподобання клієнтів, враховувати нечіткі критерії і знаходити рішення в умовах неповної інформації – тобто виконувати те, чого не забезпечують класичні кількісні методи [23]. Інтеграція різнопланових підходів (rule-based та AI-моделей) дасть синергетичний ефект: забезпечить і базову надійність та прозорість правил для обов'язкових умов, і інтелектуальну гнучкість для тонкого налаштування рекомендацій. У підсумку впровадження описаних економіко-математичних методів у інформаційну систему туристичного агентства підвищить обґрунтованість і якість управлінських рішень, допоможе клієнтам швидше знаходити оптимальні тури, а бізнесу – залишатися конкурентоспроможним за рахунок персоналізації та ефективного аналізу даних.

РОЗДІЛ 3 РОЗРОБЛЕННЯ ПРОЄКТНИХ РІШЕНЬ ТА ЇХ РЕАЛІЗАЦІЯ

3.1. Проєктування бази даних та/або сховища даних для інформаційної управляючої системи

Ефективне функціонування інформаційної управляючої системи туристичної агенції значною мірою залежить від коректно спроектованого інформаційного забезпечення — сукупності структур зберігання, організації доступу та обробки даних, які забезпечують повну, актуальну та структуровану інформацію для підтримки користувачів і внутрішніх бізнес-процесів. У межах даної системи передбачено використання декількох типів сховищ, кожен з яких виконує специфічну роль відповідно до характеру оброблюваних даних і функціональних вимог.

З огляду на це, інформаційна модель розробленої ІУС побудована на комбінації трьох основних складових:

Реляційна база даних (PostgreSQL) — використовується для зберігання структурованої інформації, пов’язаної з користувачами, турами, заявками, локалізаціями та іншими сутностями системи. З огляду на характер даних, які підлягають точному зберіганню, пошуку та зв’язкам між таблицями, вибір реляційної моделі є оптимальним. PostgreSQL забезпечує відповідність ACID-вимогам, підтримку транзакцій, індексів і складних запитів, що необхідно для коректної роботи основних модулів системи.

Зовнішнє сховище зображень (CDN) — оскільки система передбачає зберігання фотографій турів, а також потенційно інших медіафайлів (наприклад, профілі гідів), для цих цілей було використано окреме сховище, винесене за межі реляційної БД. Це дозволяє розвантажити основну базу даних, оптимізувати швидкість завантаження сторінок та забезпечити масштабованість. Файли зберігаються в хмарному середовищі з прив’язкою до унікальних ідентифікаторів, які зберігаються в основній базі.

Кеш-сховище на основі NoSQL бази Redis — реалізовано для зберігання проміжних сесійних даних, зокрема токенів авторизації, тимчасової

інформації про авторизованих користувачів, та інших об’єктів, доступ до яких має бути надшвидким і не потребує складних транзакцій. Використання Redis дозволяє зменшити навантаження на основну БД та забезпечити високу продуктивність критичних ділянок системи, пов’язаних із автентифікацією та авторизацією.

Таким чином, у системі реалізовано комбіновану модель інформаційного зберігання, яка враховує типи, обсяги та структуру даних. Застосування як реляційної, так і нереляційної БД, а також зовнішнього файлового сховища дозволяє забезпечити необхідний рівень гнучкості, масштабованості, продуктивності та надійності.

3.1.1. Проєктування бази даних для інформаційної управляючої системи

На етапі проєктування інфологічної моделі бази даних визначаються основні сутності системи, їх атрибути (поля) та взаємозв'язки між ними. Для системи туристичних турів створена така модель бази даних, яка продемонстрована ER-діаграмою, що зображена на рисунку (див. рис. 3.1).

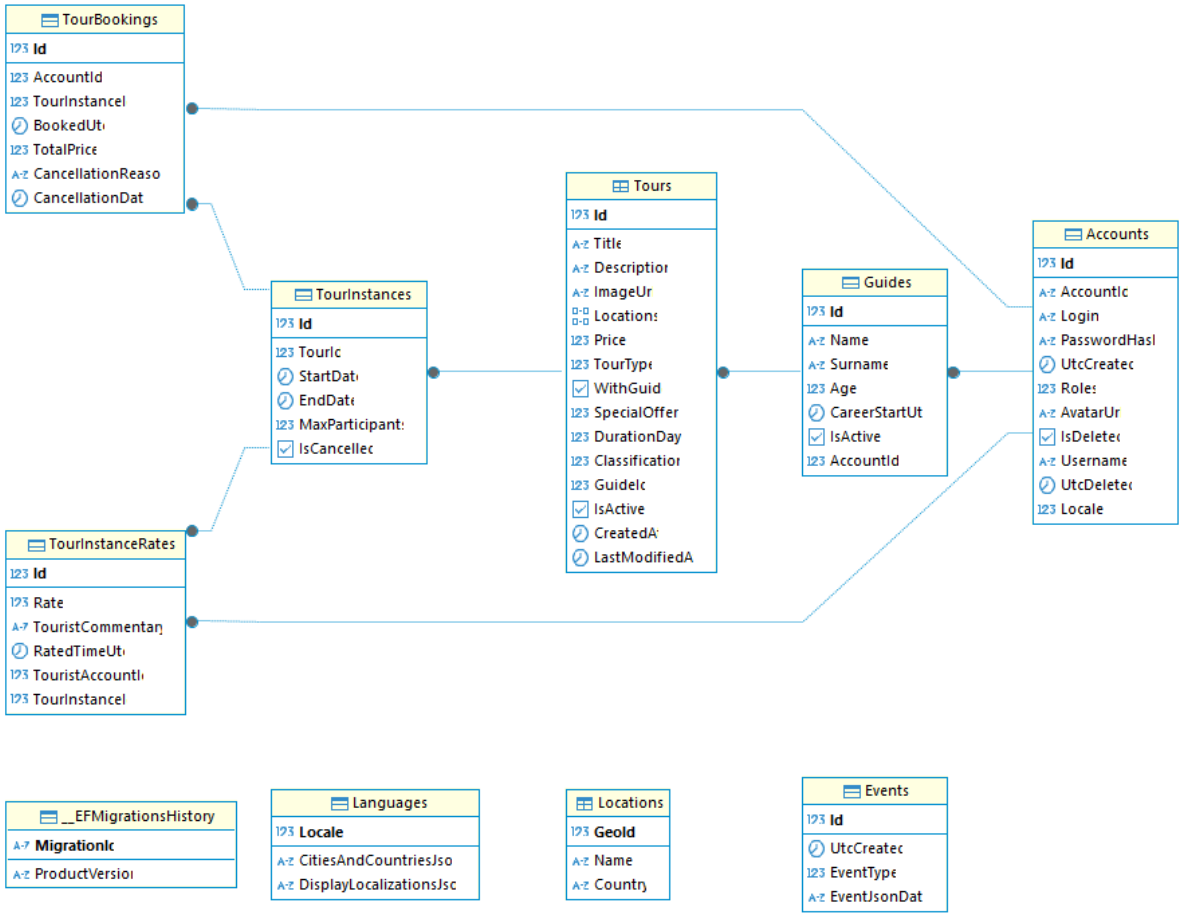


Рис. 3.1 – ER-діаграма логічної моделі бази даних

Джерело: розроблено автором на основі виконаної розробки

Ця діаграма на рисунку 3.1 відображає основні сутності системи туристичних турів та зв'язки між ними. На діаграмі графічно показані таблиці (сутності) та зв'язки між ними у вигляді ліній зі стрілками. Зокрема, позначено первинні ключі (PK) для кожної сутності та зовнішні ключі (FK), які вказують

на зв’язки: наприклад, поле GuideId у таблиці Tours посилається на таблицю Guides, поля TourId у TourInstances, AccountId у TourBookings тощо. Така діаграма наочно демонструє структуру логічної моделі та взаємозв’язки, забезпечуючи розуміння схем даних перед реалізацією.

Далі наведено структуру кожної основної таблиці бази даних у вигляді списку полів із зазначенням типів даних та призначенням:

Таблиця 3.1 – Accounts (облікові записи користувачів)

Назва поля	**Тип даних**	**Опис**
Id	INT	Первинний ключ, унікальний ідентифікатор облікового запису.
AccountId	INT	Зовнішній ключ (самопосилання) на інший запис Accounts (ідентифікатор батьківського акаунта, якщо застосовно).
Login	VARCHAR	Логін користувача для входу в систему (наприклад, email-адреса).
PasswordHash	VARCHAR	Хеш пароля користувача (для безпечного зберігання пароля).
UtcCreated	TIMESTAMP	Дата та час створення облікового запису (UTC).
Roles	VARCHAR	Ролі користувача в системі (наприклад, “Tourist”, “Guide”, “Admin”).
AvatarUrl	VARCHAR	URL-адреса аватару (зображення профілю) користувача.
IsDeleted	BOOLEAN	Ознака того, що обліковий запис видалено (логічне видалення).
Username	VARCHAR	Ім’я користувача або відображуване ім’я (нікнейм).
UtcDeleted	TIMESTAMP	Дата та час видалення облікового запису (якщо запис було видалено).

Назва поля	Тип даних	Опис
Locale	VARCHAR	Код мови/локалі користувача (наприклад, "en-US" або "uk-UA"), відповідає запису в таблиці Languages.

Джерело: сформовано автором на основі виконаної розробки

Accounts – таблиця облікових записів користувачів системи (туристів, гідів, адміністраторів). Містить інформацію про акаунти: логіни, паролі, ролі, налаштування локалі тощо.

Таблиця 3.2 – Guides (гіди, екскурсоводи)

Назва поля	Тип даних	Опис
Id	INT	Первинний ключ, унікальний ідентифікатор гіда.
Name	VARCHAR	Ім’я гіда.
Surname	VARCHAR	Прізвище гіда.
Age	INT	Вік гіда.
CareerStartUtc	TIMESTAMP	Дата та час початку кар’єри гіда (UTC).
IsActive	BOOLEAN	Статус активності гіда (чи наразі працює, доступний для проведення турів).
AccountId	INT	Зовнішній ключ на таблицю Accounts, що прив’язує гіда до облікового запису користувача.

Джерело: сформовано автором на основі виконаної розробки

Guides – таблиця гідів (екскурсоводів), яка зберігає дані про туристичних гідів (ім’я, прізвище, вік, досвід роботи тощо) та посилається на відповідний обліковий запис у таблиці Accounts (кожен гід прив’язаний до свого аккаунта).

Таблиця 3.3 – Tours (тури, туристичні маршрути)

Назва поля	**Тип даних**	**Опис**
Id	INT	Первинний ключ, унікальний ідентифікатор туру.
Title	VARCHAR	Назва туру.
Description	TEXT	Опис туру (детальна інформація про маршрут, програму тощо).
ImageUrl	VARCHAR	URL-адреса зображення, що ілюструє тур.
Locations	TEXT/JSON	Перелік локацій, які охоплює тур (наприклад, міста та країни маршруту, може зберігатися у форматі JSON або як розділений список).
Price	NUMERIC	Ціна туру (вартість, наприклад, за одного учасника або за весь тур).
TourType	VARCHAR	Тип туру (категорія або тематика, наприклад, екскурсійний, пригодницький тощо).
WithGuide	BOOLEAN	Прапорець, що вказує, чи проводиться тур з гідом (true – з гідом, false – самостійний тур).
SpecialOffer	BOOLEAN	Прапорець спеціальної пропозиції: чи є тур акційним/пільговим (true – тур пропонується як спеціальна пропозиція).
DurationDays	INT	Тривалість туру в днях.
Classification	VARCHAR	Класифікація туру (додаткова категоризація або рівень складності/комфорту туру).

Назва поля	Тип даних	Опис
GuideId	INT	Зовнішній ключ на таблицю Guides, що вказує гіда, відповідального за проведення туру.
IsActive	BOOLEAN	Ознака доступності туру (чи активний тур для бронювання).
CreatedAt	TIMESTAMP	Дата та час створення запису туру.
LastModifiedAt	TIMESTAMP	Дата та час останнього внесення змін до туру.

Джерело: сформовано автором на основі виконаної розробки

Tours – таблиця туристичних турів. Вона містить основну інформацію про кожен тур: назву, опис, зображення, список локацій маршруту, ціну, тип і класифікацію туру, тривалість, чи супроводжується тур гідом (поле WithGuide), чи є він спеціальною пропозицією (пільговим/акційним туром), а також посилання на гіда, відповідального за тур (зовнішній ключ на Guides).

Таблиця 3.4 – TourInstances (екземпляри турів / заплановані проведення туру)

Назва поля	Тип даних	Опис
Id	INT	Первинний ключ, унікальний ідентифікатор екземпляра туру.
TourId	INT	Зовнішній ключ на таблицю Tours – вказує, до якого туру відноситься цей запланований екземпляр.
StartDate	TIMESTAMP	Дата та час початку туру (початок конкретного заїзду, UTC).
EndDate	TIMESTAMP	Дата та час завершення туру (закінчення заїзду, UTC).

Назва поля	Тип даних	Опис
MaxParticipants	INT	Максимальна кількість учасників, передбачена для цього екземпляра туру.
IsCancelled	BOOLEAN	Статус скасування: чи скасовано даний екземпляр туру (true – скасовано, false – актуальний).

Джерело: сформовано автором на основі виконаної розробки

TourInstances – таблиця екземплярів турів (конкретних проведень туру). Кожен запис відповідає запланованому туру на певні дати і містить посилання на шаблон туру (зовнішній ключ на Tours), дату початку та завершення, максимальну кількість учасників і статус скасування (чи скасовано даний заїзд).

Таблиця 3.5 – TourBookings (бронювання турів)

Назва поля	Тип даних	Опис
Id	INT	Первинний ключ, унікальний ідентифікатор бронювання.
AccountId	INT	Зовнішній ключ на таблицю Accounts – посилається на акаунт користувача, який здійснив бронювання.
TourInstanceId	INT	Зовнішній ключ на таблицю TourInstances – вказує, яке саме проведення туру заброньоване.
BookedUtc	TIMESTAMP	Дата та час створення бронювання (коли користувач забронював тур, UTC).
TotalPrice	NUMERIC	Загальна вартість бронювання (розрахована ціна для цього

Назва поля	Тип даних	Опис
		замовлення, з урахуванням кількості осіб тощо).
IsCancelled	BOOLEAN	Ознака того, чи бронювання скасовано (true – бронювання скасоване).
CancellationReason	VARCHAR	Причина скасування бронювання (текстове пояснення, може бути NULL, якщо не скасовано).
CancellationDate	TIMESTAMP	Дата та час скасування бронювання (якщо було скасовано).

Джерело: сформовано автором на основі виконаної розробки

TourBookings – таблиця бронювань турів користувачами. Містить записи про бронювання: хто (Account) і який екземпляр туру (TourInstance) забронював, дату та час бронювання, загальну вартість, а також статус і причину скасування (якщо бронювання було скасовано).

Таблиця 3.6 – TourInstanceRates (рейтинги та відгуки до турів)

Назва поля	Тип даних	Опис
Id	INT	Первинний ключ, унікальний ідентифікатор відгуку/оцінки.
TourInstanceId	INT	Зовнішній ключ на таблицю TourInstances – визначає, до якого екземпляра туру належить цей відгук.
TouristAccountId	INT	Зовнішній ключ на таблицю Accounts – акаунт туриста, який залишив відгук.
Rate	INT	Оцінка (рейтинг), виставлена туристом (числове значення, напр. за шкалою 1–5).
TouristComment	TEXT	Текстовий коментар від туриста про тур (відгук).

Назва поля	Тип даних	Опис
RatedTimeUtc	TIMESTAMP	Дата та час, коли відгук був залишений (UTC).

Джерело: сформовано автором на основі виконаної розробки

TourInstanceRates – таблиця відгуків та рейтингів туристів. Вміщує оцінки (рейтинги) і коментарі, які залишають користувачі-туристи після участі в турі. Кожен відгук містить оцінку, текстовий коментар, посилання на відповідний екземпляр туру (TourInstance) і посилання на аккаунт туриста, який залишив відгук (Account).

Таблиця 3.7 – Locations (локації: міста/регіони)

Назва поля	Тип даних	Опис
GeoId	INT	Первинний ключ, унікальний код або ідентифікатор локації.
Name	VARCHAR	Назва локації (міста або місця).
Country	VARCHAR	Назва країни, до якої належить локація.

Джерело: сформовано автором на основі виконаної розробки

Locations – таблиця локацій (географічних місць: міст, країн), що використовується для зберігання довідкової інформації про місця проведення турів. В полі туру Locations може зберігатися перелік ідентифікаторів або назв локацій маршруту, що відсилається до записів у цій таблиці.

Таблиця 3.8 – Languages (мови/локалі)

Назва поля	Тип даних	Опис
Locale	VARCHAR	Первинний ключ, код локалі (мови) у форматі стандартизованого коду (напр. "en-US", "uk-UA").

Назва поля	Тип даних	Опис
CitiesAndCountriesJson	JSON	JSON-дані, що містять локалізовані назви міст та країн для даної мови (наприклад, словник назв локацій для відображення користувачу цією мовою).
DisplayLocalizationsJson	JSON	JSON-дані з іншими локалізованими текстовими ресурсами інтерфейсу (наприклад, переклади заголовків, повідомлень системи тощо для цієї мови).

Джерело: сформовано автором на основі виконаної розробки

Languages – таблиця мов/локалей системи. Містить інформацію про підтримувані мови інтерфейсу (код локалі, назву мови) та, можливо, локалізовані дані (наприклад, назви міст і країн для кожної локалі у форматі JSON, текстові ресурси інтерфейсу тощо). Кожен обліковий запис користувача в таблиці Accounts має код локалі, що відповідає одному із записів у таблиці Languages.

Таблиця 3.9 – Events (журнал подій системи):

Назва поля	Тип даних	Опис
Id	INT	Первинний ключ, унікальний ідентифікатор запису події.
EventType	VARCHAR	Тип події (назва або код, що ідентифікує вид події, напр. "BookingCreated", "TourCancelled").
EventJsonData	JSON	Детальні дані події у форматі JSON (містять динамічну інформацію, пов’язану з подією

Назва поля	Тип даних	Опис
		– наприклад, ідентифікатори об’єктів, значення полів до/після зміни тощо).
UtcCreated	TIMESTAMP	Дата та час фіксації події (UTC).

Джерело: сформовано автором на основі виконаної розробки

Events – таблиця системних подій (журнал подій). Зберігає записи про різноманітні події в системі (наприклад, створення бронювання, скасування туру, авторизація тощо) із зазначенням часу події, типу та деталей події у вигляді JSON. Це забезпечує аудит змін і дій користувачів та адміністраторів.

У спроєктованій базі даних всі основні таблиці пов’язані реляційними зв’язками, що забезпечує цілісність і узгодженість даних. Нижче перераховано ключові зв’язки між таблицями (із зазначенням кратності та зовнішніх ключів):

- **Accounts – Guides:** зв’язок 1:1. Кожен гід має свій обліковий запис (у таблиці Guides є поле AccountId – зовнішній ключ на Accounts(Id)). Таким чином, одному запису в Guides відповідає рівно один запис в Accounts, і навпаки – кожному акаунту гіда відповідає не більше одного запису Guides;
- **Guides – Tours:** зв’язок 1:N. Один гід може проводити багато різних турів, але кожен запис туру прив’язаний лише до одного гіда (в Tours зберігається GuideId – зовнішній ключ на Guides);
- **Tours – TourInstances:** зв’язок 1:N. Кожен тур (як шаблон) може мати кілька запланованих екземплярів (рейсів/заїздів), але кожен екземпляр TourInstance пов’язаний лише з одним туром (поле TourId у TourInstances є зовнішнім ключем на Tours);
- **TourInstances – TourBookings:** зв’язок 1:N. Один екземпляр туру може бути заброньований багатьма різними користувачами (кілька записів у TourBookings для одного TourInstance), при цьому кожне окреме бронювання стосується тільки одного екземпляра туру. У таблиці TourBookings зовнішній ключ TourInstanceId посилається на TourInstances;

- **Accounts – TourBookings:** зв’язок 1:N. Один користувач (акаунт) може зробити багато бронювань різних турів, але кожен запис бронювання має лише одного власника-аккаунта. У TourBookings передбачено поле AccountId – зовнішній ключ на Accounts;
- **TourInstances – TourInstanceRates:** зв’язок 1:N. Кожен екземпляр туру може мати кілька відгуків/оцінок від різних туристів, але кожен конкретний відгук стосується лише одного екземпляра. У таблиці TourInstanceRates поле TourInstanceId є зовнішнім ключем, що посилається на TourInstances;
- **Accounts – TourInstanceRates:** зв’язок 1:N. Один користувач-турист може залишити кілька відгуків (до різних турів), але кожен запис відгуку прив’язаний до конкретного аккаунта автора. Поле TouristAccountId у TourInstanceRates є зовнішнім ключем на Accounts;
- **Tours – Locations:** зв’язок, логічно, M:N. Один тур може охоплювати декілька локацій (міст/країн), і одна локація може бути задіяна у багатьох турах. У даній моделі для спрощення цієї багатозначної залежності використовується поле Locations у Tours (наприклад, у вигляді списку або JSON), що містить перелік ідентифікаторів локацій маршруту. Кожне значення відповідає запису в таблиці Locations. Таким чином досягається логічний зв’язок між туром і його локаціями;
- **Accounts – Languages:** зв’язок N:1. Багато аккаунтів можуть використовувати одну й ту саму мову інтерфейсу (Locale), вказану у полі Locale таблиці Accounts. Це поле співвідноситься зі значенням первинного ключа у таблиці Languages. Таким чином користувачі мають свої мовні налаштування, а система звертається до таблиці Languages для відображення відповідних локалізованих даних;
- **Accounts – Accounts:** самозв’язок 1:N (необов’язковий). Таблиця Accounts містить поле AccountId, яке може слугувати для зв’язку запису з іншим записом в цій же таблиці. Це дозволяє, наприклад, реалізувати ієрархію користувачів або прив’язку акаунта до профілю (хоча в даній системі основна

інформація про профіль гіда винесена в окрему таблицю Guides). Якщо значення AccountId задано, то воно вказує на батьківський або пов’язаний акаунт;

- **Events:** не має прямих зовнішніх ключів на інші таблиці, проте у полі з JSON-даними можуть фіксуватися ідентифікатори сутностей, яких стосується подія. Наприклад, подія типу “BookingCreated” може містити ідентифікатор створеного бронювання, або “TourCancelled” – містити ID туру. Таблиця Events слугує для аудиту і аналітики, зберігаючи зв’язані з системою події у гнучкому форматі.

3.1.2. Проєктування сховища даних для інформаційної управляючої системи

У рамках даної інформаційної управляючої системи передбачено використання зовнішнього сховища даних (СД) для зберігання мультимедійного контенту, зокрема зображень. Враховуючи вимоги до зберігання, масштабованості та ефективного доступу до файлів, було обрано хмарний сервіс Cloudinary як оптимальне рішення для реалізації функціоналу сховища. Вигляд сервісу Cloudinary показаний нижче (див. рис. 3.2).

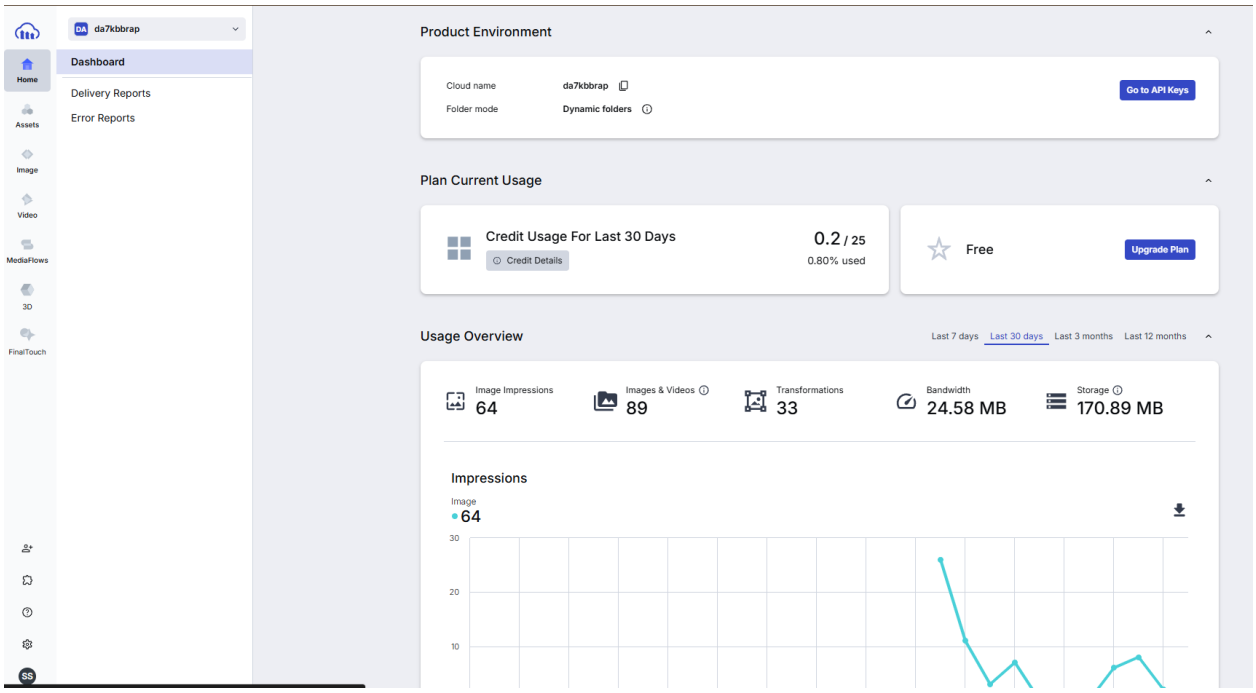


Рис. 3.2 – Вигляд сервісу Cloudinary

Джерело: зроблено автором на основі виконаної розробки

Метою впровадження СД є зберігання:

- зображень турів (обкладинки турів, ілюстрації);
- аватарів користувачів.

Таким чином, використання Cloudinary дозволяє розвантажити основну систему зберігання та базу даних від об’ємного статичного контенту, забезпечивши ефективне керування медіафайлами.

Завантаження даних до сховища здійснюється безпосередньо з бекенд-частини системи за допомогою офіційного SDK Cloudinary для платформи .NET, розповсюджуваного як NuGet-пакет cloudinary-dotnet. Взаємодія реалізується через REST API сервісу, що дає змогу автоматизовано завантажувати, оновлювати та видаляти ресурси з хмарного сховища у відповідь на дії користувача.

Після успішного завантаження зображення, система зберігає пряме посилання на ресурс у реляційній базі даних (PostgreSQL). Зокрема, URL-адреса зображення зберігається в полях:

- ImageUrl — у таблиці Tours, для візуального представлення туру;
- AvatarUrl — у таблиці Accounts, для відображення профілю користувача.

Посилання мають публічний доступ для перегляду через CDN Cloudinary, проте завантаження й обробка дозволені лише з автентифікацією через ключ доступу, що забезпечує базовий контроль безпеки.

Cloudinary автоматично надає використання мережі доставки контенту (CDN), що значно підвищує швидкість доступу до зображень у будь-якій частині світу, мінімізуючи затримки в їх завантаженні. Таким чином, CDN-сервіс забезпечує високу продуктивність фронтенду при візуалізації інтерфейсів користувача.

У поточній реалізації не застосовуються спеціальні стилі обробки зображень або трансформації (типу crop, resize чи watermark), оскільки відображення відбувається безпосередньо з URL оригінального файлу. Проте, в майбутньому Cloudinary дозволяє легко масштабувати функціональність — наприклад, шляхом генерації декількох версій зображень (мініатюри, високої роздільності тощо).

Сховище Cloudinary забезпечує:

- централізоване зберігання зображень;
- надійність та розширюваність у хмарному середовищі;
- інтеграцію з .NET середовищем;

- захищене завантаження ресурсів через SDK із використанням ключа API;
- миттєвий доступ до зображень через оптимізовану CDN-мережу.

Архітектурно Cloudinary не інтегрується у вигляді БД до схеми ER-моделі системи, проте виступає як логічно пов’язаний компонент сховища даних, посилання на яке зберігаються в основній реляційній базі.

3.1.3. Проєктування NoSQL бази даних для інформаційної управляючої системи

У межах розробки інформаційної управляючої системи для туристичного сервісу виникла потреба в забезпеченні високої швидкодії при зверненні до часто використовуваних, але рідко змінюваних даних — зокрема, локалізованих інтерфейсних елементів та географічних назв. Для цього було впроваджено базу даних NoSQL типу «ключ-значення» — Redis. Вигляд цієї бази даних у програмі Another Redis Desktop Manager показаний нижче (див. рис. 3.3).

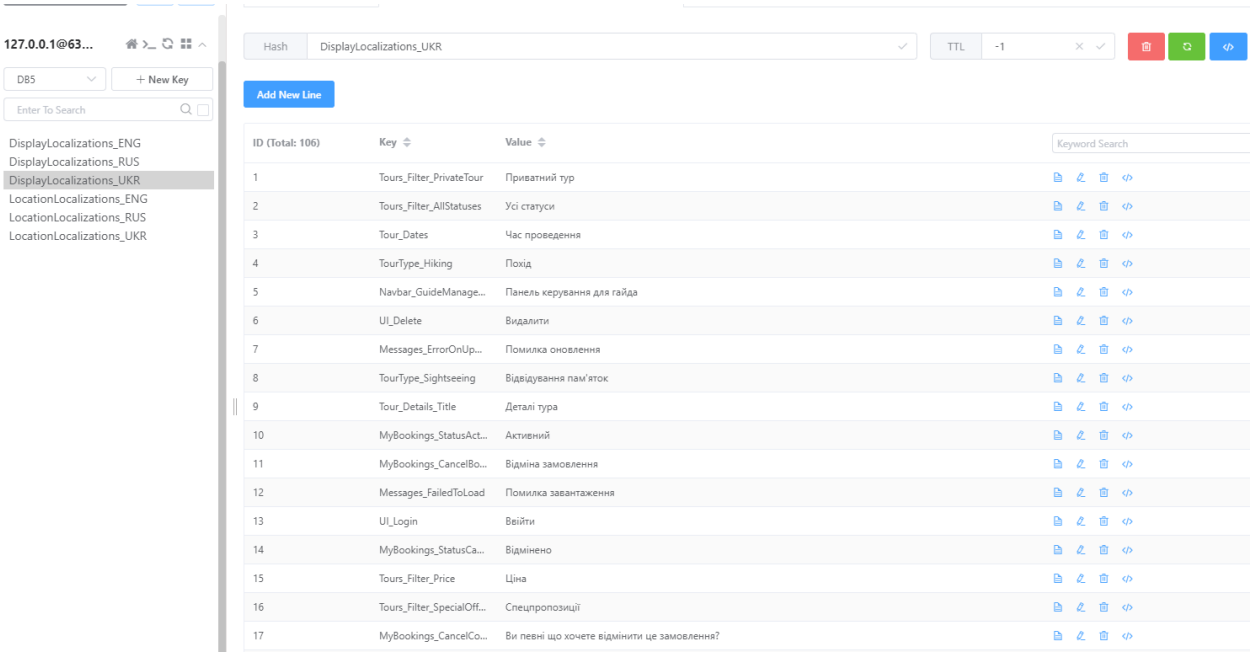


Рис. 3.3 – Вигляд БД Redis у програмі Another Redis Desktop Manager

Джерело: зроблено автором на основі виконаної розробки

Redis (Remote Dictionary Server) — це високопродуктивна система зберігання даних у пам’яті, яка забезпечує низьку затримку при зчитуванні ідеально підходить для реалізації кешування. У рамках проєкту Redis використовується як інструмент для зберігання локалізованих даних, що дозволяє значно зменшити навантаження на реляційну базу даних при виконанні запитів на відображення інтерфейсу користувача різними мовами.

У системі зберігаються дві категорії локалізованих даних:

- інтерфейсні підписи та тексти (кнопки, підказки, назви розділів, повідомлення тощо);
- назви локацій (міст, країн), що використовуються у турах.

Ці дані мають наступні властивості:

- слабоструктуровані: організовані у вигляді простих ключів-значень;
- рідко змінювані: редагуються адміністраторами в разі потреби;
- часто використовуються у фронтенді для відображення мовного інтерфейсу;
- великі за кількістю звернень, але не за обсягом.

Таким чином, застосування реляційної моделі з численними зверненнями до таблиць у цьому випадку є надмірним і знижує продуктивність. Обрана модель Redis у вигляді key-value дозволила забезпечити ефективну кешуючу інфраструктуру з мінімальною затримкою доступу.

Кожен запис у Redis відповідає певній категорії локалізованих даних для певної мови. Структура ключів має такий вигляд:

- DisplayLocalizations_UKR — локалізації елементів інтерфейсу для української мови;
- DisplayLocalizations_ENG — для англійської;
- DisplayLocalizations_RUS — для російської;
- LocationLocalizations_UKR — назви локацій українською;
- LocationLocalizations_ENG — англійською;
- LocationLocalizations_RUS — російською.

Значенням для кожного з ключів є об’єкт із ключами-полями інтерфейсу/локацій та відповідними перекладами.

Кеш Redis оновлюється у момент внесення змін до таблиці Languages основної реляційної БД. Це відбувається автоматично на рівні бекенд-логіки, після збереження нових локалізованих даних виконується очищення

відповідного ключа в Redis, після чого клієнт при наступному зверненні ініціює повторне зчитування з БД та кешування. Оскільки частота змін даних незначна, термін зберігання (TTL) ключів не встановлюється — тобто вони зберігаються у Redis до явного видалення або перезапису.

Redis-сервер розгорнутий у середовищі Docker, що спрощує його інфраструктурну інтеграцію та керованість. Взаємодія з Redis реалізована з використанням офіційного пакета StackExchange.Redis для платформи .NET. Сервер Redis функціонує лише як внутрішній кеш-сервер, недоступний ззовні, що відповідає вимогам безпеки.

Redis виступає допоміжною підсистемою для розвантаження запитів до БД та підвищення загальної швидкодії системи при обробці багатомовного інтерфейсу. Завдяки цьому користувачі отримують інтерфейс мовою обраної локалі практично миттєво, без додаткового навантаження на основну БД.

3.2. Проєктування бази знань інформаційної управляючої системи та/або засоби інтелектуального аналізу даних

3.2.1 Реалізація інтелектуального аналізу великих даних з метою наповнення бази знань

Одним із ключових завдань розробленої інформаційної управляючої системи є формування персоналізованих рекомендацій турів для користувачів на основі їх попередньої взаємодії із системою. Для реалізації цього завдання у системі здійснюється інтелектуальний аналіз даних (ІАД), що базується на комбінації структурованих даних з БД та сучасних методів генерації відповідей на основі великої мовної моделі (LLM).

Для побудови рекомендацій у системі використовуються дані з таблиць:

- TourBookings — історія бронювань користувача;
- TourInstanceRates — оцінки і коментарі, залишені користувачем після участі в турі;
- Tours — доступні тури для перегляду.

Ці дані зберігаються у реляційній базі даних PostgreSQL і постійно оновлюються в процесі взаємодії користувачів із системою. Як приклад — нижче наведені деякі з оцінок користувачів:

```
INSERT INTO public."TourInstanceRates"
("Rate", "TouristCommentary", "RatedTimeUtc", "TouristAccountId", "TourInstanceId")
VALUES (45, 'Amazing experience in the Alps! The views were breathtaking and our guide was very knowledgeable.', '2025-05-05 19:34:58.393', 1, 1);
```

Також відстежується частота, ціна та напрямки попередніх замовлень:

```
INSERT INTO public."TourBookings"
("AccountId", "TourInstanceId", "BookedUtc", "TotalPrice", "CancellationReason")
VALUES (1, 2, '2025-05-11 18:59:32.124', 3200.00, 'Cancelled by user');
```

Хоча повноцінний конвеєр ETL / ELT у системі не реалізований, проте присутня логіка, що еквівалентна витяганню, агрегації та подачі структурованих даних в інтерфейс до модуля ШІ.

При кожному запиті рекомендацій користувача, система генерує JSON-запит до OpenAI API, в якому враховуються такі ознаки:

– типи раніше замовлених турів;
– локації попередніх бронювань;
– оцінки попередніх турів;
– доступні на поточний момент тури;
– спеціальні пропозиції.

Ось приклад сформованого запиту до OpenAI API (скорочено):

```
{
  "model": "gpt-3.5-turbo",
  "messages": [
    {
      "role": "system",
      "content": "You are a tour recommendation expert. Analyze..."
    },
    {
      "role": "user",
      "content": "{...дані про попередні бронювання, оцінки і доступні тури...}"
    }
  ]
}
```

Результатом є відповідь у вигляді JSON-структури, яка містить список ідентифікаторів рекомендованих турів та обґрунтування вибору українською мовою:

```
{
  "recommendedTourIds": [2, 3, 1],
  "reason": "Рекомендую вам тур 'Scenic Italy Journey'..."
}
```

Сама логіка інтеграції реалізована у середовищі ASP.NET Core за допомогою C# і офіційного OpenAI SDK. У модулі працює власний механізм серіалізації/десеріалізації запитів і відповідей, а також збереження отриманих результатів для подальшого кешування.

Згенеровані рекомендації відображаються безпосередньо в інтерфейсі користувача у вигляді карток турів із зазначенням місця, вартості, тривалості, супроводу гіда тощо. Пояснення, яке надає модель, відображається нижче у вигляді персоналізованого тексту (див. рис. 3.4).

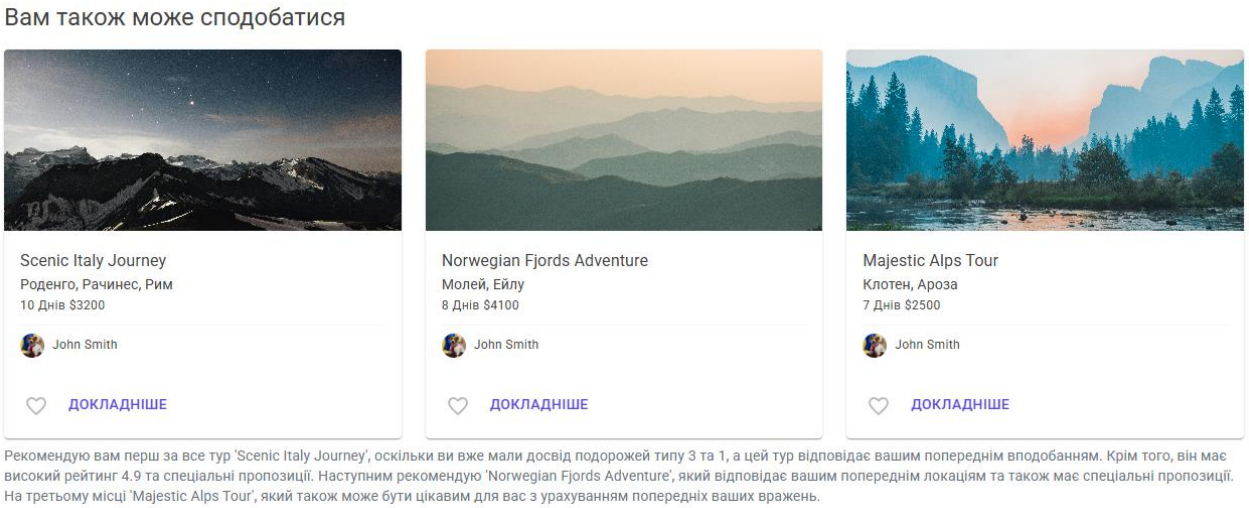


Рисунок 3.4 – Вигляд інтерфейсу з рекомендаціями для користувача

Джерело: зроблено автором на основі виконаної розробки

У прикладі, що представлений вище на рисунку 3.4, видно, як система рекомендує тури «Scenic Italy Journey», «Norwegian Fjords Adventure» та «Majestic Alps Tour» із поясненням причин вибору на основі попереднього досвіду користувача, високого рейтингу та відповідності до уподобань.

Цей підхід дозволяє обійти потребу у складній ручній побудові моделей ML чи використанні low-code інструментів, натомість спираючись на потужність LLM та просту інтеграцію через API. Він також є масштабованим та адаптивним до зміни структури турів, потреб користувачів і розширення логіки рекомендацій.

3.2.2 Проєктування та реалізація бази знань інформаційної управляючої системи

На етапі ідентифікації визначено головну задачу підсистеми бази знань – надавати персоналізовані рекомендації туристичних турів для користувачів веб-застосунку. Це означає, що система повинна аналізувати дані про попередні подорожі та вподобання туриста і пропонувати нові тури, які найбільше відповідають його інтересам. В рамках цієї загальної мети виділено кілька підзадач:

– врахування історії оцінок і бронювань користувача (які тури він замовляв раніше і як оцінював їх якість);

– врахування популярності напрямків та турів серед усіх користувачів (щоб рекомендувати не лише схожі на попередні, а й популярні варіанти);

– фільтрація доступних турів за актуальністю (наприклад, тільки активні тури, спецпропозиції, некатегоризовані як скасовані).

У базі знань цієї системи зберігаються такі основні типи даних:

– історія бронювань турів;

– оцінки турів – рейтинги, які користувачі виставляють пройденим турам, а також текстові відгуки про них;

– категорії та типи турів;

– локації турів;

– дані про тури.

Основна задача бази знань – об’єднати ці дані та на їх основі виявити, які нові тури найкраще підійдуть конкретному користувачеві. На етапі ідентифікації було сформовано вимогу, щоб система рекомендацій була інтегрована в ASP.NET Core веб-застосунок, використовувала наявну базу даних PostgreSQL з таблицями бронювань та оцінок, а також зверталася до зовнішнього AI-сервісу (OpenAI API) для генерації рекомендацій.

На етапі концептуалізації виконано змістовний аналіз предметної області туристичних рекомендацій і визначено ключові поняття та взаємозв’язки між ними. Основними поняттями (сутностями) доменної моделі є:

- користувач (Account);
- тур (Tour);
- екземпляр туру (TourInstance);
- бронювання (TourBooking);
- оцінка туру (TourInstanceRate);
- рейтинг туру;
- тип туру;
- локація.

Взаємозв’язки між цими сутностями визначають структуру знань предметної області. Кожен користувач може зробити багато бронювань, і кожне бронювання належить одному користувачу та одному екземпляру туру. Кожен екземпляр туру може мати багато пов’язаних бронювань (від різних користувачів) і багато оцінок (від користувачів, що пройшли цей тур). Кожна оцінка туру пов’язана з конкретним екземпляром туру і конкретним користувачем (тим, хто залишив оцінку). Кожен екземпляр туру відповідає певному туру (шаблону), а кожен тур належить до одного типу туру і може містити посилання на одну чи кілька локацій (через окрему таблицю або JSON-поле).

Окрім внутрішніх даних (бази даних), у концепції системи передбачено використання зовнішніх знань через API OpenAI. Логіка рекомендацій базується на тому, щоб передати модельовані знання про користувача та тури до мовної моделі OpenAI, яка виступає в ролі “движка” рекомендацій. Ця модель виконує роль експертної підсистеми, що на основі наданих фактів формує нові знання (рекомендації). Таким чином, в концептуальній моделі з’являється ще одна сутність – AI-модель рекомендацій, яка не зберігається в

локальній базі, але використовується для виведення (inferencing) рекомендацій на основі наданих фактів.

На етапі формалізації визначено, яким чином всі необхідні знання будуть представлені в системі та передаватимуться для обробки. Вибрано реляційну базу даних (PostgreSQL) для зберігання фактів і параметрів (бронювань, оцінок, турів тощо) і формат JSON для передачі узагальненої інформації в зовнішню модель OpenAI.

Подання знань в базі даних: сформовано структуру таблиць, описану вище, що однозначно зберігає всі необхідні факти. Кожен запис у TourBookings та TourInstanceRates являє собою окремий факт (про здійснене бронювання або залишену оцінку відповідно). Ці таблиці пов’язані з довідниковими таблицями Tours, Accounts, Locations тощо, що задають контекст для фактів.

Формування запиту для рекомендації: перед генерацією рекомендації система вибирає релевантні знання про користувача з бази даних.

Представлення знань у форматі JSON: профіль користувача та дані про потенційні тури для рекомендації серіалізуються у структурований формат (наприклад, як JSON-об’єкт), який буде передано в запиті до OpenAI. Використання JSON дозволяє чітко структурувати факти: окремо перелічити попередні вподобання користувача і окремо – список доступних турів-кандидатів з їх характеристиками. Передавання даних у такому структурованому вигляді спрощує інтерпретацію цих даних мовною моделлю.

Фільтрація даних: щоб обмежити обсяг переданої інформації (зважаючи на ліміт токенів моделі), до JSON-запиту включаються лише найважливіші поля. Наприклад, для турів обираються поля: назва, тип, ключові локації, середній рейтинг, наявність спеціальних пропозицій. Детальні описи чи зайві атрибути не передаються. Для профілю користувача фіксуються тільки ті характеристики, які впливають на вибір туру (категорії попередніх турів, регіони подорожей, можливо середній бюджет тощо).

Додання інструкції для моделі: окрім даних, у запит включається текстова інструкція або система повідомлень (system/user prompt), яка пояснює моделі, що потрібно зробити – а саме, обрати найкращі тури для цього користувача та обґрунтувати вибір українською мовою.

На етапі реалізації знання, підготовлені на попередніх етапах, були втілені у вигляді програмного коду, структури бази даних та інтеграції з зовнішніми сервісами. База знань “живе” одночасно у базі даних і у логіці застосунку.

Реляційна БД (PostgreSQL) зберігає всі факти та дані: таблиці TourBookings, TourInstanceRates, Tours, TourInstances, Accounts тощо реалізовано через Entity Framework Core (Code First) відповідно до концептуальної моделі. Наприклад, таблиця TourBookings має поля Id, AccountId (зовнішній ключ на користувача), TourInstanceId (зовнішній ключ на екземпляр туру), BookedUtc (дата та час бронювання), TotalPrice (вартість) та поля скасування (CancellationReason, CancellationDate, IsCancelled). Таблиця TourInstanceRates має поля Id, TourInstanceId (зв’язок з пройденим туром), TouristAccountId (зв’язок з акаунтом користувача-автора відгуку), Rate (числова оцінка, 1–5) і TouristCommentary (текст відгуку), а також RatedTimeUtc (час залишення оцінки). Ці дані наповнюються у процесі роботи системи: коли користувач бронює тур, додається запис у TourBookings; коли оцінює пройдений тур – додається запис у TourInstanceRates. Таким чином база знань постійно актуалізується новими фактами.

Логіка на рівні застосунку (ASP.NET Core, C#) реалізує отримання й обробку знань. Коли потрібна рекомендація, застосунок (наприклад, у відповідному контролері) виконує запити до БД, щоб витягти потрібні факти про поточного користувача. Потім ці дані серіалізуються у формат (JSON/текст), визначений на етапі формалізації, і передаються до зовнішньої AI-моделі. Взаємодія з OpenAI API здійснюється через офіційну .NET-бібліотеку OpenAI .NET SDK. У коді це реалізовано як виклик до API

модельної служби, з передачею згенерованого prompt та необхідних параметрів (температура генерації, максимальна довжина відповіді тощо).

Інтеграція з Redis використовується для кешування та швидкого доступу до даних, що не потребують кожного разу звернення до БД. Наприклад, можна кешувати результати частих запитів (список популярних турів, довідники локацій або категорій) в пам’яті Redis, щоб прискорити формування рекомендації. У нашій реалізації Redis під’єднано як distributed cache до ASP.NET Core, але знання, які генерує модель (рекомендації для конкретного користувача), не кешуються постійно. Це зроблено з метою, щоб кожна рекомендація враховувала найсвіжіші дані (нові оцінки, нові тури тощо) і щоб уникнути застарілих або повторюваних порад. Тим не менш, кеш може використовуватися для тимчасового зберігання вже згенерованих рекомендацій протягом однієї сесії користувача, аби при повторному запиті протягом короткого часу не виконувати однаковий запит до OpenAI API.

Зберігання зображень (Cloudinary): медіа-контент, такий як фотографії турів, зберігається не в самій БД, а у хмарному сховищі Cloudinary. В базі знань турів зберігаються лише посилання (URL) на зображення, що дозволяє легко отримувати і відображати їх у веб-інтерфейсі. Це рішення прийнято для оптимізації зберігання даних та швидкого доставлення зображень користувачам без навантаження на основну систему.

Таким чином, реалізація бази знань є розподіленою: факти знаходяться у PostgreSQL, а логіка виведення нового знання (персоналізованої рекомендації) реалізована у коді на C# із залученням можливостей штучного інтелекту через OpenAI API. Важливо, що результати роботи цієї підсистеми (список рекомендованих турів та пояснення) наразі не зберігаються в базі даних постійно – вони генеруються динамічно на вимогу. Це дозволяє уникнути накопичення потенційно застарілої інформації та гарантує, що користувач завжди отримує актуальну рекомендацію.

Після розробки систему рекомендацій було піддано тестуванню, щоб перевірити якість наданих рекомендацій і правильність роботи бази знань.

Тестування проводилось як на основі штучних сценаріїв (тестові користувачі з заздалегідь відомими вподобаннями), так і в режимі дослідної експлуатації із залученням реальних даних. У ході тестових прогонів перевіряли:

Коректність вибірки даних: чи правильно витягуються з БД всі потрібні факти про користувача (бронювання, оцінки) і чи відповідають вони очікуванням. Для цього створювались ситуації, де відомо, які тури повинен запропонувати алгоритм. Наприклад, користувачу з історією кількох пригодницьких турів система мала б рекомендувати переважно пригодницькі тури.

Адекватність рекомендацій: аналізувався зміст згенерованих OpenAI рекомендацій. Результати виявились доволі релевантними: модель, отримавши структуровані дані, пропонує тури, що логічно узгоджуються з попереднім досвідом користувача. Наприклад, у одному зі сценаріїв тестування користувач мав досвід турів категорій 3 та 1 (умовно, екстрим і екскурсії); система порекомендувала йому тур “Scenic Italy Journey” як перший варіант, оскільки той теж належить до улюбленої категорії і містить знайомі користувачу локації (Італія). Наступними були “Norwegian Fjords Adventure” та “Majestic Alps Tour”, які відповідають його попереднім вподобанням за типом і географією. Цей результат співпав з очікуваннями тестувальників – рекомендації дійсно відображали вподобання користувача.

Якість пояснень: система не тільки видає список турів, але й генерує пояснення, чому обрано ті чи інші варіанти. Було перевірено, що ці пояснення зрозумілі користувачеві, коректні фактично (наприклад, якщо згадується “високий рейтинг 4.9”, то тур дійсно має такий рейтинг), і не містять суперечностей. Модель успішно обґрунтовує рекомендації, використовуючи надані їй дані (категорії, локації, рейтинги турів тощо).

Результати тестування показали, що підсистема бази знань і рекомендацій працює згідно з поставленими вимогами: користувачам пропонуються релевантні тури, а логіка врахування їх попереднього досвіду реалізована правильно. Виявлені незначні недоліки (наприклад, подекуди

повторюваність формулювань у поясненнях або необхідність тоншого налаштування параметрів моделі) були проаналізовані для подальшого вдосконалення.

На етапі дослідної експлуатації система була запущена для реальних користувачів в обмеженому режимі. Користувачі взаємодіють із системою через веб-інтерфейс: після автентифікації на сайті їм стає доступний персоналізований блок рекомендацій. Інтерфейс у розділі рекомендацій відображає кілька карток турів зі стислою інформацією (назва, місце, тривалість, ціна) та кнопкою для детального перегляду. Під добіркою турів надається згенерований пояснювальний текст, який обґрунтовує, чому саме ці тури можуть сподобатися користувачу.

Завдяки реалізації описаної бази знань інформаційна управляюча система змогла забезпечити інтелектуальну підтримку користувачів у виборі турів. Персоналізовані рекомендації, підкріплені даними про попередній досвід та згенерованими AI поясненнями, підвищують залученість користувачів і цінність веб-застосунку загалом.

3.3. Програмне забезпечення

3.3.1 Проєктування програмного забезпечення

Застосунок розроблено за монолітною архітектурою, тобто всі його компоненти розгортаються як єдине ціле. Водночас структура рішення побудована модульно, що в майбутньому дозволить виділити окремі частини в самостійні сервіси при потребі масштабування. Проєкт логічно поділено на чотири основні частини: Client, Backend, Common та Tests. Client містить інтерфейс користувача (UI) реалізований на Blazor; Backend включає веб-контролери та бізнес-логіку; Common містить спільні класи моделей і DTO (об’єкти перенесення даних), які використовуються між клієнтом і сервером; Tests – проєкт модульних тестів для перевірки роботи застосунку. У застосунку дотримано шаблону архітектури MVC (Model-View-Controller): роль Model виконують класи моделей і DTO у спільній бібліотеці Common, Controller – це контролери в Backend, а View – інтерфейс, реалізований у клієнтському модулі на Blazor. Для побудови гнучкого і підтримуваного коду застосовано низку патернів та принципів проектування: Dependency Injection (впровадження залежностей) для керування залежностями між компонентами; патерн Repository – для абстрагування доступу до бази даних через сховище даних; дотримано принципів DRY (Don’t Repeat Yourself – «не повторюйся») та KISS (Keep It Simple, Stupid – «роби простіше»), що сприяють спрощенню підтримки коду; а також впроваджено підхід уніфікованого повернення результатів операцій (Results pattern), щоб всі методи логіки повертали стандартний об’єкт результату з інформацією про успіх або помилку. На рисунку представлена загальна архітектурна схема застосунку у вигляді блок-схеми (див. рис. 3.5).

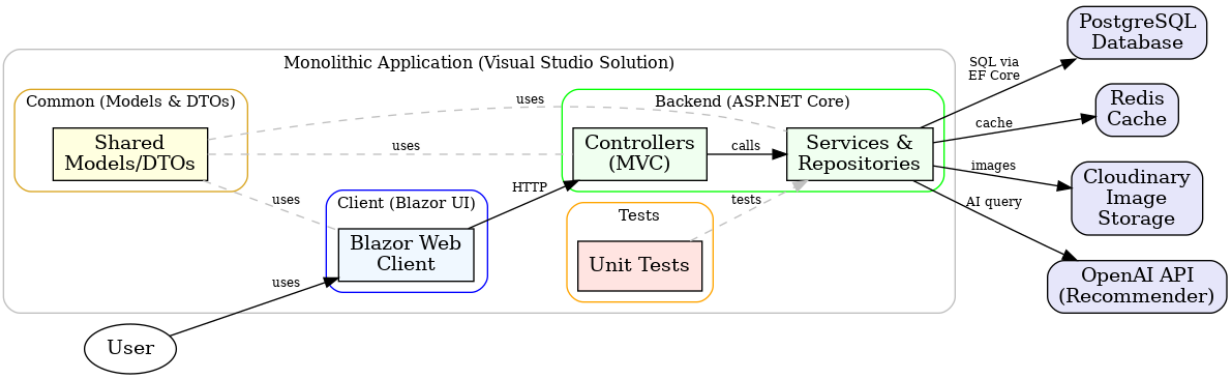


Рисунок 3.5 – Архітектурна схема застосунку (монолітна архітектура)

Джерело: зроблено автором на основі виконаної розробки

Показано поділ на проєкти Client (Blazor UI), Backend (ASP.NET Core контролери та сервіси), Common (спільні моделі й DTO) та Tests (модульні тести), а також взаємодію Backend з зовнішніми компонентами: базою даних PostgreSQL, кешем Redis, сховищем зображень Cloudinary та API рекомендацій OpenAI.

Для реалізації застосунку використано сучасний стек технологій, мов програмування та інструментів:

– мова та платформа: C# на платформі .NET Core (фреймворк ASP.NET Core для веб-розробки бекенду);

– база даних: PostgreSQL – реляційна СУБД для надійного зберігання даних застосунку;

– кешування: Redis – система кешування в пам’яті, що використовується для прискорення доступу до часто запитуваних даних;

– хмарне сховище: Cloudinary – зовнішнє хмарне сховище для зберігання та доставлення зображень (фото турів, аватарів користувачів тощо);

– зовнішній AI-сервіс: OpenAI API – сервіс штучного інтелекту (наприклад, модель GPT), використаний для генерації рекомендацій турів користувачам;

– ORM: Entity Framework Core – об’єктно-реляційний mapper для взаємодії з PostgreSQL на рівні об’єктів C# (спрощує роботу з базою даних);

– UI-бібліотека: MudBlazor – бібліотека компонентів інтерфейсу для Blazor, що забезпечує готові UI-компоненти та стилі для швидкої побудови красивого інтерфейсу;

– логування: Serilog – бібліотека для структурованого логування, використовується для запису журналів подій і помилок роботи застосунку;

– документація API: Swagger (OpenAPI) – інструмент для автоматичної генерації інтерактивної документації REST API бекенду (можна переглядати ендпоінти і тестувати їх через веб-інтерфейс Swagger UI);

– автентифікація: JWT (JSON Web Tokens) – технологія токен-базованої автентифікації; використовується для шифрування інформації про користувача в токені, що додається до кожного запиту для перевірки доступу.

Клієнтський інтерфейс реалізований за допомогою технології Blazor (WebAssembly), що дозволяє запускати інтерфейс прямо в браузері і динамічно взаємодіяти з сервером через HTTP-запити до API. Верстка інтерфейсу виконана адаптивно, тобто підтримується responsive design під різні розміри екранів. Завдяки цьому користувач може зручно працювати із застосунком як на десктопі, так і на смартфоні або планшеті – інтерфейс підлаштовується під роздільну здатність пристрою (підтримується мобільний перегляд). На головній сторінці веб-застосунку представлено вітальний контент і список доступних турів; передбачено зручну навігацію по розділах сайту через меню (див. рис. 3.6).

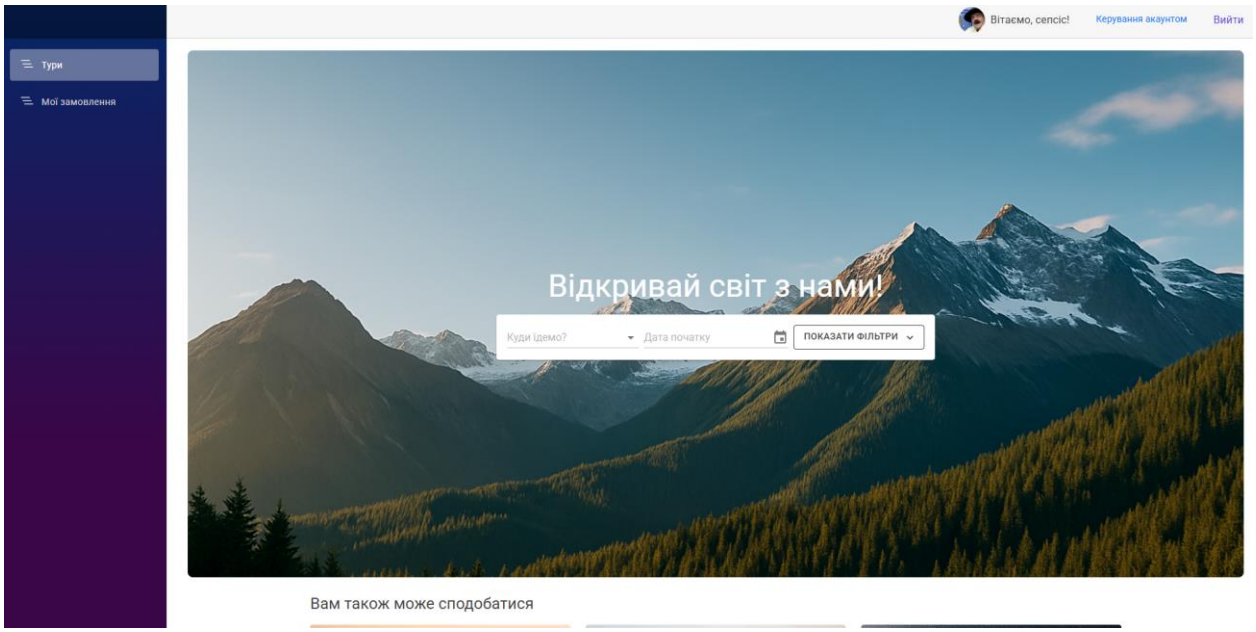


Рисунок 3.6 – Головний екран застосунку

Джерело: зроблено автором на основі виконаної розробки

Архітектура застосунку передбачає розподіл функціональності на кілька модулів (компонентів):

Модуль автентифікації та авторизації: відповідає за реєстрацію нових користувачів, вхід до системи і керування доступом (див. рис. 3.7). При автентифікації використовується JWT – після успішного входу користувачу видається JWT-токен, який надалі надсилається з запитами для перевірки прав. Модуль також опікується шифруванням паролів та перевіркою ролей користувача (адміністратор, клієнт тощо) при доступі до захищених ресурсів.

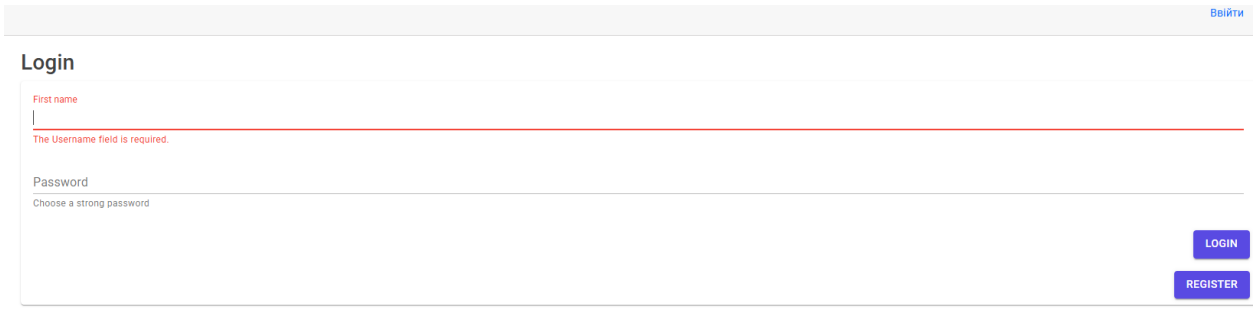


Рисунок 3.7 – Меню логіну

Джерело: зроблено автором на основі виконаної розробки

Модуль турів: забезпечує відображення списку доступних турів та перегляд детального опису вибраного туру. В цьому модулі реалізовано логіку отримання даних про тури з бази, формування списків турів для відображення на головній та інших сторінках, а також показ детальної інформації (опис, ціна, тривалість, фотографії тощо) по кожному туру. Користувач може переглядати характеристики туру, доступні дати та іншу інформацію, як показано на рисунку (див. рис. 3.8).

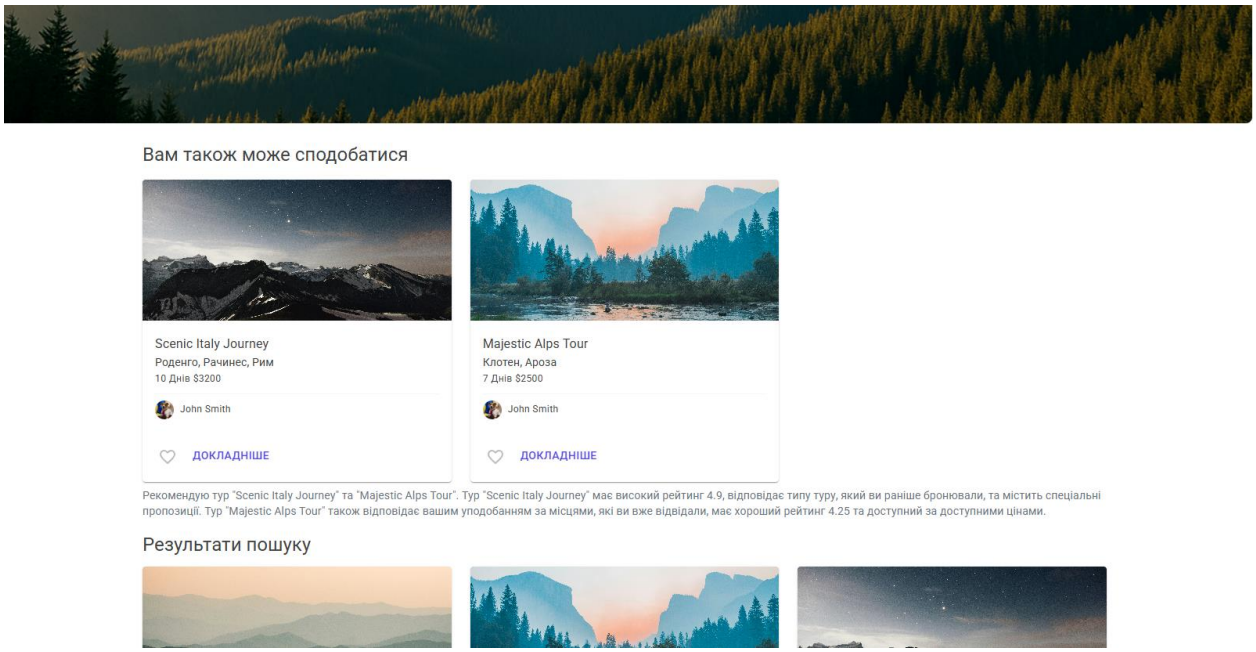


Рисунок 3.8 – Перегляд турів

Джерело: зроблено автором на основі виконаної розробки

Модуль перегляду тура: надає користувачам можливість забронювати обраний тур та переглянути повну інформацію про нього. Логіка цього модуля включає перегляд дати подорожі, кількості учасників. Також цей модуль має в собі функцію замовлення туру по натисненні на відповідну кнопку (див. рис. 3.9).

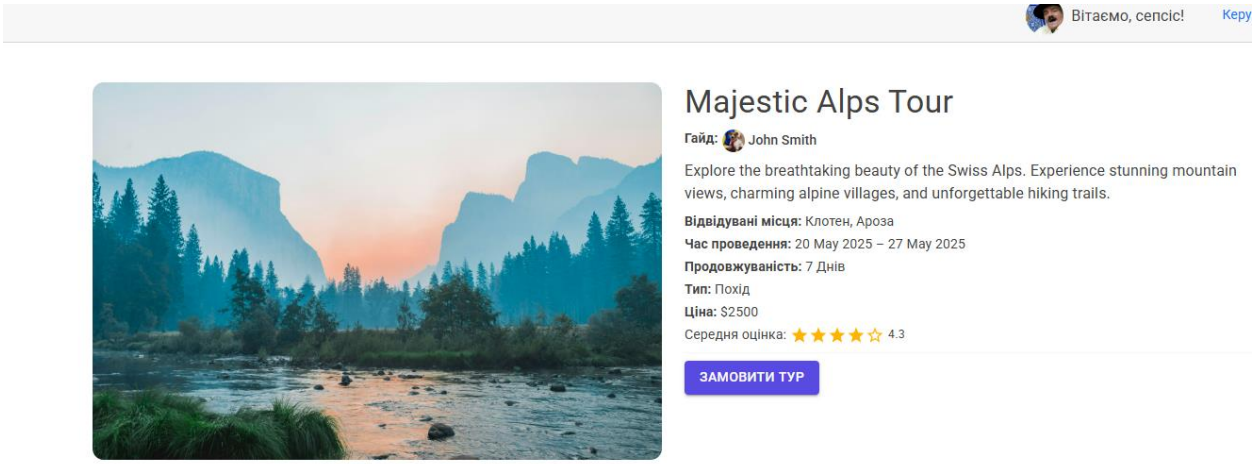


Рисунок 3.9 – Перегляд туру

Джерело: зроблено автором на основі виконаної розробки

Усі замовленні користувачем тури зберігаються в системі, та їх можна переглянути зі сторінки, перейшовши у меню «Мої замовлення», як це показано на рисунку (див. рис. 3.10). У цьому меню користувач може також відмінити майбутнє замовлення, або ж оцінити пройдене, вигляд меню де користувач може залишити відгук показане на наступному рисунку (див. рис. 3.11).

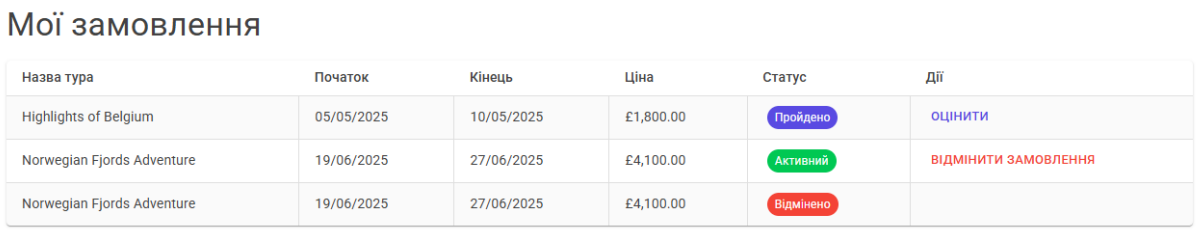


Рисунок 3.10 – Перегляд замовлень користувача

Джерело: зроблено автором на основі виконаної розробки

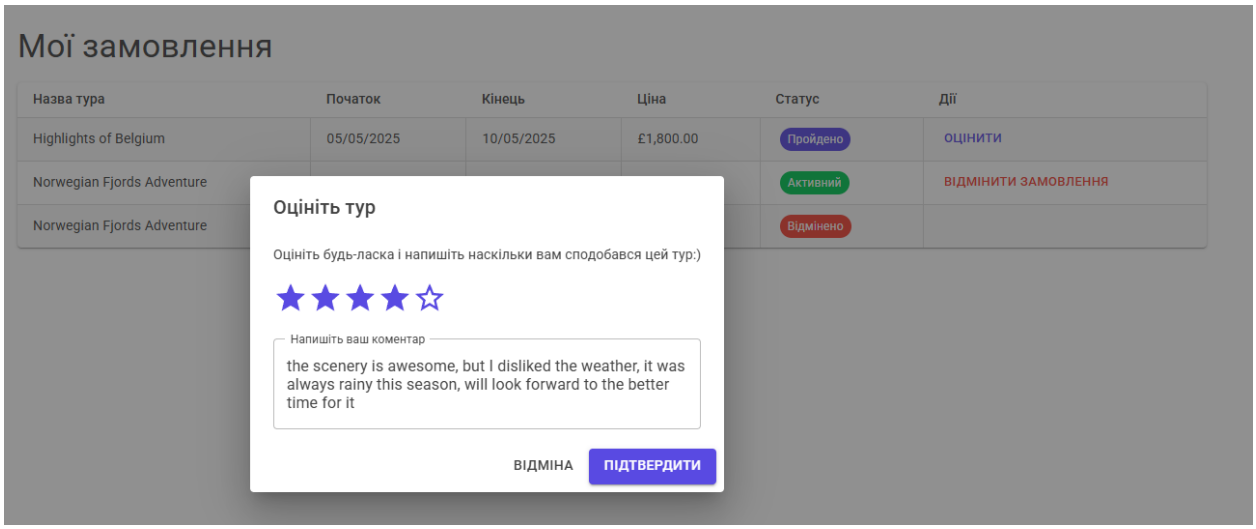


Рисунок 3.11 – Діалогове вікно оцінки

Джерело: зроблено автором на основі виконаної розробки

Модуль локалізацій та контенту: відповідає за мультимовність та керування статичним контентом. Для підтримки кількох мов інтерфейсу і контенту (наприклад, української та англійської) реалізовано механізм локалізації: текстові ресурси інтерфейсу та описів можуть завантажуватися згідно з вибраною мовою. Задля прискорення роботи застосунку ці дані кешуються в Redis – це зменшує кількість звернень до бази даних при переключенні мови або повторному доступі до одних і тих самих даних. Окремо модуль керує зберіганням та отриманням зображень: всі фотографії турів, аватарки користувачів та інші медіафайли завантажуються у хмарне сховище Cloudinary, звідки потім швидко завантажуються в інтерфейс за посиланнями. Такий підхід розвантажує сервер та прискорює доставку мультимедіа контенту користувачам (див. рис. 3.12).

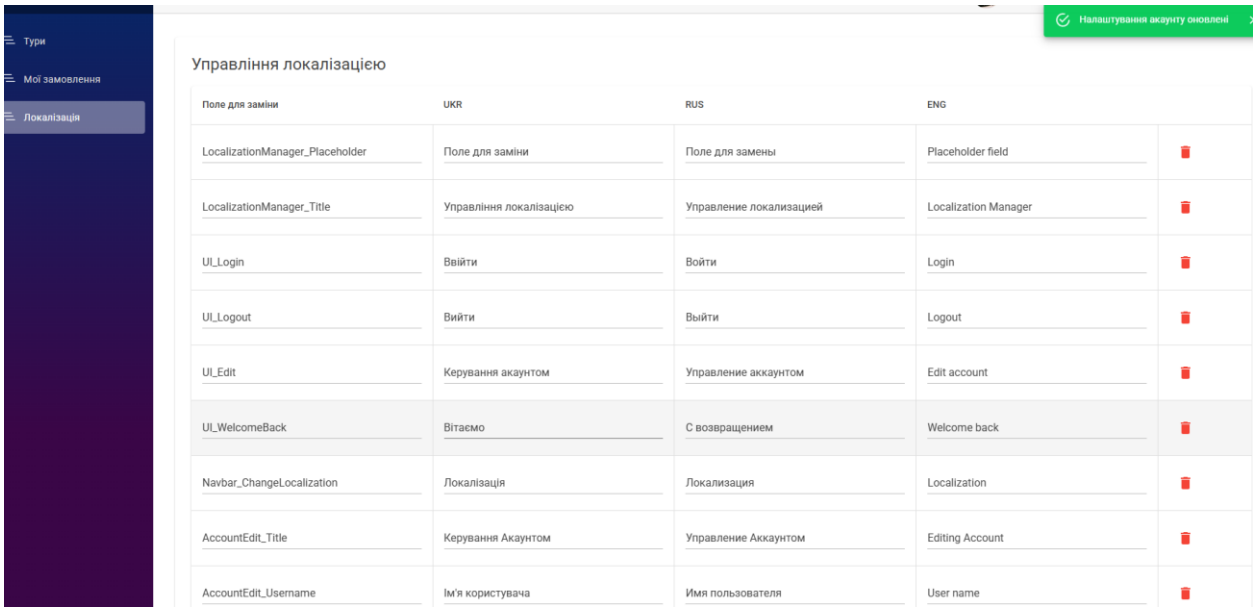


Рисунок 3.12 – Меню керування локалізацією

Джерело: зроблено автором на основі виконаної розробки

Модуль рекомендацій: реалізує функцію інтелектуальних рекомендацій турів для користувачів. На основі вподобань користувача, його історії переглядів або заданих критеріїв, система генерує персональні рекомендації щодо турів, які можуть його зацікавити. Для цього інтегровано зовнішній OpenAI API – модуль надсилає необхідні дані (наприклад, опис інтересів користувача) до сервісу штучного інтелекту, який аналізує інформацію і повертає список рекомендованих турів. Рекомендації відображаються на спеціальній сторінці інтерфейсу, що показано на рисунку 3.7. Даний підхід дозволяє покращити користувацький досвід шляхом пропонування релевантних варіантів відпочинку з використанням AI.

Модуль управління акаунтом: забезпечує функціонал кабінету користувача. Зареєстрований користувач може переглянути і відредагувати ім’я, аватар, мову інтерфейсу. Управління акаунтом підвищує зручність користування системою, дозволяючи підтримувати актуальність інформації профілю (див. рис. 3.13).

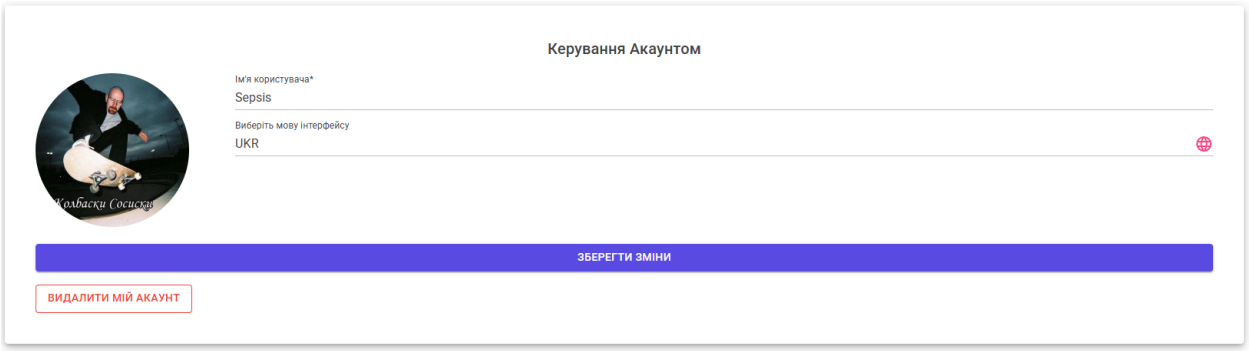


Рисунок 3.13 – Меню редагування акаунту

Джерело: зроблено автором на основі виконаної розробки

Як видно на рисунку 3.14, тут можна змінити мову інтерфейсу на іншу, до прикладу при зміні мови на англійську головний екран виглядатиме так (див. рис. 3.14).

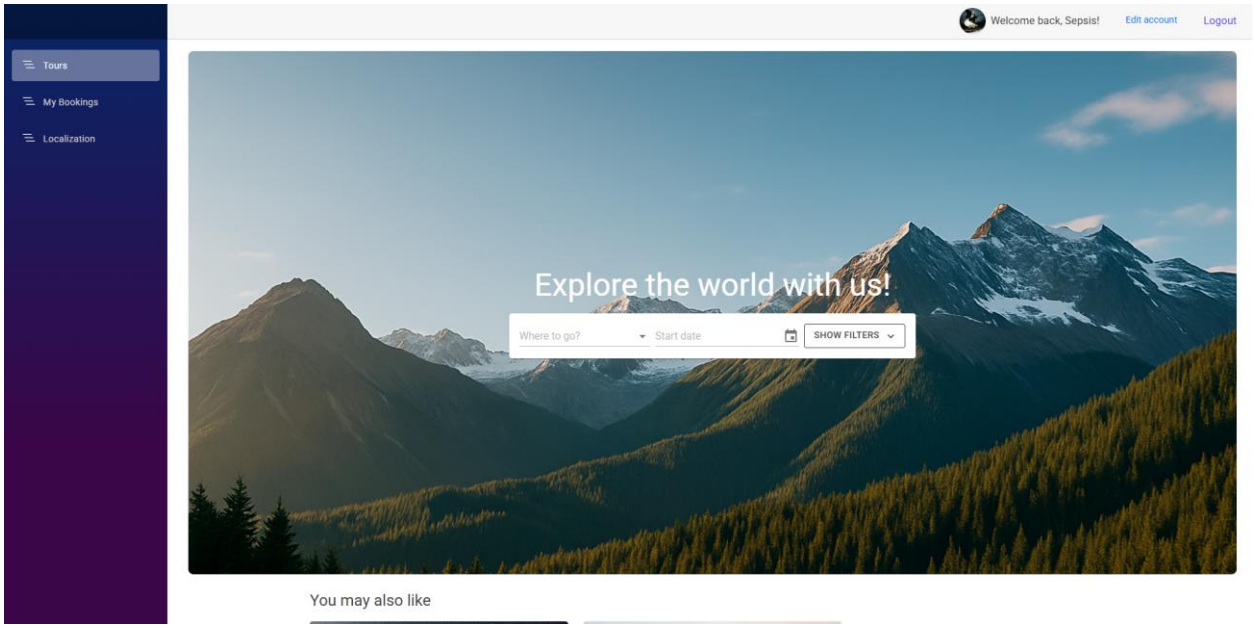


Рисунок 3.14 – Головне меню англійською мовою

Джерело: зроблено автором на основі виконаної розробки

Проєкт розгортається у середовищі Microsoft Visual Studio як єдине рішення, що містить всі згадані модулі. Для зберігання даних використовується PostgreSQL, а для кешування – Redis; під час розробки ці сервіси розгорнуті у вигляді контейнерів Docker (див. рис. 3.15). Такий підхід

спрощує налаштування середовища розробника – достатньо запуску відповідних контейнерів для бази даних та кешу. Сам застосунок (Backend і Client) запускається безпосередньо з Visual Studio, що забезпечує зручну відладку та тестування. У перспективі планується розгорнути застосунок на хмарній платформі AWS (Amazon Web Services). Розгортання в AWS дозволить забезпечити масштабованість, високу доступність та резервування даних. Можливі варіанти деплойменту включають використання сервісів AWS Elastic Beanstalk або контейнеризацію застосунку і запуск у AWS ECS/EKS. Перенесення PostgreSQL і Redis у хмару також може бути здійснено через відповідні керовані сервіси AWS (Amazon RDS для PostgreSQL, ElastiCache для Redis).

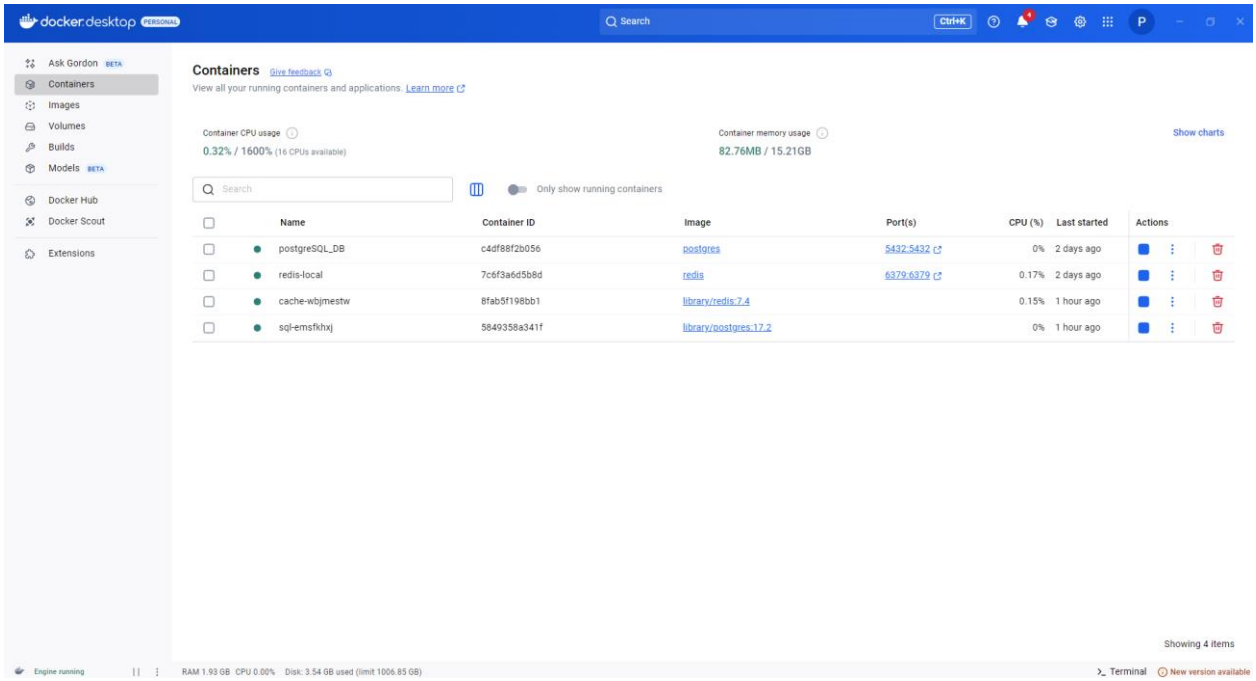


Рисунок 3.15 – Меню Docker

Джерело: зроблено автором на основі виконаної розробки

Для забезпечення якості та надійності роботи коду в проєкті реалізовано модуль Tests з юніт-тестами. Написано набір модульних тестів (unit-тестів) для перевірки ключових аспектів бізнес-логіки – зокрема, методів сервісів та репозиторіїв. Тести охоплюють сценарії успішного виконання

операцій та обробки помилкових ситуацій, перевіряючи, що модулі дотримуються очікуваної поведінки. Регулярне запускання тестів під час розробки допомагає вчасно виявляти регресії та помилки. Таким чином, впровадження автоматизованого тестування сприяє підтримці стабільності застосунку та відповідності його функціональності поставленим вимогам. У таблиці наведено головні функціональні модулі (сервіси) застосунку та структури даних (DTO), які вони використовують для обміну інформацією між фронтендом і бекендом

Таблиця 3.10 – Основні сервіси системи та відповідні DTO.

Модуль / Сервіс	**Функціональність (короткий опис)**	**DTO (об'єкти передачі даних)**
Автентифікація та авторизація	Реєстрація нових користувачів; автентифікація (вхід) існуючих; генерація і валідація JWT-токенів; контроль доступу до ресурсів на основі ролей.	UserDTO (дані користувача), LoginRequestDTO, RegisterDTO (дані для входу/реєстрації), AuthResponseDTO (JWT-токен та інформація про сесію).
Управління турами	Створення та редагування турів (адмін-функції); отримання списку турів; перегляд детальної інформації про тур (опис, ціна, характеристики).	TourDTO (основна інформація про тур), TourDetailsDTO (детальний опис туру, включно з розширеними полями).
Бронювання турів	Оформлення бронювання вибраного туру: вибір дати, кількості осіб; збереження бронювання в базі; видача підтвердження бронювання.	BookingRequestDTO (дані запиту на бронювання), BookingDTO (інформація про здійснене бронювання), BookingConfirmationDTO (підтвердження для користувача).
Локалізація та контент	Керування локалізованими текстовими даними (переклади інтерфейсу	LocalizationDTO (структура для зберігання перекладених текстів або налаштувань мови), ImageDTO (посилання на

Модуль / Сервіс	Функціональність (короткий опис)	DTO (об’єкти передачі даних)
	і описів); кешування цих даних для швидкого переключення мов; зберігання та отримання зображень з Cloudinary.	зображення/ресурс в Cloudinary).
Рекомендації	Формування персоналізованих рекомендацій турів на основі профілю та дій користувача; отримання списку рекомендованих турів через зовнішній AI-сервіс.	RecommendationRequestDTO (дані для запиту рекомендацій, напр. вподобання), RecommendationResultDTO (список рекомендованих турів або ідентифікаторів турів; може використовувати готові TourDTO).
Керування акаунтом	Керування профілем користувача: перегляд та оновлення особистих даних; зміна пароля; перегляд історії бронювань користувача (його замовлень).	UserProfileDTO (дані профілю користувача), UserUpdateDTO (структура для оновлення даних профілю), BookingHistoryDTO (список бронювань користувача для перегляду історії).

Джерело: сформовано автором на основі виконаної розробки

3.3.2 Вибір архітектури програмного забезпечення

Для розробки веб-системи було обрано типову монолітну архітектуру на платформі ASP.NET Core. Застосунок реалізовано з використанням фреймворку ASP.NET Core MVC (патерн Model-View-Controller) у поєднанні з технологією Blazor для побудови інтерактивного інтерфейсу користувача. Як основне сховище даних використовується реляційна база PostgreSQL, для кешування – сервіс Redis, а для інтеграції зі сторонніми сервісами – Cloudinary (зберігання зображень) та OpenAI API (генеративні AI-функції). Безпека та управління доступом реалізовані на основі JWT (JSON Web Token). Нижче наведено обґрунтування вибору такої архітектури, а також засоби забезпечення безпеки, ідентифікації та авторизації користувачів у системі.

Монолітна та модульна архітектура. Монолітний підхід означає, що вся серверна логіка, UI та доступ до даних інтегровані в єдиний застосунок, який розгортається як одне ціле. Така монолітна архітектура спрощує початкову розробку і розгортання, оскільки всі компоненти взаємодіють у межах одного процесу [24]. Для порівняння, в мікросервісній архітектурі кожна функціональна підсистема виділена у окремий сервіс, що дозволяє незалежне розгортання, але додає складності в оркестрації та комунікації між сервісами. З огляду на обсяг дипломного проєкту та обмежені ресурси, було раціонально обрати моноліт, який повністю покриває поточні вимоги і не потребує складної інфраструктури.

Водночас монолітний застосунок спроєктовано з урахуванням внутрішньої модульності – тобто, реалізовано концепцію модульного моноліту. Модульний моноліт поєднує переваги модульного дизайну з простотою моноліту, розділяючи систему на набір слабо-зв’язаних модулів із чітко визначеними межами відповідальності [25]. У контексті системи це означає, що різні підсистеми (управління турами, бронювання, управління користувачами, тощо) структуровані як окремі логічні компоненти всередині одного застосунку. Такий підхід підвищує підтримуваність коду та ізолює

зміни в межах модуля, мінімізуючи вплив на інші частини. Надалі це спрощує трансформацію у модульний моноліт на практиці – поступове виділення модулів у самостійні сервіси за потреби. Планується, що зі зростанням проекту деякі з цих модулів можуть бути винесені як окремі мікросервіси для незалежного масштабування та розгортання. Іншими словами, модулі моноліту слугують кандидатами на майбутні мікросервіси, що дозволяє еволюційно перейти до мікросервісної архітектури, не «стрибнувши» в неї передчасно. Такий поетапний підхід знижує ризики та зберігає простоту на ранніх етапах розробки.

Інтеграція компонентів. На рисунку схематично (через .NET Aspire Dashboard) відображено складові системи та їх стан (див. рис. 3.16). Монолітний застосунок взаємодіє з декількома ключовими компонентами: базою даних PostgreSQL, кеш-пам’яттю Redis та зовнішніми API. PostgreSQL забезпечує збереження основних даних (користувачі, тури, бронювання тощо), підтримуючи цілісність транзакцій. Redis використовується як зовнішнє кеш-сховище для прискорення доступу до часто запитуваної інформації та зменшення навантаження на базу даних. Наприклад, довідникові дані (список локацій, мов інтерфейсу) можуть кешуватися в пам’яті на заданий час, щоб при повторних запитах їх не витягувати щоразу з БД. Це підвищує продуктивність системи, особливо під навантаженням. Зовнішні сервіси OpenAI API та Cloudinary інтегровані через відповідні клієнтські бібліотеки чи HTTP-запити. OpenAI API використовується для генерації рекомендаційних текстів та інших функцій штучного інтелекту (наприклад, персоналізовані описи турів для користувача), а Cloudinary — для безпечного зберігання медіа-файлів (зображень турів, аватарів тощо) поза межами власної інфраструктури. Така інтеграція дозволяє не зберігати важкі файли локально і розвантажує сервер при обробці зображень.

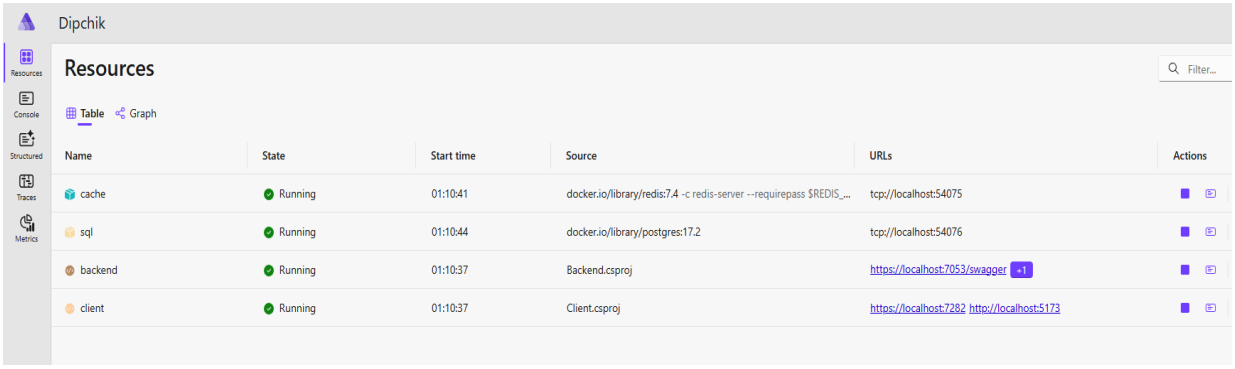


Рисунок 3.16 – Aspire Dashboard

Джерело: зроблено автором на основі виконаної розробки

Моніторинг та масштабування. Обраний стек технологій підтримує засоби моніторингу стану застосунку. Зокрема, у .NET середовищі доступна панель Aspire Dashboard для відстеження здоров’я сервісів (Health Checks). На рисунку показано приклад вкладки Health Check зі списком служб та їх статусом (див. рис. 3.17). Кожен внутрішній сервіс (наприклад, підключення до БД, до Redis, зовнішніх API) має індикатор стану. Якщо всі компоненти функціонують нормально, їх статус позначено як Healthy (зелена відмітка). У випадку проблем (відсутність підключення, помилки) панель відобразить Unhealthy або Degraded, що дозволяє оперативно виявити та усунути несправності. Такий моніторинг важливий при переході до більш розподіленої системи: він сприяє підтримці високої доступності та спрощує діагностику під час масштабування.

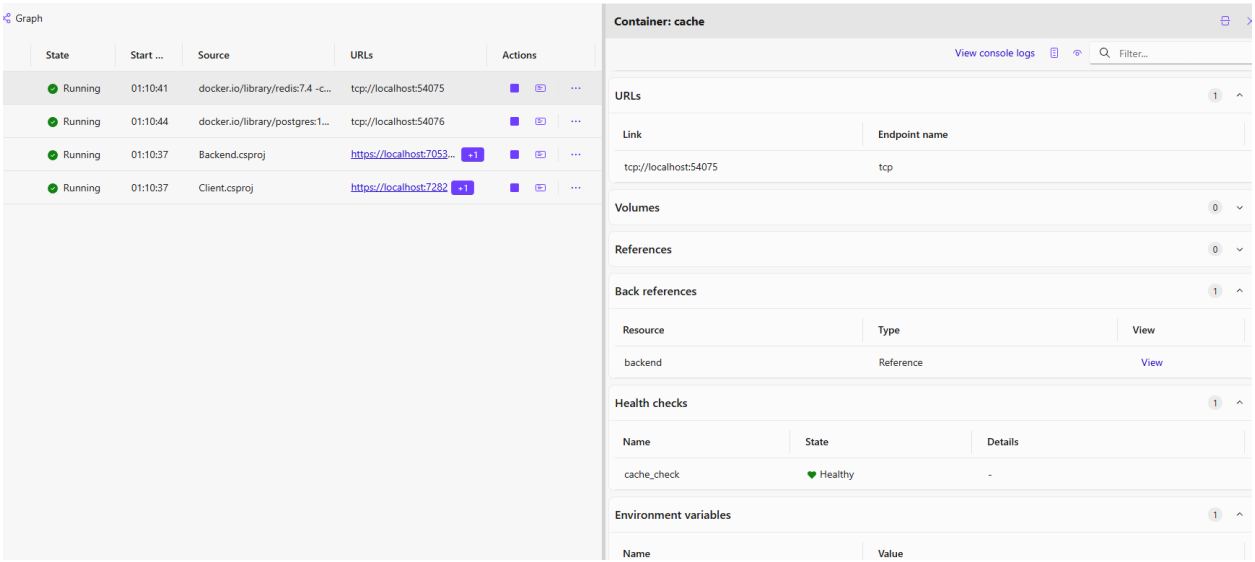


Рисунок 3.17 – Перегляд ресурсів в Aspire

Джерело: зроблено автором на основі виконаної розробки

Ідентифікація та автентифікація користувачів. У системі реалізовано власний механізм управління обліковими записами. Кожен користувач має обліковий запис (таблиця Accounts у БД), що містить унікальний ідентифікатор, логін (ім’я користувача або email) та хеш пароля. Паролі зберігаються не в відкритому вигляді, а у вигляді криптографічних хешів, що унеможливлює їх відновлення зловмисником у разі компрометації бази. Процес автентифікації відбувається наступним чином: користувач вводить свої облікові дані (логін і пароль) на захищеній сторінці входу; сервер звіряє отриманий пароль (після хешування) з збереженим хешем у базі. У разі успішного збігу користувач вважається автентифікованим.

Після успішної автентифікації сервер генерує для клієнта JWT–токен. JSON Web Token (JWT) – це стандарт токена доступу на основі JSON, визначений у RFC 7519, що зазвичай використовується для передачі даних аутентифікації у клієнт-серверних застосунках [26]. JWT являє собою зашифрований або підписаний JSON, який містить твердження (claims) – певні дані про користувача і сеанс. Зокрема, у нашій системі до складу токена включаються принаймні такі claims: унікальний ID користувача, його ім’я (або

логін) та роль користувача. Токен підписується сервером (використовується алгоритм HMAC SHA256 з секретним ключем [27]) і повертається клієнту. Відтоді JWT слугує маркером користувача для всіх подальших запитів: клієнт додає цей токен до заголовків HTTP (Authorization: Bearer ...) при зверненні до захищених ресурсів сервера. Сервер, отримавши запит, перевіряє підпис токена (автентичність) і читає claims, щоб ідентифікувати користувача та його права.

На стороні клієнта веб-застосунку токен JWT зберігається у localStorage браузера. Зберігання токена в localStorage означає, що він існує в пам’яті браузера навіть після закриття вкладки, що дозволяє не перезаходити в акаунт кожен раз коли відкриваєш сайт (див. рис. 3.18). Важливо, що токен передається тільки по захищеному протоколу HTTPS, що унеможливлює його перехоплення в мережі у відкритому вигляді. Таким чином, використання JWT забезпечує stateless-автентифікацію – сервер не зберігає інформацію про сеанс у пам’яті, а довіряє пред’явленому токену. Це добре масштабується: будь-який інстанс сервера або мікросервіс (у майбутньому) зможе перевірити токен і обслужити запит, маючи спільний секрет для валідації.

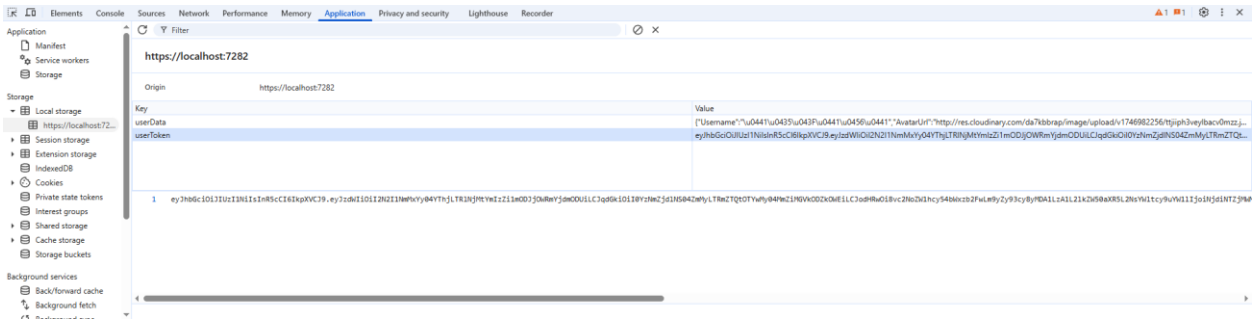


Рисунок 3.18 – Вигляд даних в localStorage

Джерело: зроблено автором на основі виконаної розробки

Авторизація та розмежування доступу. Після ідентифікації користувача система застосовує рольову модель доступу. Кожен обліковий запис має одну або кілька ролей, які визначають його повноваження. У JWT токені ролі користувача містяться у вигляді claim типу

http://schemas.microsoft.com/ws/2008/06/identity/claims/role – це стандартний URI для ролей в .NET [28]. Наявність відповідних claims дозволяє фреймворку ASP.NET Core автоматично здійснювати авторизацію на основі ролей.

У системі передбачено такі ролі користувачів: незареєстрований користувач, турист, гід та адміністратор. Незареєстровані (анонімні) відвідувачі мають мінімальні права – вони можуть переглядати загальнодоступну інформацію (наприклад, список турів, опис), але не можуть здійснювати привілейовані дії (бронювання, додавання контенту). Роль “турист” надається автентифікованим звичайним користувачам системи – туристи можуть бронювати тури, залишати відгуки та отримувати персоналізовані рекомендації. Роль “гід” має користувач, який пропонує власні тури: гіди володіють додатковими можливостями, такими як створення нових турів, редагування опису турів, перегляд списку власних екскурсій тощо. На рисунку нижче продемонстровано приклад інтерфейсу для гіда – у меню присутній розділ управління своїми турами, якого нема у туриста (див. рис. 3.19). Адміністратор системи — це окрема роль з найвищим рівнем доступу. Адміни можуть виконувати всі функції, доступні іншим ролям, а також мають доступ до адміністративної панелі: керування користувачами та їх ролями, модерація контенту, редагування довідникових даних (списку локацій, мов інтерфейсу), перегляд системних логів тощо. Адміністраторські сторінки захищені ролями, тому для звичайних користувачів вони недоступні.

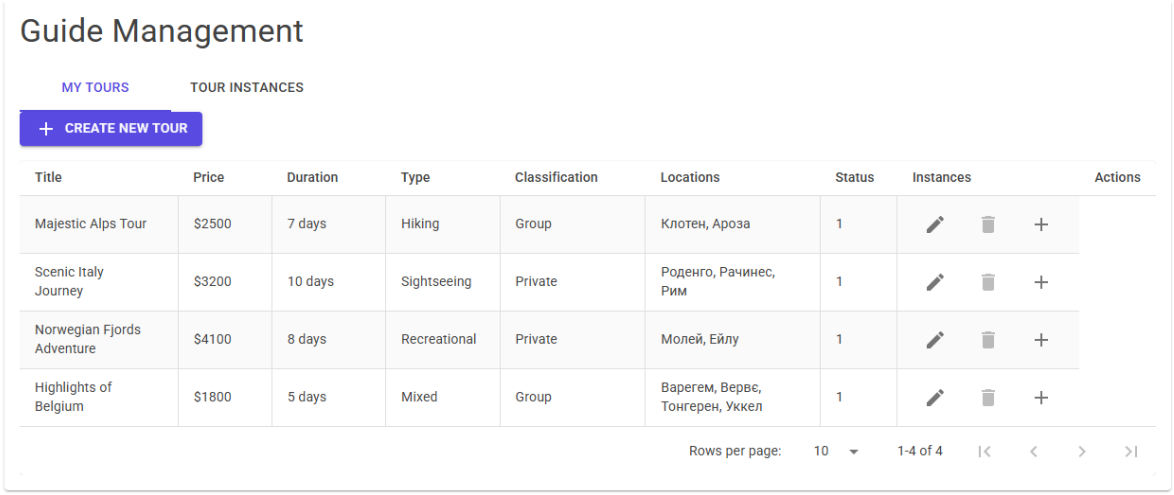


Рисунок 3.19 – Меню редагування турів

Джерело: зроблено автором на основі виконаної розробки

Безпека з’єднань. Усі чутливі взаємодії в системі здійснюються по захищеному протоколу HTTPS. В продакшн-середовищі сервер має встановлений SSL-сертифікат, що шифрує трафік між клієнтом і сервером. Використання HTTPS гарантує конфіденційність даних (наприклад, паролі при вході або JWT токени при передачі) та захист від «прослуховування» трафіку. Крім того, впроваджено й інші базові заходи безпеки: перевірка введених даних на стороні сервера для запобігання SQL-ін’єкціям і XSS, обмеження доступу до функціоналу згідно з ролями та логування ключових подій. Сукупність цих рішень забезпечує належний рівень безпеки та контроль доступу в системі, відповідаючи вимогам дипломного проекту.

3.3.3 Інструментальні засоби розробки для створення прикладного програмного забезпечення інформаційних управляючих систем.

Для реалізації ІУС обрано сучасний стек технологій .NET з єдиною мовою програмування C# на клієнті й сервері. Архітектура системи клієнт-серверна: фронтенд-частина побудована на основі фреймворку Blazor (WebAssembly), а серверна частина – у вигляді веб-API на платформі ASP.NET Core (C#). Використання Blazor WebAssembly у зв'язці з ASP.NET Core забезпечує повноцінну розробку «клієнт + сервер» засобами .NET із можливістю спільно використовувати код між клієнтським та серверним додатками [29]. Зокрема, створено спільну бібліотеку Common з моделями даних і DTO-об'єктами, яка підключається до обох частин, забезпечуючи єдину модель даних для обміну інформацією. Фронтенд реалізовано у вигляді інтерактивного інтерфейсу Blazor, що використовує адаптивну верстку для різних пристроїв, і взаємодіє з бекендом через HTTP-запити до API (формат даних – JSON, серіалізація/десеріалізація відбувається автоматично на основі спільних моделей). Серверна частина на ASP.NET Core побудована за принципами MVC (Model-View-Controller) – тут контролери Web API виступають посередниками між даними та інтерфейсом, отримуючи запити від клієнта та повертаючи результати у форматі JSON.

Для зберігання даних застосовано реляційну СУБД PostgreSQL. Доступ до БД реалізовано через технологію об'єктно-реляційного відображення Entity Framework Core (EF Core) з використанням провайдера Npgsql. Це дозволяє розробникам маніпулювати даними за допомогою об'єктних моделей (ORM-моделей), що відображають таблиці БД, та виконувати запити мовою LINQ. EF Core є сучасним крос-платформним ORM, що значно спрощує доступ до БД та підтримує роботу з різними типами СУБД, автоматичні міграції схеми і відстеження змін у даних [30]. Провайдер Npgsql повністю інтегрує PostgreSQL в EF Core, діючи аналогічно іншим провайдерам (наприклад, для SQL Server) [31]. Таким чином, бізнес-логіка додатку ізольована від специфіки

SQL-запитів – розробник працює з даними на рівні C# об’єктів, а EF Core транслює операції у SQL для PostgreSQL. Це підвищує продуктивність розробки та забезпечує незалежність від конкретної СУБД.

Для підвищення швидкодії застосунку використовується кешування даних за допомогою Redis. Redis – це популярне відкрито-кодове рішення для кешування в пам’яті, що славиться високою швидкістю доступу до даних. Його основне призначення – зберігання часто запитуваної інформації з метою зменшення навантаження на основну базу даних та прискорення роботи додатка [32]. У даному проекті Redis використовується, зокрема, для кешування локалізацій, що дозволяє швидко перемикати мови без повторних звернень до БД.

Для зберігання та обробки зображень інтегровано хмарний сервіс Cloudinary. Він надає REST API та офіційну .NET SDK-бібліотеку для завантаження зображень, їх трансформації та доставляння через CDN. Cloudinary відомий простотою інтеграції та багатим API, а також наявністю безкоштовного тарифу з достатньою квотою для початкових потреб проекту [33]. В нашій ІУС Cloudinary використовується для зберігання завантажених користувачами фотографій та інших медіа-файлів: серверна частина надсилає зображення до хмарного сховища, отримуючи натомість URL для подальшого відображення цих зображень у клієнтському інтерфейсі.

Ще однією особливістю проекту є задіяння API штучного інтелекту OpenAI для формування рекомендацій. Зокрема, інтегровано виклики до моделі ChatGPT (через OpenAI API) для генерування текстових підказок та персоналізованих рекомендацій у системі. OpenAI API надає потужний інструмент для генерування людиноподібного тексту та рекомендацій на основі ШІ [34]. У контексті ІУС це означає, що система може аналізувати вподобання або попередні дії користувача і генерувати рекомендації у текстовій формі. Використання хмарного AI-сервісу позбавляє необхідності розгортати власні моделі машинного навчання, надаючи готові високоякісні результати через прості виклики API.

Конфігурація проекту централізована за допомогою файлів налаштувань appsettings.json. Для розмежування середовищ (розробка, тестування, продуктив) використовуються окремі файли appsettings.Development.json, appsettings.Production.json тощо, що автоматично підвантажуються .NET середовищем залежно від поточного середовища виконання [35]. Такий підхід відповідає прийнятим практикам ASP.NET Core і полегшує керування секретами та параметрами (наприклад, рядками підключення до БД, ключами API тощо) без зміни коду.

У проекті реалізовано модульне тестування з використанням фреймворку xUnit (проєкт Tests). xUnit – один із найбільш популярних фреймворків для unit-тестування в .NET, безкоштовний і з активною спільнотою [36]. Набір автоматичних тестів дозволяє перевіряти правильність роботи ключових методів бекенду при кожній зміні коду, що забезпечує стабільність і спрощує рефакторинг.

Для логування подій та помилок використовується бібліотека Serilog – сучасний інструмент для структурованого логування у .NET. Serilog відомий як один з найпопулярніших та найшвидших логувальних фреймворків для .NET, що підтримує зберігання логів у різноманітні сховища (файли, бази даних, хмарні сервіси тощо) та гнучке конфігурування форматів [37]. У даному проекті Serilog налаштовано на запис логів у консоль і файли з використанням структурованого форматування (JSON), що полегшує їх подальший аналіз. Логи містять технічну інформацію про виконання запитів, винятки та інші події, допомагаючи у відстеженні роботи системи та налагодженні.

Розробка ІУС на .NET здійснювалась із дотриманням перевірених архітектурних патернів та принципів проектування. Використано патерн MVC (Model-View-Controller) для розділення відповідальностей: моделі даних зосереджено у спільній бібліотеці Common, представлення (View) реалізовано у вигляді компонентів інтерфейсу Blazor, а контролери (Controller) – у бекенді ASP.NET Core Web API, які опрацьовують HTTP-запити і формують відповіді.

Така багатошарова структура сприяє підтримуваності коду та спрощує розвиток системи.

Іншим ключовим патерном є Dependency Injection (DI) – принцип впровадження залежностей. У .NET Core наявний вбудований механізм DI, який активно використаний у проекті: усі службові компоненти (сервіси, репозиторії тощо) реєструються в колекції залежностей і автоматично впроваджуються у потрібні контролери чи інші служби. Це підвищує гнучкість системи, полегшує тестування (можна заміняти реалізації інтерфейсів на мок-об’єкти) та зменшує зв’язність між компонентами.

Проектування і кодинг здійснювались згідно з принципами DRY (Don’t Repeat Yourself) та KISS (Keep It Simple, Stupid). Принцип DRY означає уникнення дублювання логіки – кожен фрагмент знань у системі має єдине представлення, що знижує кількість помилок і спрощує підтримку. Принцип KISS вимагає максимально спрощувати реалізацію – код повинен бути зрозумілим і читабельним, без зайвої складності [38]. Дотримання цих підходів сприяло створенню чистого, підтримуваного коду, зручного для подальшого розширення.

Крім того, у компонентах логіки застосовано патерн «Result» для обробки результатів виконання операцій. Суть його полягає в тому, що методи повертають спеціальний об’єкт-результат , який містить або успішне значення, або інформацію про помилку. Це дозволяє явно обробляти помилки без використання виключень, підвищуючи надійність і прозорість роботи сервісів. Такий підхід став популярним у .NET спільноті останнім часом [39], оскільки допомагає чітко розмежувати успішні та неуспішні результати викликів і полегшує відлагодження. Зведена характеристика інструментальних засобів.

3.3.4 Тестування програмного забезпечення

Метою тестування розробленого програмного забезпечення є перевірка коректності реалізації функціоналу, виявлення помилок та дефектів, а також забезпечення стабільності, надійності та якості роботи системи. У межах проєкту було реалізовано модульне тестування (unit testing), яке дозволяє локально перевірити роботу окремих частин логіки застосунку.

Модульне тестування реалізовано у вигляді окремого проєкту під назвою Tests, розміщеного в структурі рішення, вигляд фрагменту тесту показано на рисунку (див. рис. 3.20). Для написання тестів використовувався популярний фреймворк xUnit, а також бібліотека FluentAssertions для зручної перевірки результатів. Тестам піддавалися насамперед сервіси (наприклад, BookingService, GuideService), які містять бізнес-логіку, а також окремі утилітарні методи (див. рис. 3.21).

```
[Fact]
0 references
public async Task GetAllTours_WithPriceFilter_ReturnsFilteredTours()
{
    // Arrange
    var filters = new TourFiltersDto
    {
        FromPrice = 120.00m,
        ToPrice = 200.00m,
        ToDurationDays = int.MaxValue
    };
    var page = 1;
    var pageSize = 10;

    // Act
    var result = await _service.GetAllTours(page, pageSize, filters, CancellationToken.None);

    // Assert
    Assert.True(result.TryGetValue(out var data :PagedResult<TourDto>));
    Assert.NotNull(data);
    Assert.Equal(1, data.TotalCount); // Only the Nature Adventure tour
    Assert.Single(data.Items);
    Assert.Equal("Nature Adventure", data.Items[0].Title);
}
```

Рисунок 3.20 – Фрагмент коду одного з юніт-тестів

Джерело: зроблено автором на основі виконаної розробки

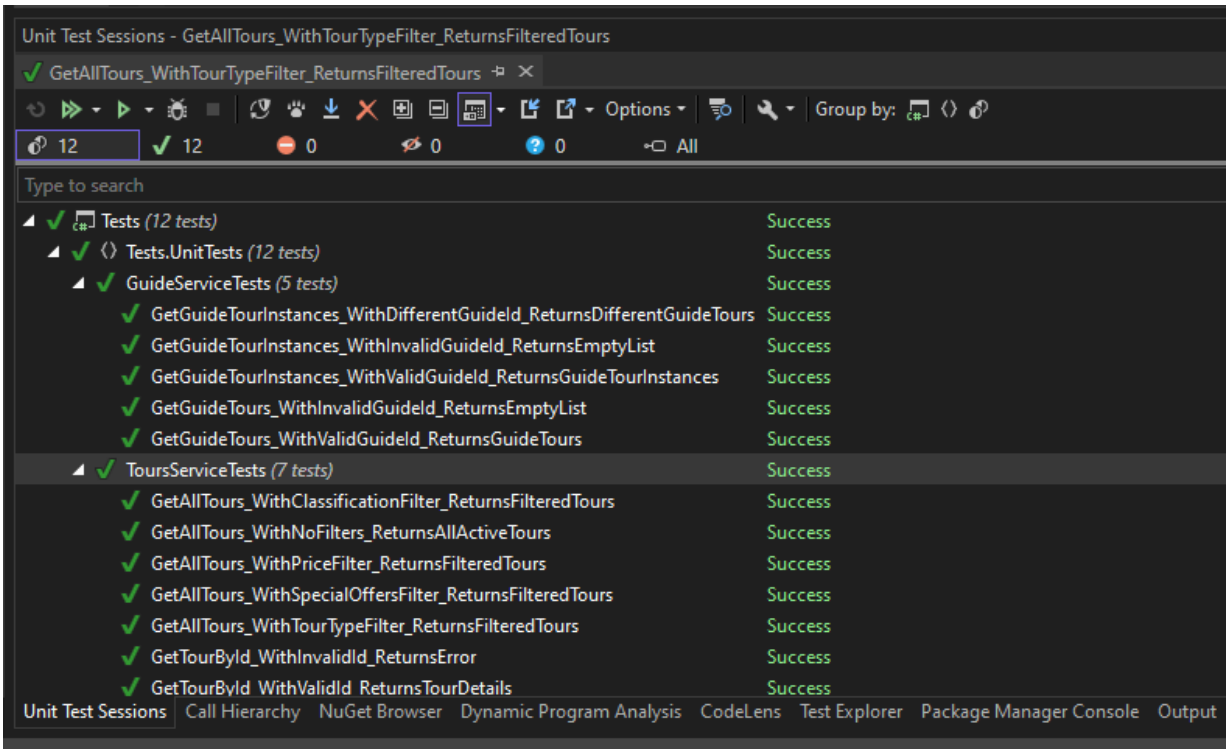


Рисунок 3.21 – Результат запуску юніт-тестів

Джерело: зроблено автором на основі виконаної розробки

Інтеграційне тестування передбачалося як перевірка взаємодії між окремими модулями – наприклад, між контролером, сервісом і базою даних. У рамках проєкту такі перевірки проводилися вручну під час розробки з використанням Swagger UI, Postman та інтерфейсу користувача Blazor (див. рис.3.22).

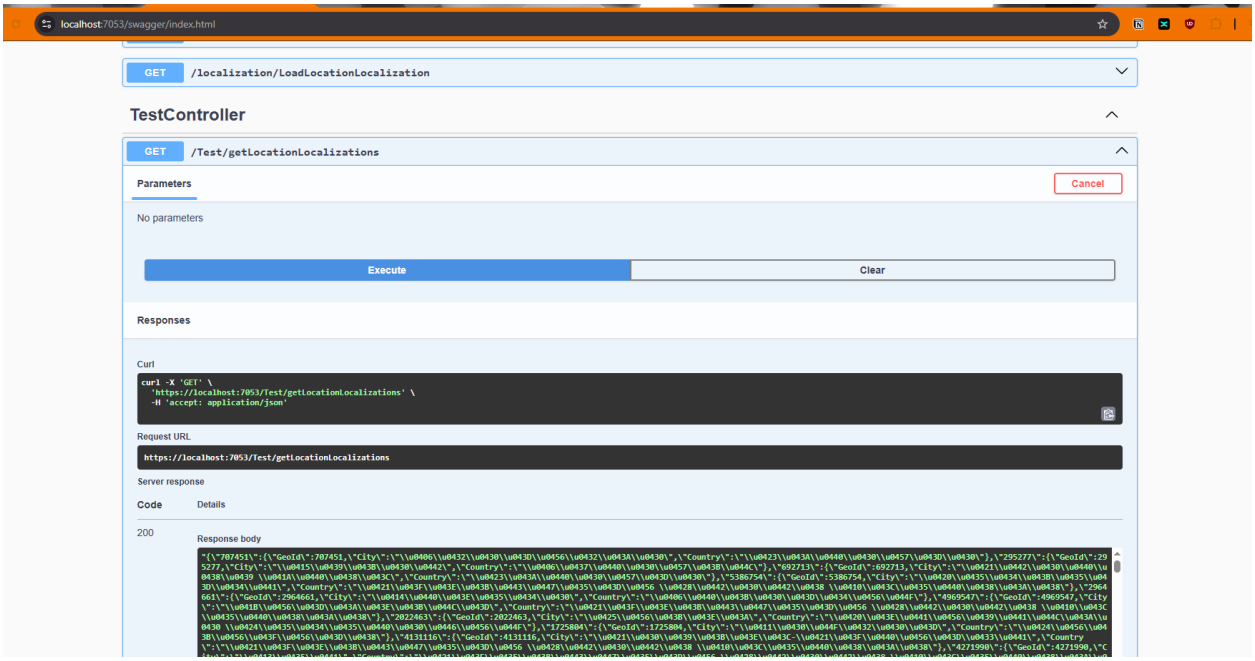


Рисунок 3.22 – Тестування ендпоінта через Swagger UI

Джерело: зроблено автором на основі виконаної розробки

Повна перевірка працездатності застосунку проводилась у середовищі розробки, шляхом запуску всіх компонентів через Visual Studio. Оскільки система використовує реальні сервіси (Redis, PostgreSQL у Docker, зовнішні API), фактично було здійснено системну перевірку функціонування у максимально наближених до продакшен-умов.

3.3.5 Керівництво (інструкція) користувача

Цей розділ містить рекомендації щодо локального запуску та налаштування проєкту для розробників. Система побудована з використанням технологій .NET (ASP.NET Core) з базою даних PostgreSQL і кешуванням на Redis. Нижче наведено кроки для розгортання проєкту на локальній машині розробника:

Підготовка середовища: Переконайтеся, що на комп'ютері встановлено Visual Studio 2022 або інша IDE, придатна для запуску .NET Core проєктів. Також встановіть Docker Desktop, оскільки проєкт використовує Docker-контейнери для служб PostgreSQL та Redis. Переконайтеся, що Docker запущений і працює коректно.

Клонування репозиторію та відкриття проєкту: Отримайте вихідний код проєкту. Відкрийте файл рішення (.sln) у Visual Studio. Переконайтеся, що всі необхідні проєктні залежності (NuGet-пакети) відновлені – IDE зазвичай зробить це автоматично при відкритті рішення з підключенням до інтернету.

Налаштування конфігурації: У каталозі проєкту знайдіть файли конфігурації appsettings.json та appsettings.Development.json. Відкрийте appsettings.Development.json. Перевірте налаштування підключення до бази даних та кешу. Зазвичай там міститься рядок підключення до PostgreSQL (Host, Port, Database, User, Password) та параметри Redis. За замовчуванням, у проекті могли бути використані стандартні значення (наприклад, хост localhost і порт 5432 для PostgreSQL, 6379 для Redis). В разі потреби змініть ці налаштування відповідно до вашого локального середовища або конфігурації Docker. Якщо PostgreSQL і Redis працюватимуть у контейнерах Docker на локальній машині, як хост можна вказати localhost або назви контейнерів, якщо використовуєте Docker Compose з мережевим оточенням.

Запуск залежностей (PostgreSQL, Redis): Проект передбачає, що база даних PostgreSQL та сервер Redis доступні при запуску. Найзручніше скористатися Docker-контейнерами. В терміналі перейдіть до директорії

проєкту (де, можливо, розташований файл docker-compose.yml). Якщо docker-compose файл наявний і налаштований, виконайте команду docker-compose up -d – це завантажить необхідні образи (наприклад, postgres та redis) і запустить контейнери у фоновому режимі. У разі відсутності docker-compose файлу, ви можете запустити контейнери вручну командами:

docker run --name tour-postgres -e POSTGRES_USER=postgres -e POSTGRES_PASSWORD=postgres -e POSTGRES_DB=TourDB -p 5432:5432 -d postgres:latest;

docker run --name tour-redis -p 6379:6379 -d redis:latest.

Переконайтеся, що контейнери запустилися успішно (командою docker ps можна перевірити активні контейнери). База даних PostgreSQL буде порожньою при першому запуску, ви маєте виконати усі міграції в базі даних для правильної роботи додатку.

Запуск застосунку через Visual Studio: В середовищі Visual Studio оберіть профіль запуску. Переконайтеся, що вибрано конфігурацію Debug і середовище Development. Натисніть Start (F5) для запуску програми. Visual Studio збілдить проект і запустить веб-сервер..

Перевірка роботи застосунку: Після успішного запуску Visual Studio відкриє браузер з дашбордом Aspire, де можна перейти до самого клієнта. Ви повинні побачити головну сторінку веб-застосунку. Перевірте основні функції: відкрийте сторінку реєстрації, спробуйте створити тестового користувача, перегляньте список турів. Якщо всі компоненти налаштовані правильно, ви зможете зареєструвати нового туриста, увійти під ним і здійснити тестове бронювання. В базі даних з’являться відповідні записи, а Redis може почати кешувати запити.

3.4. Технічне забезпечення

3.4.1 Обґрунтування вибору технічного забезпечення

Для розгортання веборієнтованої інформаційної системи на .NET Core з базою даних PostgreSQL доступні кілька основних хмарних платформ. Найпопулярніші варіанти – Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP). Кожна з цих платформ підтримує хостинг ASP.NET Core додатків і використання PostgreSQL, проте відрізняється підходами до розгортання й набором сервісів:

AWS (Amazon Web Services) – пропонує найбільшу екосистему хмарних сервісів для застосунків [40]. Для розгортання .NET Core застосунків AWS надає інфраструктурний підхід: EC2 (Elastic Compute Cloud) – віртуальні машини (IaaS), на яких можна розгорнути веб-додаток (.NET Core) під Linux або Windows. Також є керовані сервіси, зокрема Amazon RDS для PostgreSQL та Amazon ElastiCache для Redis, що полегшують обслуговування бази даних і кешу. AWS відомий надійністю, масштабованістю та гнучкістю налаштування – ця платформа масштабована, надійна та підтримує широкий вибір ОС (Linux, Windows тощо) [41]. Водночас конфігурація може бути складнішою через величезну кількість опцій і моделей ціноутворення.

Microsoft Azure – платформа від Microsoft, тісно інтегрована з технологіями .NET. Azure підтримує розгортання .NET Core застосунків у кілька способів; зокрема, через Azure App Service – платформу як сервіс (PaaS) для веб-додатків. App Service дозволяє розгорнути ASP.NET Core застосунок без керування сервером – розробник деплойть код, а всіма оновленнями ОС і платформними налаштуваннями керує Azure. Для PostgreSQL Azure надає керовану службу Azure Database for PostgreSQL, а для кешування – Azure Cache for Redis. Azure відзначається відмінною інтеграцією з продуктами Microsoft і орієнтованістю на корпоративний сегмент. Платформа має дещо менше типів VM, ніж AWS, але також глобально розгорнута по багатьох регіонах. PaaS-можливості Azure вважаються дуже сильними – надаються

вбудовані інструменти для розробки, розгортання, CI/CD, що дозволяє швидко створювати нові хмарні служби [42]. З іншого боку, за таку зручність Azure обмежує прямий контроль: наприклад, виконання власних фонових сервісів або встановлення специфічного ПЗ на App Service неможливе без окремих сервісів. Вартість хостингу в Azure конкурентна, хоча для Linux-навантажень ціна може бути дещо вищою.

Google Cloud Platform (GCP) – пропонує підтримку .NET Core через Google Compute Engine (віртуальні машини IaaS) або керовані платформи App Engine і Cloud Run (контейнерний PaaS) для контейнеризованих .NET-додатків. Для PostgreSQL доступний керований сервіс Cloud SQL for PostgreSQL, для Redis – сервіс Memorystore. GCP відомий інноваційними рішеннями в сфері великих даних, машинного навчання та зручним дрібним білінгом. Інфраструктурно GCP має трохи менше глобальних регіонів і спеціалізованих сервісів, ніж AWS/Azure, і частка ринку GCP поки нижча. Тим не менш, GCP забезпечує повний набір необхідних функцій для хостингу .NET Core + PostgreSQL: підтримує Windows і Linux VM, інтегрується з інструментами DevOps Google, а керовані БД і кеш спрощують розгортання. Цю платформу часто обирають, якщо пріоритетом є аналітика даних або мультихмарна стратегія, а для суто .NET-проєктів GCP використовується рідше (через меншу орієнтованість на Microsoft-екосистему).

Таблиця 3.11 – Порівняльна характеристика AWS EC2, Azure App Service та GCP Compute Engine:

Характеристика	**AWS EC2 (IaaS, віртуальна машина)**	**Azure App Service (PaaS, веб-сервіс)**	**Google Compute Engine (IaaS, віртуальна машина)**
Тип платформи	Інфраструктура як сервіс: повноцінна VM з повним контролем ОС та середовища.	Платформа як сервіс: кероване середовище для веб-додатків.	Інфраструктура як сервіс: VM з повним доступом до ОС та налаштувань.
Розгортання .NET Core	Встановлення .NET Runtime/SDK на вибрану ОС (Linux/Windows) вручну або	Вбудована підтримка ASP.NET Core: платформа має	Розгортання як на звичайному сервері: користувач розгортає .NET

Характеристика	AWS EC2 (IaaS, віртуальна машина)	Azure App Service (PaaS, веб-сервіс)	Google Compute Engine (IaaS, віртуальна машина)
	використання готових AMI з .NET. Повний контроль над конфігурацією середовища.	готове середовище для .NET-додатка.	Core додаток на VM, самостійно встановивши необхідний .NET Runtime або використовуючи контейнер/образ VM з .NET.
Підтримка PostgreSQL	Можливість встановити PostgreSQL на ту ж VM або використовувати Amazon RDS for PostgreSQL – повністю керовану базу даних.	Локально база даних на App Service не підтримується – використовується окремий керований сервіс Azure Database for PostgreSQL або розгортання PostgreSQL в контейнері окремо.	Можна встановити PostgreSQL і Redis безпосередньо на VM
Масштабування	Горизонтальне: автоматичне додавання/видалення інстансів за метриками. Вертикальне: зміна типу інстансу.Балансування навантаження реалізується через Elastic Load Balancer при масштабуванні на кілька екземплярів.	Горизонтальне: App Service Plan може автоматично масштабуватися. Вертикальне: переключення на інший тарифний план (SKU) з більшими ресурсами.	Горизонтальне: налаштування Managed Instance Groups з автоскейлером – Вертикальне: зміна типу машини. Балансувальник навантаження використовується спільно з групами VM.
Керування та гнучкість	Повний адміністративний доступ (SSH/RDP) до сервера. Розробник сам відповідає за оновлення ОС, налаштування безпеки та встановлення потрібного ПЗ.	Повністю кероване середовище: прямих доступів до ОС немає. Гнучкість обмежена рамками платформи	Повний контроль, як і в AWS EC2: адміністратор сам налаштовує ОС, мережу, безпеку. Google надає інструменти для керування VM
Екосистема додаткових сервісів	Надзвичайно широка екосистема AWS. AWS пропонує більше функцій і шаблонів автоматизації для інфраструктури.	Azure тісно інтегрований з екосистемою Microsoft.	GCP має трохи менше готових рішень саме під .NET, проте добре підтримує сучасні підходи.

Джерело: сформовано автором на основі виконаної розробки

На основі наведеного порівняння можна зробити висновок, що AWS EC2 є бажаною платформою для розгортання даної системи (ASP.NET Core Blazor Server + Redis + PostgreSQL). Основні причини вибору AWS EC2:

Гнучкість і контроль. EC2 надає максимальний контроль над середовищем: можна налаштувати сервер під свої потреби, встановити потрібні версії .NET, налаштувати середовище виконання Blazor Server, розгорнути кеш Redis та інші компоненти на власних умовах. Це важливо для нашої системи, оскільки вона складається з кількох частин (веб-додаток, база даних, кеш), які потребують узгодженої конфігурації. На PaaS-платформах (як Azure App Service) розгортання такої комбінованої інфраструктури вимагало б використання декількох окремих сервісів і не дає можливості тримати, наприклад, Redis на тому ж сервері. AWS EC2 же дозволяє як монолітне розгортання (все на одній VM для простоти або економії) так і гнучку мікросервісну архітектуру – за потреби можна розділити веб-сервер, БД і кеш на різні інстанси.

Повнота екосистеми та сумісність. AWS пропонує всі необхідні додаткові сервіси для нашого стеку: керовану базу даних RDS PostgreSQL та кеш ElastiCache Redis, які легко інтегруються з EC2. Це знижує операційне навантаження – наприклад, RDS автоматично робить бекапи і може масштабуватися при зростанні даних [43]. Також AWS має розвинені засоби моніторингу, логування, безпеки, що можна одразу використати. В результаті ми отримуємо єдину платформу для всіх частин системи. Крім того, AWS повністю підтримує технології з відкритим кодом та від Microsoft: є офіційні SDK для .NET, плагіни для Visual Studio, тому .NET Core працює на AWS без проблем, і команда розробників не відчуватиме браку інструментів.

Зрілість платформи та функціональність. AWS – лідер хмарного ринку з 2006 року. AWS EC2 має найбільший вибір типів інстансів і конфігурацій під будь-які потреби – від бюджетних t-серій до високопродуктивних c-серій, GPU-інстансів тощо [44]. Це дозволяє точно підібрати сервер під наші вимоги по CPU/RAM. AWS також випереджає конкурента у ряді можливостей: більше

автоматизації, шаблонів розгортання, готових образів та інструментів моніторингу. Фактично, незалежні огляди відзначають, що AWS часто «задає планку» для хмарних сервісів, а Azure і GCP наздоганяють [45]. Обравши EC2, ми отримуємо доступ до всіх цих напрацювань і можливостей, що позитивно вплине на надійність та продуктивність нашої системи.

Масштабованість і продуктивність. Архітектура на AWS EC2 легко масштабується: при зростанні навантаження можна додати більше EC2-інстансів за допомогою Auto Scaling, або збільшити потужність окремого серверу, змінивши тип інстансу. Ці процеси добре автоматизуються і перевірені практикою у AWS. Azure та GCP також мають механізми масштабування, але AWS пропонує дуже гнучкі та зрілі політики, що особливо корисно при непостійних навантаженнях. До того ж, AWS дає можливість вибору оптимізованих інстансів під специфіку застосунку – це гарантує необхідну швидкодію для Blazor Server і низьку латентність для звернень до Redis/PostgreSQL.

Економічні фактори. Вартість використання AWS є конкурентною. AWS і GCP часто пропонують нижчу або рівну ціну за аналогічні ресурси порівняно з Azure (як було показано, Linux VM середнього розміру на AWS коштує дещо менше, ніж на Azure) [46]. Також варто врахувати, що наша .NET Core програма може працювати на Linux – це уникає ліцензійних витрат Windows, і в AWS така конфігурація особливо ефективна. Таким чином, з точки зору ціни/продуктивності AWS є вигідним вибором для заданих вимог.

3.4.2 Ресурсні вимоги до технічного забезпечення

Для успішної реалізації проєкту слід врахувати мінімальні апаратні вимоги та конфігурації для різних середовищ: робочої станції розробника, клієнтських машин користувачів та серверної інфраструктури на AWS.

Робоча станція розробника: Потужний ПК для комфортної розробки та тестування. Рекомендована конфігурація – процесор рівня AMD Ryzen 5 (6-ядерний або аналогічний Intel Core i5/i7), 32 ГБ оперативної пам’яті, швидкий SSD-накопичувач. Такий ПК дозволить одночасно запускати середовище розробки, локальні сервери та інші необхідні інструменти без пригальмовувань. Операційна система – Windows 10/11 (для повної сумісності з .NET та Azure/AWS інструментами) або сучасний дистрибутив Linux, якщо розробка ведеться кросплатформено. Важливим є наявність підтримки апаратної віртуалізації (для запуску Docker, віртуальних машин). Зазначена конфігурація забезпечує запас продуктивності для збірки великих рішень, запуску кількох проектів одночасно і емулювання різних середовищ.

Клієнтська машина користувача: Кінцеві користувачі взаємодіятимуть із системою через веб-браузер, тому вимоги до клієнтських ПК відносно невисокі. Необхідний сучасний браузер з підтримкою JavaScript і WebAssembly. Браузер має підтримувати WebAssembly, оскільки інтерфейс на Blazor потребує цієї технології для швидкої роботи в клієнті. Операційна система – не нижче Windows 7 (або сучасні версії Linux, macOS); фактично всі ОС, що підтримують актуальні браузери, підійдуть. Оперативна пам’ять клієнта – від 4 ГБ і більше, щоб браузер міг комфортно працювати з веб-застосунком. Зазвичай, будь-який офісний комп’ютер чи ноутбук останнього десятиліття відповідає цим вимогам. Також потрібне стабільне мережеве з’єднання для роботи веб-додатку.

Серверне середовище (AWS): Для початкового розгортання системи пропонується базова конфігурація на AWS. Мінімальні ресурси для серверу, що виконуватиме роль хоста Blazor Server та контейнера для Redis і

PostgreSQL, – це 2 віртуальних CPU та 4 ГБ RAM. На практиці це відповідає, наприклад, типу інстансу t3.medium в AWS EC2 (2 vCPU, ~4 ГБ пам’яті). Така конфігурація здатна обслуговувати веб-додаток зі середнім навантаженням і декількома десятками одночасних користувачів. Якщо планується невелике тестове розгортання, можливе використання навіть t3.small (2 vCPU, 2 ГБ), але для продуктивної системи 4 ГБ ОЗП рекомендується, оскільки Blazor Server тримає стан сесій в пам’яті, а PostgreSQL і Redis також потребують ресурсів. Диск: EC2 інстанс має використовувати SSD-накопичувач об’ємом від ~20 ГБ для швидкого доступу до даних. ОС серверу: обирається 64-розрядна ОС з підтримкою .NET – оптимальним є Linux (Amazon Linux 2 або Ubuntu 20.04 LTS), оскільки .NET Core повністю сумісний з Linux і це зменшує витрати. Встановлюється .NET Runtime/SDK для запуску Blazor Server. На сервері також розгортається служба Redis та PostgreSQL. Для забезпечення продуктивності бажано розділити навантаження: база даних PostgreSQL інтенсивно використовує диск і пам’ять, тому за можливості краще винести PostgreSQL на окремий керований сервіс AWS RDS – стартовий рівень, наприклад, db.t3.micro або db.t3.small для початку. RDS поліпшить надійність та масштабування БД без навантаження на основний EC2. Аналогічно, Redis можна винести на Amazon ElastiCache (cache.t2.micro або аналог) для виробничого середовища. Втім, базова конфігурація передбачає, що всі компоненти можуть працювати на одному EC2 для спрощення: 4 ГБ RAM достатньо, щоб тримати декілька сотень МБ для Redis кешу та декілька гігабайт для процесу PostgreSQL. Мережа і масштабування: EC2 інстанс розгортається в приватній VPC з налаштованими Security Groups. У базовій конфігурації достатньо одного інстансу без балансування, але для відмовостійкості на майбутнє слід передбачити можливість додавання другого екземпляра EC2 і балансувальник Elastic Load Balancer, а також перехід на Multi-AZ для бази даних. На початковому етапі це не обов’язково, але апаратні ресурси і архітектура вибрані з урахуванням потенційного росту.

3.5. Реалізація інформаційної управляючої системи

У результаті виконання дипломного проекту створено інформаційну систему туристичних рекомендацій, що об’єднує сучасні технології для забезпечення персоналізованого сервісу користувачам. Система генерує рекомендації маршрутів та туристичних об’єктів на основі аналізу профілю користувача і наявних даних про напрями подорожей. Ключовими технологіями реалізації є OpenAI API для побудови рекомендацій на базі великих мовних моделей, PostgreSQL для зберігання даних і Redis для кешування популярних запитів та локацій. Фронтенд системи розроблено з використанням Blazor, що забезпечує сучасний інтерактивний інтерфейс, а адміністративний модуль із власною рольовою моделлю дозволяє гнучко керувати правами доступу користувачів. Серед основних функціональних характеристик реалізованої системи виділяють:

– генерація персоналізованих рекомендацій за допомогою OpenAI API з урахуванням вподобань та поведінки користувача;

– надійне зберігання даних (маршрути, профілі користувачів, уподобання) у реляційній базі даних PostgreSQL;

– використання кешування через Redis для швидкого доступу до популярних даних (наприклад, інформації про локації та результати рекомендацій);

– інтуїтивний клієнтський інтерфейс на основі Blazor для зручної взаємодії користувачів із сервісом;

– кастомна рольова модель адміністрації, що дозволяє налаштовувати різні рівні доступу та операцій для співробітників.

Унікальність розробленого проекту полягає в комплексному поєднанні сучасного технічного стеку (Blazor + Redis + PostgreSQL) та інноваційних методів формування рекомендацій. Застосування великих мовних моделей (OpenAI API) підвищує якість персоналізації сервісу, а продумана система кешування зменшує затримки під час запитів. Відповідно до тез наукової

доповіді автора, інтеграція LLM дозволяє досягти високої релевантності рекомендацій, а легковагова архітектура робить рішення придатним для використання у невеликих турагенціях. Основні переваги системи включають:

– суміщення потужного функціоналу з простою підтримкою завдяки відкритому стеку технологій;

– гнучка рольова модель адміністрації, що підвищує безпеку та адаптивність до різних сценаріїв використання;

– модульна архітектура і невелика кількість залежностей, що спрощують подальший супровід та розвиток проекту;

– висока продуктивність за рахунок кешування та оптимізованих алгоритмів, що дозволяє ефективно обслуговувати користувацькі запити в реальному часі.

Економічна ефективність проекту обумовлена низкою факторів. По-перше, використання повністю відкритого програмного забезпечення (Blazor, Redis, PostgreSQL) виключає ліцензійні витрати. По-друге, проста структура системи та зрозумілий код знижують витрати на технічну підтримку і обслуговування. По-третє, систему можна розгорнути на недорогому обладнанні або у хмарному середовищі, що мінімізує необхідні інвестиції в інфраструктуру. Загалом, з точки зору економії ресурсів, проект забезпечує низьку вартість запуску та експлуатації. Наразі створена система не використовується у промислових умовах, але має значний потенціал для впровадження. Легка вага і гнучкість рішення роблять його придатним для малих туристичних агенцій, які можуть підвищити свою конкурентоздатність завдяки персоналізованому підходу до клієнтів. Рекомендується провести пілотне впровадження системи з використанням реальних даних для перевірки отриманих рекомендацій та налаштування алгоритмів. Практична цінність проекту полягає в автоматизації процесу створення рекомендацій, що дозволяє покращити якість обслуговування туристів та зменшити навантаження на співробітників агентства.

Перспективи розвитку та впровадження:

– інтеграція системи з онлайн-сервісами бронювання квитків та готелів для розширення функціональності;

– впровадження механізмів зворотного зв’язку від користувачів (відгуків і рейтингів) для адаптивного вдосконалення рекомендаційних алгоритмів та їх використання для фільтрації контенту;

– розширення мовної підтримки та локалізації інтерфейсу для обслуговування клієнтів з різних регіонів;

– оптимізація продуктивності за допомогою оновлення стратегій кешування та можливого використання додаткових аналітичних інструментів;

– розробка мобільної версії сервісу або окремого додатку для збільшення доступності системи;

– співпраця з туристичними компаніями та проведення пілотного тестування для збору даних і подальшого вдосконалення системи.

ВИСНОВКИ

У межах виконаної кваліфікаційної магістерської роботи було досягнуто основну мету дослідження — спроєктовано та реалізовано інформаційну управляючу систему (ІУС) для туристичної агенції, яка забезпечує підтримку ключових бізнес-процесів, включаючи персоналізовані рекомендації турів, бронювання, оцінювання турів, а також управління акаунтами користувачів.

У ході роботи:

- розроблено повноцінну серверну архітектуру на основі ASP.NET Core з використанням патерну MVC;
- реалізовано інтерфейс користувача на Blazor Server із підтримкою адаптивності;
- побудовано логіку інтеграції з OpenAI API для генерації персоналізованих туристичних рекомендацій;
- реалізовано базу знань, яка поєднує динамічну генерацію знань (через AI) та структуровані дані з реляційної бази PostgreSQL;
- забезпечено високошвидкісний доступ до даних завдяки кешуванню локалізацій через Redis;
- реалізовано підтримку декількох мов інтерфейсу;
- проведено тестування системи, підтверджено її працездатність і доцільність обраного підходу.

У першому розділі роведено аналіз предметної області, досліджено сучасні ІУС у сфері туризму. Обґрунтовано необхідність створення власної системи з урахуванням сучасних вимог до персоналізації, швидкодії та гнучкого адміністрування.

У другому розділі представлено структуру та моделі проєктованої системи. Визначено ключові функціональні блоки ІУС, методи збирання, зберігання та обробки даних. Описано архітектуру рекомендаційного модуля.

У третьому розділі розроблено програмне забезпечення, яке поєднує серверну логіку, клієнтську частину, базу даних і систему кешування. Здійснено тестування системи, підготовлено інструкції для користувачів і описано технічні характеристики розгортання.

У процесі роботи було реалізовано ряд оригінальних рішень, серед яких інтеграція OpenAI API для рекомендацій, кешування багатомовних локалізацій через Redis, спільне використання DTO між сервером і клієнтом завдяки загальній бібліотеці, а також побудова кастомної рольової системи.

Система демонструє високу гнучкість і масштабованість, що дозволяє адаптувати її для малих або середніх туристичних компаній. Хоча проєкт наразі не впроваджено у виробництво, його структура та реалізація свідчать про готовність до використання за мінімальних доопрацювань.

Можливими напрямами подальшого розвитку є:

- впровадження системи у реальних умовах з розширенням функціоналу (наприклад, модуль оплати);
- перехід до мікросервісної архітектури;
- поглиблення рекомендаційного модуля із залученням машинного навчання на власних даних;
- розширення мультимовної підтримки та локалізації для інших регіонів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Туризм. *Географія* / С. Кобернік, Р. Коваленко. 2015.
2. Феномен глобального туризму. *КНУКІМ*. 09.10.2023. URL: https://fgritb.knukim.edu.ua/home/news/fenomen-hlobalnoho-turyzmu.html (дата звернення: 17.05.2025).
3. Виговський Д. С. ТЕНДЕНЦІЇ ТА ПЕРСПЕКТИВИ РОЗВИТКУ ТУРИЗМУ: УКРАЇНА ТА СВІТ : дис. … канд. політ. наук / Громадська організація «Інститут досліджень соціального капіталу». 2024. 2 с. URL: http://bses.in.ua/journals/2024/86_2024/4.pdf (дата звернення: 17.05.2025).
4. Гурова Д. Д. УКРАЇНСЬКИЙ ТУРИЗМ ПІД ЧАС ВІЙНИ. *Українські студії в європейському контексті. 2024. № 8*, Україна. 2024. С. 144–148. URL: http://obrii.org.ua/usec/storage/article/Gurova_2024_144.pdf (дата звернення: 17.05.2025).
5. Якубовський С. О., Кириченко О. В. СТАН І ПЕРСПЕКТИВИ РОЗВИТКУ ІНДУСТРІЇ ТУРИЗМУ В УКРАЇНІ В УМОВАХ ГЛОБАЛЬНИХ ВИКЛИКІВ. *ЕКОНОМІКА ТА СУСПІЛЬСТВО*. 2024. вип. 67. URL: https://dspace.onu.edu.ua/server/api/core/bitstreams/0e85b93d-76c4-463a-84d3-339196b14a7f/content (дата звернення: 17.05.2025).
6. Popular Indicators. *DataBank*. URL: https://databank.worldbank.org/indicator/NY.GDP.PCAP.CD/1ff4a498/Popular-Indicators (дата звернення: 17.05.2025).
7. Олеськів М. ПУБЛІЧНИЙ ЗВІТ Голови Державного агентства розвитку туризму України. *Google Drive*. 01.01.2024. URL: https://drive.google.com/file/d/1ixkNelbBVc7vyV3buYUyS29sCkvam-Tk/view (дата звернення: 17.05.2025).

8. Ukraine needs nearly $9 billion to rebuild its cultural sites and tourism industry, UN agency says. *AP News*. *World News*. 13.02.2024. URL: https://apnews.com/article/unesco-ukraine-russia-cultural-damage-tourism-0cc4a00b45229d7d85646180d89efb41 (дата звернення: 17.05.2025).
9. Седойкін В. Внесок туристичної галузі України до держбюджету: динаміка та аналіз. *The City*. *Новини*. 11.11.2023. URL: https://thecity.com.ua/vnesok-turystychnoyi-galuzi-ukrayiny-do-derzhbyudzhetu-dynamika-ta-analiz.html (дата звернення: 17.05.2025).
10. Шевченко Ю. Україна заробила 3 мільярди гривень від туризму за 2024 рік. *Перший Бізнесовий Biz*. *Економіка*. 22.01.2025. URL: https://fbc.biz.ua/news/ekonomika-uk/ukrayina-zarobila-3-milyardi-griven-vid-turizmu-za-2024-rik/ (дата звернення: 17.05.2025).
11. Моца А. А., Шевчук С. М., Середа Н. М. ПЕРСПЕКТИВИ ПІСЛЯВОЄННОГО ВІДНОВЛЕННЯ СФЕРИ ТУРИЗМУ В УКРАЇНІ. *ЕКОНОМІКА ТА СУСПІЛЬСТВО*. URL: https://economyandsociety.in.ua/index.php/journal/article/view/1560/1501 (дата звернення: 17.05.2025).
12. Цепенда М. М., Бурка В. Й. ІНФОРМАЦІЙНІ СИСТЕМИ, КОМУНІКАЦІЇ І ТЕХНОЛОГІЇ У ТУРИСТИЧНІЙ ІНДУСТРІЇ. Чернівці : Чернівецький національний університет імені Юрія Федьковича, 2024. URL: https://ecogeo.chnu.edu.ua/media/m2vl1bn0/is.pdf (дата звернення: 17.05.2025).
13. TravelOperations. *TravelOperations*. URL: https://traveloperations.com/ (дата звернення: 17.05.2025).
14. TravelOperations Enterprise. *Microsoft AppSource*. *Apps*. URL: https://appsource.microsoft.com/en-sa/product/dynamics-365-for-operations/traveloperations.travel_operations_enterprise (дата звернення: 17.05.2025).

15. TRAVEL AGENCY SOLUTIONS. *WeTravel*. URL: https://product.wetravel.com/travel-agency-software (дата звернення: 17.05.2025).
16. The one platform your travel business has been waiting for. *Tern*. URL: https://www.tern.travel/ (дата звернення: 17.05.2025).
17. ClientBase. *Capterra*. *Travel Agency Software*. URL: https://www.capterra.com/p/2463/ClientBase (дата звернення: 17.05.2025).
18. Run your travel business like a pro.. *Travefy*. URL: https://travefy.com/ (дата звернення: 17.05.2025).
19. Гриненко В., Романенко Ю. Туризм военного времени. *Youtube*. 23.04.2025. URL: https://www.youtube.com/watch?v=vHcAJxN07Rs (дата звернення: 17.05.2025).
20. Draw.io. *App Diagrams*. URL: https://app.diagrams.net/ (дата звернення: 17.05.2025).
21. Рахманов Т. Е. Інформаційна система формування туристичних маршрутів в Ферганській долині. Модуль формування багатоденного маршруту. *СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ КАФЕДРА КОМП'ЮТЕРНИХ НАУК*. 13.05.2021. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/85763/1/Rakhmanov_bac_rob.pdf (дата звернення: 17.05.2025).
22. Booking.com and OpenAI personalize travel at scale. *OpenAI*. URL: https://openai.com/index/booking-com/ (дата звернення: 17.05.2025).
23. Артеменко О. І., Пасічник В. В., Єгорова В. В. ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В ГАЛУЗІ ТУРИЗМУ. АНАЛІЗ ЗАСТОСУВАНЬ ТА РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ. *ІНФОРМАЦІЙНІ СИСТЕМИ, МЕРЕЖІ ТА ТЕХНОЛОГІЇ*. URL: https://science.lpnu.ua/sites/default/files/journal-

paper/2017/may/2615/814ism2015min-3-22_0.pdf (дата звернення: 17.05.2025).

24. Harris C. Microservices vs. Monolithic architecture. *Atlassian*. URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith (дата звернення: 17.05.2025).
25. Ozkaya M. Microservices Killer: Modular Monolithic Architecture. *Medium. Design Microservices Arc*. 16.02.2023. URL: https://medium.com/design-microservices-architecture-with-patterns/microservices-killer-modular-monolithic-architecture-ac83814f6862 (дата звернення: 17.05.2025).
26. JSON Web Token. *Wikipedia*. 24.05.2024. URL: https://uk.wikipedia.org/wiki/JSON_Web_Token (дата звернення: 17.05.2025).
27. HMACSHA256 Class. *Microsoft Learn*. *API browser*. URL: https://learn.microsoft.com/uk-ua/dotnet/api/system.security.cryptography.hmacsha256?view=net-8.0 (дата звернення: 17.05.2025).
28. The Role of Claims. *Microsoft Learn*. *Identity and access*. 08.04.2025. URL: https://learn.microsoft.com/en-us/windows-server/identity/ad-fs/technical-reference/the-role-of-claims (дата звернення: 17.05.2025).
29. ASP.NET Core Blazor hosting models. *Microsoft Learn*. *ASP.NET Core*. 12.11.2024. URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-9.0 (дата звернення: 17.05.2025).
30. Huffenreuter A. L. 9 Best ORM Solutions for .NET: Comparison Guide. *Devart*. *ADO.NET Data Providers*. 02.01.2025. URL: https://blog.devart.com/best-postgresql-orm-in-dotnet.html (дата звернення: 17.05.2025).
31. Npgsql - .NET Access to PostgreSQL. *Npgsql*. *Home*. URL: https://www.npgsql.org/index.html (дата звернення: 17.05.2025).

32. Emadamerho-Atori N. The Good and the Bad of Redis In-Memory Database. *Altexsoft*. *Blog*. 10.02.2025. URL: https://www.altexsoft.com/blog/redis-pros-and-cons/ (дата звернення: 17.05.2025).
33. Kovač R. Free Image Hosting for .NET Projects. *Medium*. 24.08.2022. URL: https://medium.com/@kova98/free-image-hosting-for-net-6-projects-d166d8767150 (дата звернення: 17.05.2025).
34. Shrivastava A. Integrating the OpenAI API into a C# Project with Visual Studio. *Dev*. 01.10.2023. URL: https://dev.to/amitshri05/integrating-the-openai-api-into-a-c-project-with-visual-studio-3ee8 (дата звернення: 17.05.2025).
35. Configuration in ASP.NET Core. *Microsoft Learn*. *ASP.NET Core*. 31.10.2024. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-9.0 (дата звернення: 17.05.2025).
36. Kovač R. Unit Testing in .NET with xUnit. *Medium*. 23.04.2023. URL: https://medium.com/@kova98/unit-testing-net-7-code-with-xunit-a0cfbca75599 (дата звернення: 17.05.2025).
37. Stanley U. Logging in .NET: A Comparison of the Top 4 Libraries. *Better Stack*. *Guides*. 03.06.2024. URL: https://betterstack.com/community/guides/logging/best-dotnet-logging-libraries/ (дата звернення: 17.05.2025).
38. Сверчков В. Що таке патерни проєктування у програмуванні. *ITVDN*. *Блог*. 29.05.2024. URL: https://itvdn.com/ua/blog/article/what-are-design-patterns-in-programming (дата звернення: 17.05.2025).
39. Lock A. Is the result pattern worth it?. *Andrew Lock*. 29.10.2024. URL: https://andrewlock.net/working-with-the-result-pattern-part-4-is-the-result-pattern-worth-it/ (дата звернення: 17.05.2025).
40. Nisar A. AWS vs. Azure vs. Google Cloud: Cloud Services Compared 2025. *Channel Insider*. *Cloud computing*. 11.10.2024. URL:

https://www.channelinsider.com/cloud-computing/aws-vs-azure-vs-google-cloud (дата звернення: 17.05.2025).

41. admin. Azure vs. AWS vs. Google Cloud: A Comprehensive Comparison. *AnsiByteCode*. *AWS*. 15.04.2025. URL: https://ansibytecode.com/azure-vs-aws-vs-google-cloud-a-comprehensive-comparison/ (дата звернення: 17.05.2025).
42. Drew R. AWS EC2 vs Azure: Cloud Services Comparison. *ServerWatch*. *Virtualization*. 25.03.2022. URL: https://www.serverwatch.com/virtualization/aws-ec2-vs-azure/ (дата звернення: 17.05.2025).
43. AWS vs. Google Cloud vs. Azure: Comparison of Managed PostgreSQL Providers. *Reddit*. *PostgreSQL*. 05.08.2021. URL: https://www.reddit.com/r/PostgreSQL/comments/oykuk9/aws_vs_google_cloud_vs_azure_comparison_of/ (дата звернення: 17.05.2025).
44. AWS EC2 vs Azure Virtual Machines: Top Comparison Guide [2025]. *CCS Learning Academy*. *Cloud computing*. 21.11.2024. URL: https://www.ccslearningacademy.com/aws-ec2-vs-azure-virtual-machines/ (дата звернення: 17.05.2025).
45. Hasura Team. Comparison of Managed PostgreSQL Providers: AWS RDS vs Google Cloud SQL vs Azure PostgreSQL. *Hasura*. *PostgreSQL*. 19.07.2021. URL: https://hasura.io/blog/comparison-of-managed-postgresql-aws-rds-google-cloud-sql-azure-postgresql (дата звернення: 17.05.2025).
46. Rifai M. Cloud comparison: AWS EC2 vs Azure Virtual Machines vs Google Compute Engine. *Pluralsight*. *Cloud*. 08.06.2023. URL: https://www.pluralsight.com/resources/blog/cloud/cloud-comparison-aws-ec2-vs-azure-virtual-machines-vs-google-compute-engine (дата звернення: 17.05.2025).
47. TravelOperations. *LinkedIn*. URL: https://www.linkedin.com/posts/travel-operations_microsoftcopilot-

microsoftdynamics365-activity-7158066807839301632-s43U/ (дата звернення: 17.05.2025).

48. WeTravel - 2025 Pricing, Features, Reviews & Alternatives. *GetApp*. *Reservations*. 01.05.2025. URL: https://www.getapp.com/hospitality-travel-software/a/wetravel/ (дата звернення: 17.05.2025).
49. Travel smart. *Tern*. URL: https://travelwithtern.com/ (дата звернення: 17.05.2025).
50. ClientBase Windows Overview for New Users. *Tres*. *Overview and Profiles*. URL: https://trestechnologieshelp.zendesk.com/hc/en-us/articles/4404267469715-ClientBase-Windows-Overview-for-New-Users (дата звернення: 17.05.2025).

ДОДАТКИ

Додаток А

Паплінський В.В., студент
Київський національний економічний університет імені Вадима Гетьмана
paplinskyi.vladyslav@kneu.ua

ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ДЛЯ ТУРИСТИЧНОЇ ГАЛУЗІ

У сучасних умовах цифрової трансформації та високої конкуренції в туристичній галузі інформаційні системи управління (ІУС) стають критично важливим інструментом для ефективного функціонування туристичних агенцій. З огляду на різноманітність пропозицій на ринку програмного забезпечення та швидкі темпи змін у технологічному середовищі, туристичні підприємства стикаються з низкою викликів під час вибору та впровадження ІУС. До них належать труднощі з адаптацією систем до індивідуальних бізнес-процесів, надмірна вартість впровадження, недостатня кваліфікація персоналу для роботи з новими технологіями, а також ризик придбання програмного забезпечення, яке не відповідає реальним потребам компанії. Особливої актуальності ці питання набувають в умовах кризових явищ, зокрема війни в Україні, яка суттєво вплинула на туристичний сектор, змусивши його переосмислити підходи до управління ресурсами, клієнтським сервісом та логістикою. Ефективна ІУС дозволяє автоматизувати ключові операції, забезпечити аналітичну підтримку управлінських рішень, покращити комунікацію з клієнтами та адаптувати бізнес до мінливих умов ринку.

Мета дослідження полягає у визначенні ролі та ефективності впровадження інформаційних управлінських систем (ІУС) у сучасній туристичній галузі України, особливо в умовах війни та трансформаційного повоєнного періоду.

Основне завдання – виявити ключові переваги ІУС у забезпеченні стабільності роботи туристичних агенцій, їхньої адаптації до умов криз, а також проаналізувати, як автоматизація та цифровізація бізнес-процесів впливають на конкурентоспроможність підприємств галузі. У межах дослідження також передбачено виявлення інших труднощів, пов’язаних з упровадженням ІУС, таких як фінансові витрати, потреба в підготовці персоналу та технічна сумісність із уже наявними системами, а також визначення перспектив розвитку ІУС з урахуванням сучасних цифрових технологій та політичної ситуації.

У сучасних умовах туризм як сфера життя людей зазнає подекуди нерівномірно розподілених впливів явищ геополітики, мікро- і макро-фінансового становища, біологічної ситуації, як до прикладу пандемія COVID-

19, цифровізації, тощо. Взявши у порівняння будь-яку галузь людського життя сучасної людини зі становищем навіть наприкінці 20-го століття, ми бачимо що описані чинники змінили звичний побут, хоч і з різною інтенсивністю, до невпізнання. В умовах України ці чинники набули надзвичайно помітного впливу на суспільство, і серед них особливо сильно відмітили туристичну галузь.

Україна за 21-ше століття зазнала значних змін в туристичній галузі, серед позитивних можна відмітити: покращення фінансового статусу українців, безвізовий режим з Євросоюзом, полегшене отримання закордонного паспорту завдяки реформі ЦНАП. Але, не дивлячись на перелічені позитивні зміни у туристичній сфері України, необхідно зазначити значні негативні зміни: поточна російсько-українська війна, включно з фазою 2014-2022 років, світова пандемія COVID-19, закриття кордонів для чоловіків мобілізаційного віку (18-60 ти років), та багато інших випливаючих факторів. Враховуючи усі ці фактори, можна сказати, що туристична галузь зазнає великих змін в рамках України, та українського суспільства загалом, і, на жаль, здебільшого в гіршу сторону.

Наразі можна стверджувати, що українське суспільство після 2014-го, та особливо після 2022-го року розділилося, як мінімум, на 3 значні частини, щодо яких вищеперелічені фактори вплинули по-різному:

- Українські біженці за кордоном – ця група населення, по різним оцінкам, складає близько 5,2 мільйонів осіб, що виїхали за кордон з лютого 2022-го року. Щодо цієї групи населення заборона перетинання кордонів України не діє, і тому ця група більш схильна до туризму, додатково враховуючи те, що в країні реципієнті немає гострої проблеми з війною. Звичайно, що біженці не є тими людьми, які схильні до надмірностей, як туризм, але навіть біженці, у порівнянні з іншими українцями, мають набагато кращі політичні та фінансові можливості для туризму, здебільного за кордоном України.
- Українські чоловіки призивного віку та жінки невиїздних спеціальностей – ця частина населення зазнала найважчого впливу негативних факторів війни. Більша частина цього прошарку населення намагаються не «відсвічувати» зайвий раз і тому не розглядають туризм взагалі, але все ж залишається менша частина населення, які подорожують, але вони переорієнтувалися на внутрішній туризм по містам (наприклад Київ, Львів, Одеса, тощо) та природі (наприклад Карпати, Трахтемирів, тощо).
- Українські жінки та чоловіки, що мають можливість перетину кордону – це є найменш постраждала група населення з усіх перелічених, хоча навіть вона зазнала значних змін у фінансовому та сімейному становищі. Ця верства населення більш за все схожа на довоєнне українське суспільство через те, що вони не підпадають примусовій мобілізації, та можуть вільно пересуватися країною та

поза її межі, але, не дивлячись на це, попит на туризм впав порівняно з 2021 роком через гірше фінансове становище та політичну нестабільність.

Звичайно, що існує багато інших факторів та прошарків населення, за якими можна виділити людей по туристичному потенціалу, але вони є менш значними частинами і, за виключенням військових, відносяться до однієї з перелічених груп.

Туристична галузь України, окрім своєї традиційної діяльності, співпрацюють з військовими та займаються волонтерською діяльністю, в залежності від ступеню взаємодії з відповідними контрагентами, наприклад плануючи логістику чи розміщення людей в готелях біля зони бойових дій чи біля навчальних полігонів. Такий збільшений спектр спричинений тим, що фінансове становище змушує адаптуватися до нових умов, що робить ігнорування військової складової з діяльності туристичних агенцій розкішшю, яку більшість агенцій малого та середнього розміру не можуть собі дозволити.

Враховуючи великий спектр різноманітних сучасних завдань для туристиичних агенцій України, включаючи ексклюзивний для України військовий фактор, складність необхідної системи роботи з ними зростає, саме тому впровадження інформаційних управлінських систем (ІУС) стає ключовим фактором успішної діяльності туристичних агенцій. ІУС забезпечують комплексну автоматизацію бізнес-процесів, що сприяє підвищенню ефективності управління, покращенню якості обслуговування клієнтів та збільшенню конкурентоспроможності компанії.

Інформаційні управлінські системи в туристичних агенціях відіграють ключову роль у забезпеченні ефективної організації внутрішніх бізнес-процесів, а також у підвищенні рівня сервісу для кінцевого споживача. Вони дозволяють здійснювати автоматизоване управління бронюванням турів, готелів, квитків, обробкою запитів клієнтів, розрахунком вартості, веденням бази даних, аналітикою продажів і керуванням маркетинговими кампаніями. Такі системи забезпечують централізовану структуру зберігання інформації, що дозволяє синхронізувати дані між усіма відділами агентства в реальному часі, зменшуючи ризик помилок і сприяючи оперативному прийняттю рішень.

Велику увагу в сучасних умовах привертає інтеграція з глобальними системами бронювання (GDS), яка відкриває прямий доступ до міжнародних баз постачальників туристичних послуг. Це дозволяє суттєво розширити асортимент пропозицій для клієнтів та оперативно реагувати на зміну попиту. Вбудовані CRM-модулі, які є невід’ємною частиною сучасних ІУС, забезпечують глибоке розуміння поведінки клієнта, дозволяють сегментувати базу споживачів, здійснювати персоналізовані комунікації та підтримувати високу якість обслуговування. У результаті впровадження таких систем туристичні компанії отримують можливість суттєво знижувати операційні витрати, покращувати якість управлінських рішень, а також підвищувати продуктивність працівників за рахунок автоматизації рутинних завдань.

Особливу важливість інформаційні системи отримали в періоди глобальних криз, зокрема під час пандемії або повномасштабного вторгнення Росії в Україну. У ці моменти туристичні компанії були змушені оперативно змінювати маршрути, переносити дати подорожей, здійснювати повернення коштів і зберігати довіру клієнтів.

Отже завдяки ІУС стає можливим швидке оновлення інформації, забезпечення прозорої комунікації та адаптація до постійно змінних умов. Таким чином, інформаційні управлінські системи стають не лише інструментом оптимізації роботи туристичних агенцій, а й фундаментом для стійкого розвитку бізнесу в умовах цифрової економіки та глобальної нестабільності.

Список використаних джерел

1. Туризм в умовах війни: як подорожувати Україною безпечно. *greentour*. *Спілка сільського зеленого туризму України*. 19.03.2024. URL: https://greentour.com.ua/turyzm-v-umovah-vijny-yak-podorozhuvaty/ (дата звернення: 09.04.2025).
2. Макаренко Н. Нова сторінка. Як війна змінить мандрівки Україною після перемоги над ворогом. *РБК-Україна*. *Travel*. 15.04.2022. URL: https://www.rbc.ua/ukr/travel/novaya-stranitsa-voyna-izmenit-puteshestviya-1649942226.html (дата звернення: 09.04.2025).
3. Ширкова А. О., Погасій С. О. Вплив війни в Україні на туризм: аналіз наслідків та нові функції туризму. *Сучасні тенденції розвитку індустрії туризму та гостинності: глобальні виклики* : Матеріали ІІІ Міжнар. наук.-практ. конф., Харків, Україна, 15 квіт. 2024. Харків : ХНУМГ імені О.М. Бекетова, 2024. С. 34–36.
4. Відновлення туризму і туризм для відновлення. *Ukraïner*. *Взаємодопомога на війні*. 29.11.2023. URL: https://www.ukrainer.net/turyzm-vidnovlennia/ (дата звернення: 09.04.2025).
5. Українські біженці після трьох років за кордоном. Четверта хвиля дослідження / Г. Вишлінський та ін. *Центр Економічної Стратегії*. *Дослідження*. 03.03.2025. URL: https://ces.org.ua/refugees_fourth_wave/ (дата звернення: 09.04.2025).

Науковий керівник: Тішков Б.О., кандидат економічних наук.

Додаток Б

Backend:

AccountController:

```
using CloudinaryDotNet;
using CloudinaryDotNet.Actions;
using Dipchik.Services;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Hosting.DbContexts;
using Npgsql.BackendMessages;
using Shared.Model;
using Shared.Model.Dtos;

namespace Dipchik.Controllers;

public static class AccountController
{
    public static IEndpointRouteBuilder AddAccountController(this IEndpointRouteBuilder builder,
params AccountRolesEnum[] roleRequirements)
    {
        var group = builder.MapGroup("Account");
        foreach (var role in roleRequirements)
        {
            group.RequireAuthorization(role.ToString());
        }

        group.MapGet("/GetAccountInfo", GetAccountInfo);
        group.MapGet("/DeleteAccount", DeleteAccount);
        group.MapPost("/UpdateUser", UpdateUser);

        return builder;
    }

    public static async Task<IResult> UpdateUser([FromBody]byte[] imageBytes, string Username,
string localeCode, HttpContext context, SqlContext dbContext, Cloudinary cloudinary,
CancellationToken stoppingToken)
    {
        var accountId = context.UserId();
        var account = await dbContext.Accounts.FirstOrDefaultAsync(x => x.AccountId == accountId,
stoppingToken);
        if (account is null)
        {
            return Results.BadRequest("Account not found");
        }

        if (imageBytes.Length > 0)
        {
            Stream stream = new MemoryStream(imageBytes);
            var uploadParams = new ImageUploadParams()
            {
                Overwrite = true,
                DisplayName = $"{account.AccountId}_avatar",
                File = new FileDescription($"{account.AccountId}_avatar", stream)
            };
            var uploadResult = cloudinary.Upload(uploadParams);
            if (uploadResult.Url is null)
            {
                return Results.BadRequest("Failed to upload image to CDN");
            }

            account.AvatarUrl = uploadResult.Url.OriginalString;
        }

        account.Username = Username;
        if (!Enum.TryParse<LocalizationCode>(localeCode, out var localization))
        {
            return Results.BadRequest("Bad localization code provided");
        }

        if (localization == LocalizationCode.None)
        {
            return Results.BadRequest("Localization can't be None");
```

```
        }

        account.Locale = localization;
        await dbContext.SaveChangesAsync(stoppingToken);
        return Results.Ok(new AccountInfoDto(account));
    }

    public static async Task<IResult> GetAccountInfo(HttpContext context, SqlContext dbContext,
CancellationToken stoppingToken)
    {
        var accountId = context.UserId();
        var account = await dbContext.Accounts.FirstOrDefaultAsync(x => x.AccountId == accountId,
stoppingToken);
        if (account is null)
        {
            return Results.BadRequest("Account not found");
        }

        return Results.Ok(new AccountInfoDto(account));
    }

    public static async Task<IResult> DeleteAccount(HttpContext context, SqlContext dbContext,
CancellationToken stoppingToken)
    {
        var accountId = context.UserId();
        var account = await dbContext.Accounts.IgnoreQueryFilters().FirstOrDefaultAsync(x =>
x.AccountId == accountId, stoppingToken);
        if (account is null)
        {
            return Results.BadRequest("Account not found");
        }
        if (account.IsDeleted)
        {
            return Results.BadRequest("Account already deleted");
        }

        account.IsDeleted = true;
        account.UtcDeleted = DateTime.UtcNow;
        await dbContext.SaveChangesAsync(stoppingToken);

        return Results.Ok();
    }
}
```

AdminController:

```
using Dipchik.Services;
using Microsoft.AspNetCore.Mvc;
using Shared.Model;
using Shared.Model.Dtos;

namespace Dipchik.Controllers;

public static class AdminController
{
    public static IEndpointRouteBuilder AddAdminController(this IEndpointRouteBuilder builder,
params AccountRolesEnum[] roleRequirements)
    {
        var group = builder.MapGroup("Admin");
        foreach (var role in roleRequirements)
        {
            group.RequireAuthorization(role.ToString());
        }

        group.MapGet("/updateLocationsLocalizations", UpdateLocationsLocalizations);
        group.MapGet("/getAllDisplayLocalizations", GetAllDisplayLocalizations);
        group.MapPost("/updateDisplayLocalizations", UpdateDisplayLocalizations);

        return builder;
    }

    public static async Task<IResult> UpdateLocationsLocalizations(LocalizationsService
localizationsService, CancellationToken stoppingToken)
    {
        await localizationsService.UpdateLocaleLocalizations(stoppingToken);
        return Results.Ok();
    }
```

```
    public static async Task<IResult> UpdateDisplayLocalizations([FromBody]
List<LocalizationManagerDto> entries, LocalizationsService localizationsService, CancellationToken
stoppingToken)
    {
        var result = new Dictionary<LocalizationCode, Dictionary<string, string>>();

        foreach (var entry in entries)
        {
            foreach (var (langStr, value) in entry.Translations)
            {
                var lang = Enum.Parse<LocalizationCode>(langStr);
                if (!result.TryGetValue(lang, out var langDict))
                {
                    langDict = new Dictionary<string, string>();
                    result[lang] = langDict;
                }

                langDict[entry.Placeholder] = value;
            }
        }

        await localizationsService.UpdateDisplayLocalizations(result, stoppingToken);
        return Results.Ok();
    }

    public static async Task<IResult> GetAllDisplayLocalizations(LocalizationsService
localizationsService, CancellationToken stoppingToken)
    {
        var data = await localizationsService.GetAllDisplayLocalizations(stoppingToken);
        var lookup = new Dictionary<string, LocalizationManagerDto>();

        foreach (var (lang, translations) in data)
        {
            foreach (var (key, value) in translations)
            {
                if (!lookup.TryGetValue(key, out var entry))
                {
                    entry = new LocalizationManagerDto { Placeholder = key };
                    lookup[key] = entry;
                }

                entry.Translations[lang.ToString()] = value;
            }
        }
        return Results.Ok(lookup.Values);
    }
}
```

AuthController:

```
using Dipchik.Services;
using Shared.Model;

namespace Dipchik.Controllers;

public static class AuthController
{
    public static IEndpointRouteBuilder AddAuthController(this IEndpointRouteBuilder builder,
params AccountRolesEnum[] roleRequirements)
    {
        var group = builder.MapGroup("Auth");
        foreach (var role in roleRequirements)
        {
            group.RequireAuthorization(role.ToString());
        }

        group.MapGet("/register", Register);
        group.MapGet("/login", Login);

        return builder;
    }
    public static async Task<IResult> Register(string login, string password, AuthService
authService, JwtTokenGenerator tokenGenerator)
    {
        var result = await authService.AddNewUser(login, AuthService.PasswordHasher(password));
        if (!result.TryGetValue(out var account))
```

```
        {
            return Results.BadRequest(result.ErrorMessageCode);
        }

        return Results.Ok(tokenGenerator.GenerateJwt(account));
    }
    public static async Task<IResult> Login(string login, string password, AuthService authService,
JwtTokenGenerator tokenGenerator)
    {
        var result = await authService.ValidateUser(login, AuthService.PasswordHasher(password));
        if (!result.TryGetValue(out var account))
        {
            return Results.BadRequest(result.ErrorMessageCode);
        }

        return Results.Ok(tokenGenerator.GenerateJwt(account));
    }
}
```

BookingController:

```
using Backend.Services;
using Dipchik.Services;
using Shared.Model;

namespace Dipchik.Controllers;

public static class BookingController
{
    public static IEndpointRouteBuilder AddBookingController(this IEndpointRouteBuilder builder,
params AccountRolesEnum[] roleRequirements)
    {
        var group = builder.MapGroup("Booking");
        foreach (var role in roleRequirements)
        {
            group.RequireAuthorization(role.ToString());
        }

        group.MapGet("/BookTour", BookTour);
        group.MapGet("/GetMyBookings", GetMyBookings);
        group.MapGet("/CancelBooking", CancelBooking);

        return builder;
    }

    public static async Task<IResult> BookTour(int id, AuthService authService, BookingService
bookingService, CancellationToken stoppingToken)
    {
        var accountResult = await authService.GetAccount(stoppingToken);
        if (!accountResult.TryGetValue(out var account))
        {
            return Results.BadRequest(accountResult.ErrorMessageCode);
        }

        var result = await bookingService.BookTour(account, id, stoppingToken);
        if (!result.IsSuccess)
        {
            return Results.BadRequest(result.ErrorMessageCode);
        }
        return Results.Ok();
    }

    public static async Task<IResult> GetMyBookings(AuthService authService, BookingService
bookingService, CancellationToken stoppingToken)
    {
        var accountResult = await authService.GetAccount(stoppingToken);
        if (!accountResult.TryGetValue(out var account))
        {
            return Results.BadRequest(accountResult.ErrorMessageCode);
        }

        var result = await bookingService.GetMyBookings(account, stoppingToken);
        if (!result.TryGetValue(out var bookings))
        {
            return Results.BadRequest(result.ErrorMessageCode);
        }
```

```
        return Results.Ok(bookings);
    }

    public static async Task<IResult> CancelBooking(int id, AuthService authService, BookingService
bookingService, CancellationToken stoppingToken)
    {
        var accountResult = await authService.GetAccount(stoppingToken);
        if (!accountResult.TryGetValue(out var account))
        {
            return Results.BadRequest(accountResult.ErrorMessageCode);
        }

        var result = await bookingService.CancelBooking(account, id, stoppingToken);
        if (!result.IsSuccess)
        {
            return Results.BadRequest(result.ErrorMessageCode);
        }

        return Results.Ok();
    }
}
```

GuideController:

```
using Backend.Services;
using Shared.Model;

namespace Dipchik.Controllers;

public static class GuideController
{
    public static IEndpointRouteBuilder AddGuideController(this IEndpointRouteBuilder builder,
params AccountRolesEnum[] roleRequirements)
    {
        var group = builder.MapGroup("Guide");
        foreach (var role in roleRequirements)
        {
            group.RequireAuthorization(role.ToString());
        }

        group.MapGet("GetTours", GetGuideTours);
        group.MapGet("GetTourInstances", GetGuideTourInstances);

        return builder;
    }

    public static async Task<IResult> GetGuideTours(GuideService guideService, HttpContext context,
CancellationToken stoppingToken)
    {
        var accountId = context.UserId();
        if (accountId == null)
        {
            return Results.BadRequest(ErrorCodes.ErrorCode_Unauthorized);
        }

        var result = await guideService.GetGuideTours(accountId, stoppingToken);
        if (!result.TryGetValue(out var tours))
        {
            return Results.BadRequest(result.ErrorMessageCode);
        }

        return Results.Ok(tours);
    }

    public static async Task<IResult> GetGuideTourInstances(GuideService guideService, HttpContext
context, CancellationToken stoppingToken)
    {
        var accountId = context.UserId();
        if (accountId == null)
        {
            return Results.BadRequest(ErrorCodes.ErrorCode_Unauthorized);
        }

        var result = await guideService.GetGuideTourInstances(accountId, stoppingToken);
        if (!result.TryGetValue(out var tours))
        {
            return Results.BadRequest(result.ErrorMessageCode);
```

```
        }

        return Results.Ok(tours);
    }
}
```

LocalizationController:

```
using Dipchik.Services;
using Shared.Model;

namespace Dipchik.Controllers;

public static class LocalizationController
{
    public static IEndpointRouteBuilder AddLocalizationController(this IEndpointRouteBuilder
builder, params AccountRolesEnum[] roleRequirements)
    {
        var group = builder.MapGroup("localization");
        foreach (var role in roleRequirements)
        {
            group.RequireAuthorization(role.ToString());
        }

        group.MapGet("/LoadDisplayLocalization", LoadDisplayLocalization);
        group.MapGet("/LoadLocationLocalization", LoadLocationLocalization);

        return builder;
    }

    public static async Task<IResult> LoadDisplayLocalization(string localeCode, CacheManager
cache, CancellationToken stoppingToken)
    {
        if (!Enum.TryParse<LocalizationCode>(localeCode, out var localization))
        {
            return Results.BadRequest(ErrorCodes.ErrorCode_LocaleNotFound);
        }
        if (localization == LocalizationCode.None)
        {
            return Results.BadRequest(ErrorCodes.ErrorCode_LocaleNotSupported);
        }

        var res = await cache.GetDisplayLocalizations(localization, stoppingToken);
        return Results.Ok(res);
    }

    public static async Task<IResult> LoadLocationLocalization(string localeCode, CacheManager
cache, CancellationToken stoppingToken)
    {
        if (!Enum.TryParse<LocalizationCode>(localeCode, out var localization))
        {
            return Results.BadRequest(ErrorCodes.ErrorCode_LocaleNotFound);
        }
        if (localization == LocalizationCode.None)
        {
            return Results.BadRequest(ErrorCodes.ErrorCode_LocaleNotSupported);
        }

        var res = await cache.GetLocationLocalizations(localization, stoppingToken);
        return Results.Ok(res);
    }
}
```

ToursController:

```
using Microsoft.AspNetCore.Mvc;
using Shared.Model;
using Shared.Model.Dtos;
using Backend.Services;
using Dipchik.Services;

namespace Dipchik.Controllers;

public static class ToursController
{
    public static IEndpointRouteBuilder AddToursController(this IEndpointRouteBuilder builder,
params AccountRolesEnum[] roleRequirements)
```

```
{
    var group = builder.MapGroup("Tours");
    foreach (var role in roleRequirements)
    {
        group.RequireAuthorization(role.ToString());
    }

    group.MapPost("/GetAll", GetAllTours);
    group.MapGet("/GetById/{id}", GetTourById);
    group.MapGet("/GetRecommendations", GetTourRecommendations);
    group.MapPost("/RateTour", RateTour);

    return builder;
}

public static async Task<IResult> GetAllTours(
    [FromQuery] int page,
    [FromQuery] int pageSize,
    [FromBody] TourFiltersDto filters,
    ToursService toursService,
    CancellationToken stoppingToken)
{
    var result = await toursService.GetAllTours(page, pageSize, filters, stoppingToken);
    if (!result.TryGetValue(out var tours))
    {
        return Results.BadRequest(result.ErrorMessageCode);
    }

    return Results.Ok(tours);
}

public static async Task<IResult> GetTourById(
    int id,
    ToursService toursService,
    CancellationToken stoppingToken)
{

    var result = await toursService.GetTourById(id, stoppingToken);
    if (!result.TryGetValue(out var tours))
    {
        return Results.BadRequest(result.ErrorMessageCode);
    }

    return Results.Ok(tours);
}

public static async Task<IResult> GetTourRecommendations(
    AuthService authService,
    ToursService toursService,
    CancellationToken stoppingToken)
{
    var accountResult = await authService.GetAccount(stoppingToken);
    if (!accountResult.TryGetValue(out var account))
    {
        return Results.BadRequest(accountResult.ErrorMessageCode);
    }

    var result = await toursService.GetTourRecommendations(account, stoppingToken);
    if (!result.TryGetValue(out var recommendations))
    {
        return Results.BadRequest(result.ErrorMessageCode);
    }

    return Results.Ok(recommendations);
}

public static async Task<IResult> RateTour(
    AuthService authService,
    ToursService toursService,
    [FromBody] TourRateDto request,
    CancellationToken stoppingToken)
{
    var accountResult = await authService.GetAccount(stoppingToken);
    if (!accountResult.TryGetValue(out var account))
    {
        return Results.BadRequest(accountResult.ErrorMessageCode);
    }
```

```
        var result = await toursService.RateTour(account, request, stoppingToken);
        if (!result.IsSuccess)
        {
            return Results.BadRequest(result.ErrorMessageCode);
        }

        return Results.Ok();
    }
}
```

AuthService:

```
using System.Security.Cryptography;
using System.Text;
using Common.Model.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Hosting.DbContexts;
using Shared.Model;

namespace Dipchik.Services;

public class AuthService
{
    private ILogger<AuthService> logger;
    private readonly SqlContext dbContext;
    private readonly HttpContext httpContext;

    public AuthService(ILoggerFactory loggerFactory, SqlContext dbContext, IHttpContextAccessor
httpContextAccessor)
    {
        logger = loggerFactory.CreateLogger<AuthService>();
        this.dbContext = dbContext;
        this.httpContext = httpContextAccessor.HttpContext;
    }

    private async Task<OperationResult<Account>> GetExistingUser(string login)
    {
        var account = await dbContext.Accounts.FirstOrDefaultAsync(x => x.Login == login);
        if (account is null)
        {
            logger.LogWarning("User doesn't exist, login: {login}", login);
            return new OperationResult<Account>(ErrorCodes.ErrorCode_AccountNotFound);
        }

        return new OperationResult<Account>(account);
    }

    public async Task<OperationResult<Account>> AddNewUser(string login, string passwordHash)
    {
        var result = await GetExistingUser(login);
        if (result.TryGetValue(out var account))
        {
            logger.LogWarning("Tried to register already existing user: {userGuid}",
account.AccountId);
            return account.PasswordHash != passwordHash! ? new
OperationResult<Account>(ErrorCodes.ErrorCode_TriedToRegisterExistingAccount) : result;
        }

        account = new Account(passwordHash, login);
        await dbContext.Accounts.AddAsync(account);
        await dbContext.SaveChangesAsync();
        return new OperationResult<Account>(account);
    }

    public async Task<OperationResult<Account>> ValidateUser(string login, string passwordHash)
    {
        var result = await GetExistingUser(login);
        if (!result.TryGetValue(out var account))
        {
            return result;
        }

        if (passwordHash != account.PasswordHash)
        {
            logger.LogInformation("Wrong password, login: {login}", login);
            return new OperationResult<Account>(ErrorCodes.ErrorCode_AccountWrongPassword);
```

```
        }

        return result;
    }

    public async Task<OperationResult<Account>> GetAccount(CancellationToken stoppingToken)
    {
        var accountId = httpContext.UserId();
        if (accountId == null)
        {
            return new OperationResult<Account>(ErrorCodes.ErrorCode_AccountNotFound);
        }

        var account = await dbContext.Accounts.FirstOrDefaultAsync(x => x.AccountId == accountId,
stoppingToken);
        if (account is null)
        {
            return new OperationResult<Account>(ErrorCodes.ErrorCode_AccountNotFound);
        }

        return new OperationResult<Account>(account);
    }

    public static string PasswordHasher(string password)
    {
        using SHA256 sha256 = SHA256.Create();
        var bytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(password));
        return Convert.ToHexString(bytes);
    }
}
```

BookingService:

```
using Common.Model.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Hosting.DbContexts;
using Shared.Model.Dtos;
using Shared.Model;

namespace Backend.Services
{
    public class BookingService
    {
        private readonly SqlContext dbContext;
        private ILogger<BookingService> logger;

        public BookingService(ILoggerFactory loggerFactory, SqlContext dbContext)
        {
            logger = loggerFactory.CreateLogger<BookingService>();
            this.dbContext = dbContext;
        }

        public async Task<OperationResult> BookTour(Account account, int id, CancellationToken
stoppingToken)
        {
            var tour = await dbContext.TourInstances
                .Where(x => x.Id == id)
                .Select(x => new { x.IsCancelled, x.MaxParticipants, x.Tour.Price, x.EndDate })
                .FirstOrDefaultAsync(stoppingToken);
            if (tour is null)
            {
                logger.LogError("Tour with ID {id} not found", id);
                return new OperationResult(ErrorCodes.ErrorCode_TourNotFound);
            }
            if (TourInstance.GetStatus(tour.IsCancelled, tour.EndDate) !=
TourInstanceStatus.Scheduled)
            {
                logger.LogError("Tour with ID {id} is not available for booking", id);
                return new OperationResult(ErrorCodes.ErrorCode_TourIsNotAvailableForBooking);
            }

            var bookings = await dbContext.TourBookings
                .Where(x => x.TourInstanceId == id && x.AccountId == account.Id &&
x.CancellationDate == null)
                .Select(x => new
                {
                    x.Id,
```

```
                    x.AccountId
                })
                .ToListAsync(stoppingToken);
            if (bookings.Count >= tour.MaxParticipants)
            {
                logger.LogError("Tour with ID {id} is fully booked", id);
                return new OperationResult(ErrorCodes.ErrorCode_TourIsFullyBooked);
            }
            if (bookings.Any(x => x.AccountId == account.Id))
            {
                logger.LogError("Tour with ID {id} is already booked by account {accountId}", id,
account.Id);
                return new OperationResult(ErrorCodes.ErrorCode_TourIsBookedByAccount);
            }

            var booking = new TourBooking
            {
                TourInstanceId = id,
                AccountId = account.Id,
                BookedUtc = DateTime.UtcNow,
                TotalPrice = tour.Price
            };

            await dbContext.TourBookings.AddAsync(booking, stoppingToken);
            await dbContext.SaveChangesAsync(stoppingToken);

            return OperationResult.Success();
        }

        public async Task<OperationResult<List<BookingDto>>> GetMyBookings(Account account,
CancellationToken stoppingToken)
        {
            var bookings = await dbContext.TourBookings
                .Where(x => x.AccountId == account.Id)
                .Include(x => x.TourInstance)
                .ThenInclude(x => x.Tour)
                .Include(x => x.TourInstance)
                .ThenInclude(x => x.Rates)
                .OrderByDescending(x => x.BookedUtc)
                .Select(x => new BookingDto
                {
                    Id = x.Id,
                    TourInstanceId = x.TourInstanceId,
                    TourTitle = x.TourInstance.Tour.Title,
                    StartDate = x.TourInstance.StartDate,
                    EndDate = x.TourInstance.EndDate,
                    TotalPrice = x.TotalPrice,
                    IsCancelled = x.IsCancelled,
                    HasRated = x.TourInstance.Rates.Any(r => r.TouristAccountId == account.Id)
                })
                .ToListAsync(stoppingToken);

            return new OperationResult<List<BookingDto>>(bookings);
        }

        public async Task<OperationResult> CancelBooking(Account account, int id, CancellationToken
stoppingToken)
        {
            var booking = await dbContext.TourBookings
                .Include(x => x.TourInstance)
                .FirstOrDefaultAsync(x => x.Id == id && x.AccountId == account.Id, stoppingToken);

            if (booking is null)
            {
                logger.LogError("Booking with ID {id} not found for account {accountId}", id,
account.Id);
                return new OperationResult(ErrorCodes.ErrorCode_BookingNotFound);
            }

            if (booking.IsCancelled)
            {
                logger.LogError("Booking with ID {id} is already cancelled", id);
                return new OperationResult(ErrorCodes.ErrorCode_BookingIsCancelled);
            }

            if (booking.TourInstance.StartDate <= DateTime.UtcNow.AddDays(1))
            {
```

```
                logger.LogError("Booking with ID {id} is too late to be cancelled for account
{account}", id, account.Id);
                return new OperationResult(ErrorCodes.ErrorCode_TooLateToCancelBooking);
            }

            booking.CancellationDate = DateTime.UtcNow;
            booking.CancellationReason = "Cancelled by user";

            await dbContext.SaveChangesAsync(stoppingToken);

            return OperationResult.Success();
        }
    }
}
```

CacheManager:

```
using Shared.Model.Dtos;
using Shared.Model;
using System.Text.Json;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Hosting.DbContexts;
using StackExchange.Redis;

namespace Dipchik.Services;

public class CacheManager
{
    private readonly IDatabase redisDB;
    private readonly IConfiguration configuration;
    private readonly IConnectionMultiplexer connectionMultiplexer;
    private readonly IServiceScopeFactory scopeFactory;
    private ILogger<CacheManager> logger;
    private readonly int localizationsDbInt;

    public CacheManager(ILoggerFactory loggerFactory, IConfiguration configuration,
IConnectionMultiplexer connectionMultiplexer, IServiceScopeFactory scopeFactory)
    {
        logger = loggerFactory.CreateLogger<CacheManager>();
        localizationsDbInt = configuration.GetValue<int>("Redis:LocalizationsDbInt");

        redisDB = connectionMultiplexer.GetDatabase(localizationsDbInt);
        this.configuration = configuration;
        this.connectionMultiplexer = connectionMultiplexer;
        this.scopeFactory = scopeFactory;
    }

    public async Task ClearLocationLocalizationsCache()
    {

        var endpoints = connectionMultiplexer.GetEndPoints();
        var server = connectionMultiplexer.GetServer(endpoints[0]);

        var keys = server.Keys(database: localizationsDbInt, pattern:
$"{Constants.LOCATION_LOCALIZATION_CACHE_KEY}_*").ToArray();

        foreach (var key in keys)
        {
            await redisDB.KeyDeleteAsync(key);
            logger.LogInformation("Clearing Location Localization for {lang} language",
key.ToString());
        }
    }
    public async Task ClearDisplayLocalizationsCache()
    {

        var endpoints = connectionMultiplexer.GetEndPoints();
        var server = connectionMultiplexer.GetServer(endpoints[0]);

        var keys = server.Keys(database: localizationsDbInt, pattern:
$"{Constants.DISPLAY_LOCALIZATION_CACHE_KEY}_*").ToArray();

        foreach (var key in keys)
        {
            await redisDB.KeyDeleteAsync(key);
```

```
            logger.LogInformation("Clearing Display Localization for {lang} language",
key.ToString());
        }
    }

    public async Task<Dictionary<int, LanguageLocationsDto>>
GetLocationLocalizations(LocalizationCode code, CancellationToken stoppingToken)
    {
        var scope = scopeFactory.CreateScope();
        var dbContext = scope.ServiceProvider.GetService<SqlContext>();

        var transactionRowValue = await
redisDB.HashGetAllAsync($"{Constants.LOCATION_LOCALIZATION_CACHE_KEY}_{code.ToString()}");
        if (transactionRowValue.Any())
        {
            return transactionRowValue.ToDictionary(x => int.Parse(x.Name), x =>
JsonSerializer.Deserialize<LanguageLocationsDto>(x.Value));
        }

        var data = await dbContext.Languages.Where(x => x.Locale == code).Select(x =>
x.CitiesAndCountriesJson)
            .FirstOrDefaultAsync(stoppingToken);
        var list = JsonSerializer.Deserialize<List<LanguageLocationsDto>>(data);
        await
redisDB.HashSetAsync($"{Constants.LOCATION_LOCALIZATION_CACHE_KEY}_{code.ToString()}",
            list.Select(x => new HashEntry(x.GeoId.ToString(),
JsonSerializer.Serialize(x))).ToArray());
        logger.LogInformation("Setting Location Localization for {lang} language",
code.ToString());

        return list.ToDictionary(x => x.GeoId);
    }

    public async Task<Dictionary<string, string>> GetDisplayLocalizations(LocalizationCode code,
CancellationToken stoppingToken)
    {
        var scope = scopeFactory.CreateScope();
        var dbContext = scope.ServiceProvider.GetService<SqlContext>();

        var transactionRowValue = await
redisDB.HashGetAllAsync($"{Constants.DISPLAY_LOCALIZATION_CACHE_KEY}_{code.ToString()}");
        if (transactionRowValue.Any())
        {
            return transactionRowValue.ToDictionary(x => x.Name.ToString(), x =>
x.Value.ToString());
        }

        var data = await dbContext.Languages.Where(x => x.Locale == code).Select(x =>
x.DisplayLocalizationsJson).FirstOrDefaultAsync(stoppingToken);
        var list = JsonSerializer.Deserialize<Dictionary<string, string>>(data);
        await redisDB.HashSetAsync($"{Constants.DISPLAY_LOCALIZATION_CACHE_KEY}_{code.ToString()}",
            list.Select(x => new HashEntry(x.Key, x.Value)).ToArray());
        logger.LogInformation("Setting Display Localization for {lang} language", code.ToString());

        return list.ToDictionary(x => x.Key, x => x.Value);
    }
}
```

GuideService:

```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Hosting.DbContexts;
using Shared.Model;
using Shared.Model.Dtos;

namespace Backend.Services;

public class GuideService
{
    private readonly SqlContext dbContext;
    private readonly ILogger<GuideService> logger;

    public GuideService(ILoggerFactory loggerFactory, SqlContext dbContext)
    {
        logger = loggerFactory.CreateLogger<GuideService>();
        this.dbContext = dbContext;
    }
```

```
    public async Task<OperationResult<List<TourDto>>> GetGuideTours(string accountId,
CancellationToken stoppingToken)
    {
        var guideId = await dbContext.Guides
            .Where(x => x.Account.AccountId == accountId)
            .Select(x => x.Id)
            .FirstOrDefaultAsync(stoppingToken);

        var tours = dbContext.TourInstances
            .Include(x => x.Tour)
            .ThenInclude(x => x.Guide)
            .ThenInclude(x => x.Account)
            .Where(x => x.Tour.GuideId == guideId)
            .AsQueryable();


        var result = await tours.Select(t => new TourDto
        {
            Id = t.Id,
            Title = t.Tour.Title,
            Description = t.Tour.Description,
            ImageUrl = t.Tour.ImageUrl,
            Locations = t.Tour.Locations,
            Price = t.Tour.Price,
            TourType = t.Tour.TourType,
            WithGuide = t.Tour.WithGuide,
            SpecialOffers = t.Tour.SpecialOffers,
            DurationDays = t.Tour.DurationDays,
            GuideName = t.Tour.Guide.Name,
            GuideSurname = t.Tour.Guide.Surname,
            GuideAvatarUrl = t.Tour.Guide.Account.AvatarUrl,
            Classification = t.Tour.Classification
        }).ToListAsync(stoppingToken);

        return new OperationResult<List<TourDto>>(result);
    }

    public async Task<OperationResult<List<TourDescDto>>> GetGuideTourInstances(string accountId,
CancellationToken stoppingToken)
    {
        var guideId = await dbContext.Guides
            .Where(x => x.Account.AccountId == accountId)
            .Select(x => x.Id)
            .FirstOrDefaultAsync(stoppingToken);

        var tours = dbContext.TourInstances
            .Include(x => x.Tour)
            .ThenInclude(x => x.Guide)
            .ThenInclude(x => x.Account)
            .Where(x => x.Tour.GuideId == guideId)
            .AsQueryable();

        var result = await tours.Select(tour => new TourDescDto
        {
            Id = tour.Id,
            TourId = tour.TourId,
            ImageUrl = tour.Tour.ImageUrl,
            Title = tour.Tour.Title,
            Description = tour.Tour.Description,
            Locations = tour.Tour.Locations,
            Price = tour.Tour.Price,
            TourType = tour.Tour.TourType,
            StartDate = tour.StartDate,
            EndDate = tour.EndDate,
            Rating = tour.Rating ?? 0d,
            MaxParticipants = tour.MaxParticipants,
            CurrentParticipants = tour.Bookings.Count,
            GuideName = tour.Tour.Guide.Name,
            GuideSurname = tour.Tour.Guide.Surname,
            GuideAvatarUrl = tour.Tour.Guide.Account.AvatarUrl,
            IsCancelled = tour.IsCancelled,
            Classification = tour.Tour.Classification
        }).ToListAsync(stoppingToken);

        return new OperationResult<List<TourDescDto>>(result);
    }
```

```
}
```

JwtTokenGenerator:

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using Common.Model.Entities;
using Microsoft.IdentityModel.Tokens;
using Shared.Model;

namespace Dipchik.Services;

public class JwtTokenGenerator
{
    private readonly string jwtKey;
    private readonly SigningCredentials credentials;

    public JwtTokenGenerator(IConfiguration configuration)
    {
        jwtKey = configuration.GetValue<string>("Auth:JwtKey");
        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtKey));
        credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
    }

    public string GenerateJwt(Account account)
        => GenerateJwt(account.Login, account.AccountId, account.Roles);

    public string GenerateJwt(string login, string accountId, AccountRolesEnum roles)
    {
        var claims = new[]
        {
            new Claim(JwtRegisteredClaimNames.Sub, accountId),
            new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
            new Claim(ClaimTypes.Name, accountId),
            new Claim(ClaimTypes.Surname, login),
            new Claim(ClaimTypes.Role, ((int)roles).ToString())
        };

        var token = new JwtSecurityToken(
            claims: claims,
            expires: DateTime.UtcNow.AddDays(1),
            signingCredentials: credentials
        );

        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}
```

LocalizationsService:

```
using System.Globalization;
using System.Text.Json;
using System.Text.Json.Serialization;
using System.Threading.Channels;
using Common.Model.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Hosting.DbContexts;
using Shared.Model;
using Shared.Model.Dtos;

namespace Dipchik.Services;


public record class GeoNamesLocalizationsDto(
    int geonameid,
    LocalizationCode locale,
    string Name,
    bool isPreferredName,
    bool isShortName,
    bool isColloquial,
    bool isHistoric);

public class GeoNamesCountryDto
{
    public int GeoId { get; set; }
    public string CountryCode { get; set; }
```

```
    public Dictionary<LocalizationCode, string> Names { get; set; } = new
Dictionary<LocalizationCode, string>();
}


public class LocalizationsService
{
    private readonly SqlContext dbContext;
    private readonly ChannelWriter<EventDto> eventProceeder;

    public LocalizationsService(SqlContext dbContext, ChannelWriter<EventDto> eventProceeder)
    {
        this.dbContext = dbContext;
        this.eventProceeder = eventProceeder;
    }

    public record class LanguageDisplayDto(LocalizationCode locale, string placeholder, string
value);

    public async Task<Dictionary<LocalizationCode, Dictionary<string, string>>>
GetAllDisplayLocalizations(CancellationToken stoppingToken)
    {
        var data = await dbContext.Languages.Select(x => new KeyValuePair<LocalizationCode,
string>(x.Locale, x.DisplayLocalizationsJson)).ToListAsync(stoppingToken);
        var result = data.ToDictionary(x => x.Key,
            x => JsonSerializer.Deserialize<Dictionary<string, string>>(x.Value));

        return result;
    }

    public async Task UpdateDisplayLocalizations(Dictionary<LocalizationCode, Dictionary<string,
string>> data, CancellationToken stoppingToken)
    {
        var codes = data.Keys.ToList();
        var dbLanguages = await dbContext.Languages.Where(x =>
codes.Contains(x.Locale)).ToListAsync(stoppingToken);
        foreach (var lang in dbLanguages)
        {
            var json = JsonSerializer.Serialize(data[lang.Locale]);
            lang.DisplayLocalizationsJson = json;
        }

        await dbContext.SaveChangesAsync(stoppingToken);
        await eventProceeder.WriteAsync(new EventDto(EventTypeEnum.DisplayLocalizationsUpdated,
DateTime.UtcNow, data), stoppingToken);
    }
    public async Task UpdateLocaleLocalizations(CancellationToken stoppingToken)
    {
        var data = await ParseCountries(stoppingToken);
        var dic = new Dictionary<LocalizationCode, List<LanguageLocationsDto>>();


        foreach (var item in data)
        {
            foreach (var lang in item.Names)
            {
                if (!dic.TryGetValue(lang.Key, out var list))
                {
                    list = [];
                    dic[lang.Key] = list;
                }
                list.Add(new LanguageLocationsDto(item.GeoId, lang.Value.Item1, lang.Value.Item2));
            }
        }


        var dbLocalizations = await dbContext.Languages.ToListAsync(stoppingToken);
        if (dic.Any(x => !dbLocalizations.Exists(y => y.Locale == x.Key)))
        {
            var existingLocales = dbLocalizations.Select(x => x.Locale);
            var nonExistantLangs = dic.Keys.Except(existingLocales);
            foreach (var lang in nonExistantLangs)
            {
                var init = new List<KeyValuePair<string, string>>
                {
                    new("init", "_")
                };
```

```
                dbContext.Languages.Add(new Language(locale: lang, citiesAndCountriesJson: "[]",
displayLocalizationsJson: JsonSerializer.Serialize(init)));
            }
            await dbContext.SaveChangesAsync(stoppingToken);

            dbLocalizations = await dbContext.Languages.ToListAsync(stoppingToken);
        }
        foreach (var lang in dbLocalizations)
        {
            var json = JsonSerializer.Serialize(dic[lang.Locale]);
            lang.CitiesAndCountriesJson = json;
        }

        await dbContext.SaveChangesAsync(stoppingToken);
        await eventProceeder.WriteAsync(new EventDto(EventTypeEnum.LocationsLocalizationsUpdated,
DateTime.UtcNow, data), stoppingToken);
    }

    public async Task<List<CityLocationDto>> ParseCountries(CancellationToken stoppingToken)
    {
        var dic = new Dictionary<int, CityLocationDto>();

        var fileLocation = Path.Combine(Environment.CurrentDirectory, "countryInfo.txt");
        StreamReader countriesReader = new StreamReader(fileLocation);
        var line = await countriesReader.ReadLineAsync(stoppingToken);
        var countriesDic = new Dictionary<int, GeoNamesCountryDto>();
        while (!countriesReader.EndOfStream)
        {
            var splitArr = line.Split('\t');
            var geoId = int.Parse(splitArr[16]);
            countriesDic[geoId] = new GeoNamesCountryDto
            {
                GeoId =  geoId,
                CountryCode = splitArr[0]
            };
            line = await countriesReader.ReadLineAsync(stoppingToken);
        }

        fileLocation = Path.Combine(Environment.CurrentDirectory, "cities500.txt");
        StreamReader citiesReader = new StreamReader(fileLocation);
        line = await citiesReader.ReadLineAsync(stoppingToken);
        while (!citiesReader.EndOfStream)
        {
            var splitArr = line.Split('\t');
            var cityCode = EnumSwitches.GetGeoFeatureCode(splitArr[7]);

            if(cityCode != GeoFeatureCode.None)
            {
                dic[int.Parse(splitArr[0])] = new CityLocationDto
                {
                    GeoId = int.Parse(splitArr[0]),
                    Latitude = float.Parse(splitArr[4], CultureInfo.InvariantCulture),
                    Longitude = float.Parse(splitArr[5], CultureInfo.InvariantCulture),
                    Name = splitArr[1],
                    CountryCode = splitArr[8]
                };
            }

            line = await citiesReader.ReadLineAsync(stoppingToken);
        }

        citiesReader.Close();

        fileLocation = Path.Combine(Environment.CurrentDirectory, "alternateNames.txt");
        StreamReader localizationsReader = new StreamReader(fileLocation);
        line = await localizationsReader.ReadLineAsync(stoppingToken);
        while (!localizationsReader.EndOfStream)
        {
            var splitArr = line.Split('\t');
            var info = new GeoNamesLocalizationsDto(
                geonameid: int.Parse(splitArr[1]),
                locale: EnumSwitches.GetLocalizationCode(splitArr[2]),
                Name: splitArr[3],
                isPreferredName: splitArr[4] == "1",
                isShortName: splitArr[5] == "1",
                isColloquial: splitArr[6] == "1",
                isHistoric: splitArr[7] == "1"
```

```
            );

            line = await localizationsReader.ReadLineAsync(stoppingToken);

            if (info.isShortName || info.isColloquial || info.isHistoric || info.locale ==
LocalizationCode.None)
            {
                continue;
            }

            if (countriesDic.TryGetValue(info.geonameid, out var countryLocalization))
            {
                if (!countryLocalization.Names.ContainsKey(info.locale) || info.isPreferredName)
                {
                    countryLocalization.Names[info.locale] = info.Name;
                }
            }
            if (dic.TryGetValue(info.geonameid, out var cityLocalization))
            {
                if (!cityLocalization.Names.ContainsKey(info.locale) || info.isPreferredName)
                {
                    cityLocalization.Names[info.locale] = (info.Name, null);
                }
            }
        }

        var newCountriesDic = countriesDic.Values.ToDictionary(x => x.CountryCode);
        var res = dic.Values.Where(x => x.Names.Count >= 3).ToList();

        foreach (var item in res)
        {
            if (newCountriesDic.TryGetValue(item.CountryCode, out var countryLocalization))
            {
                foreach (var lang in item.Names)
                {
                    item.Names[lang.Key] = (lang.Value.Item1, countryLocalization.Names[lang.Key]);
                }
            }
        }

        return res;
    }
}
```

ToursService

```
using Common.Model.Dtos;

using Common.Model.Entities;

using Microsoft.Extensions.Hosting.DbContexts;

using Shared.Model.Dtos;

using Shared.Model;

using System.Net.Http.Headers;

using System.Text.Json;

using System.Text;

using Microsoft.EntityFrameworkCore;


namespace Backend.Services;


public class ToursService

{

   private readonly SqlContext dbContext;

   private readonly IConfiguration configuration;
```

```
    private readonly ILogger<ToursService> logger;


    public ToursService(ILoggerFactory loggerFactory, SqlContext dbContext, IConfiguration configuration)
    {
        logger = loggerFactory.CreateLogger<ToursService>();
        this.dbContext = dbContext;
        this.configuration = configuration;
    }



    public async Task<OperationResult<PagedResult<TourDto>>> GetAllTours(int page, int pageSize, TourFiltersDto filters, CancellationToken
stoppingToken)
    {
        if (page < 1)
        {
            return new OperationResult<PagedResult<TourDto>>(ErrorCodes.ErrorCode_PageMustBeBiggerThanZero);
        }

        if (pageSize < 1 || pageSize > Constants.MaxPageSize)
        {
            return new OperationResult<PagedResult<TourDto>>(ErrorCodes.ErrorCode_PageSizeNotAllowed);
        }

        var query = dbContext.TourInstances
            .Where(x => !x.IsCancelled && x.StartDate > DateTime.UtcNow)
            .Include(x => x.Tour)
            .ThenInclude(x => x.Guide)
            .ThenInclude(x => x.Account)
            .AsQueryable();

        if (filters.SelectedDestination.HasValue)
        {
            query = query.Where(t => t.Tour.Locations.Contains(filters.SelectedDestination.Value));
        }

        if (filters.FromPrice > Constants.DefaultMinPrice)
        {
            query = query.Where(t => t.Tour.Price >= filters.FromPrice);
        }

        if (filters.ToPrice < Constants.DefaultMaxPrice)
```

```
{
    query = query.Where(t => t.Tour.Price <= filters.ToPrice);
}

if (filters.TourTypes != null && filters.TourTypes.Any())
{
    query = query.Where(t => filters.TourTypes.Contains(t.Tour.TourType));
}

if (filters.WithGuide.HasValue)
{
    query = query.Where(t => t.Tour.WithGuide == filters.WithGuide.Value);
}

if (filters.PrivateTour == true)
{
    query = query.Where(t => (t.Tour.Classification == TourClassificationEnum.Private));
}

else if (filters.GroupTour == true)
{
    query = query.Where(t => t.Tour.Classification == TourClassificationEnum.Group);
}

if (filters.OnSale.HasValue && filters.OnSale.Value)
{
    query = query.Where(t => (t.Tour.SpecialOffers & SpecialOfferEnum.OnSale) > 0);
}

if (filters.SpecialDiscount.HasValue && filters.SpecialDiscount.Value)
{
    query = query.Where(t => (t.Tour.SpecialOffers & SpecialOfferEnum.SpecialDiscount) == SpecialOfferEnum.SpecialDiscount);
}

if (filters.DestinationsCount.HasValue)
{
    switch (filters.DestinationsCount.Value)
    {
        case DestinationCountEnum.Single:
            query = query.Where(t => t.Tour.Locations.Count == 1);
```

```
            break;
        case DestinationCountEnum.TwoToThree:
            query = query.Where(t => t.Tour.Locations.Count == 2 || t.Tour.Locations.Count == 3);
            break;
        case DestinationCountEnum.FourOrMore:
            query = query.Where(t => t.Tour.Locations.Count >= 4);
            break;
    }
}

// Apply instance-level filters
if (filters.StartDate.HasValue)
{
    query = query.Where(i => i.StartDate.Date == filters.StartDate.Value.Date);
}

query = query.Where(i => i.Tour.DurationDays >= filters.FromDurationDays);
query = query.Where(i => i.Tour.DurationDays <= filters.ToDurationDays);

if (filters.MinRating > 1.0d)
{
    query = query.Where(i => i.Rating >= filters.MinRating);
}

if (filters.StartsSoon.HasValue && filters.StartsSoon.Value)
{
    var now = DateTime.UtcNow;
    query = query.Where(i => (i.StartDate - now).TotalDays <= Constants.STARTS_SOON_DAYS);
}

var totalCount = await query.CountAsync(stoppingToken);
var totalPages = (int)Math.Ceiling(totalCount / (double)pageSize);

// Apply sorting
if (!string.IsNullOrEmpty(filters.SortBy))
{
    query = filters.SortBy switch
    {
        "price" => query.OrderBy(t => t.Tour.Price),
        "rating" => query.OrderByDescending(i => i.Rating ?? 0),
```

```
        "duration" => query.OrderBy(t => t.Tour.DurationDays),
        "startdate" => query.OrderBy(i => i.StartDate),
        _ => query
    };
}

var tours = await query
    .Skip((page - 1) * pageSize)
    .Take(pageSize)
    .ToListAsync(stoppingToken);

var tourDtos = tours.Select(t => new TourDto
{
    Id = t.Id,
    Title = t.Tour.Title,
    Description = t.Tour.Description,
    ImageUrl = t.Tour.ImageUrl,
    Locations = t.Tour.Locations,
    Price = t.Tour.Price,
    TourType = t.Tour.TourType,
    WithGuide = t.Tour.WithGuide,
    SpecialOffers = t.Tour.SpecialOffers,
    DurationDays = t.Tour.DurationDays,
    GuideName = t.Tour.Guide.Name,
    GuideSurname = t.Tour.Guide.Surname,
    GuideAvatarUrl = t.Tour.Guide.Account.AvatarUrl,
    Classification = t.Tour.Classification
}).ToList();

var absoluteData = await dbContext.Tours.GroupBy(x => 1).Select(x => new
{
    MinDuration = x.Min(x => x.DurationDays),
    MaxDuration = x.Max(x => x.DurationDays),
    MinPrice = x.Min(x => x.Price),
    MaxPrice = x.Max(x => x.Price),
}).FirstAsync(stoppingToken);

var result = new PagedResult<TourDto>
{
    Items = tourDtos,
```

```
        TotalCount = totalCount,
        TotalPages = totalPages,
        MinDuration = absoluteData.MinDuration,
        MaxDuration = absoluteData.MaxDuration,
        MinPrice = absoluteData.MinPrice,
        MaxPrice = absoluteData.MaxPrice,
        CurrentPage = page,
        PageSize = pageSize
    };

    return new OperationResult<PagedResult<TourDto>>(result);
}

public async Task<OperationResult<TourDescDto>> GetTourById(int id, CancellationToken stoppingToken)
{
    var tour = await dbContext.TourInstances.Include(x => x.Rates).Where(t => t.Id == id).Select(tour => new TourDescDto
    {
        Id = tour.Id,
        TourId = tour.TourId,
        ImageUrl = tour.Tour.ImageUrl,
        Title = tour.Tour.Title,
        Description = tour.Tour.Description,
        Locations = tour.Tour.Locations,
        Price = tour.Tour.Price,
        TourType = tour.Tour.TourType,
        StartDate = tour.StartDate,
        EndDate = tour.EndDate,
        Rating = tour.Rating ?? 0d,
        MaxParticipants = tour.MaxParticipants,
        CurrentParticipants = tour.Bookings.Count,
        GuideName = tour.Tour.Guide.Name,
        GuideSurname = tour.Tour.Guide.Surname,
        GuideAvatarUrl = tour.Tour.Guide.Account.AvatarUrl,
        IsCancelled = tour.IsCancelled,
        Classification = tour.Tour.Classification
    }).FirstOrDefaultAsync(stoppingToken);

    if (tour == null)
    {
        logger.LogWarning("Tour with ID {id} not found", id);
```

```
            return new OperationResult<TourDescDto>(ErrorCodes.ErrorCode_TourNotFound);
        }

        return new OperationResult<TourDescDto>(tour);
    }

    public async Task<OperationResult<TourRecommendationResponseDto>> GetTourRecommendations(Account account, CancellationToken
stoppingToken)
    {
        // Get user's booking history
        var userBookings = await dbContext.TourBookings
            .Where(x => x.AccountId == account.Id && x.CancellationDate == null)
            .Include(x => x.TourInstance)
            .ThenInclude(x => x.Tour)
            .OrderByDescending(x => x.BookedUtc)
            .Take(10)
            .ToListAsync(stoppingToken);

        // Get user's ratings
        var userRatings = await dbContext.TourInstanceRates
            .Where(x => x.TouristAccountId == account.Id)
            .Include(x => x.TourInstance)
            .ThenInclude(x => x.Tour)
            .OrderByDescending(x => x.RatedTimeUtc)
            .Take(10)
            .ToListAsync(stoppingToken);

        // Get popular tours based on ratings
        var popularTours = await dbContext.TourInstances
            .Where(x => !x.IsCancelled && x.StartDate > DateTime.UtcNow)
            .Include(x => x.Tour)
            .Include(x => x.Rates)
            .OrderByDescending(x => x.Rates.Average(r => r.Rate))
            .Take(20)
            .ToListAsync(stoppingToken);

        // Prepare data for ChatGPT
        var userPreferences = new
        {
            PreviouslyBookedTourTypes = userBookings.Select(x => x.TourInstance.Tour.TourType).ToList(),
            PreviouslyBookedLocations = userBookings.SelectMany(x => x.TourInstance.Tour.Locations).Distinct().ToList(),
```

```
    PreviouslyRatedTours = userRatings.Select(x => x.TourInstance.TourId).ToList(),
    PreviousRatings = userRatings.Select(x => x.Rate / 10.0).ToList(),
};

var availableTours = popularTours.Select(x => new
{
    TourId = x.TourId,
    Title = x.Tour.Title,
    Description = x.Tour.Description,
    TourType = x.Tour.TourType,
    Locations = x.Tour.Locations,
    Price = x.Tour.Price,
    DurationDays = x.Tour.DurationDays,
    WithGuide = x.Tour.WithGuide,
    Rating = x.Rating ?? 0,
    SpecialOffers = x.Tour.SpecialOffers
}).ToList();

// Prepare ChatGPT prompt
var prompt = new
{
    model = "gpt-3.5-turbo",
    messages = new[]
    {
        new { role = "system", content = $"You are a tour recommendation expert. Analyze the user's preferences and available tours to recommend the best matches. Consider tour types, locations, price range, duration, and special offers. Your reason text must be made in {account.Locale.ToString()} language. Return a JSON array of tour IDs in order of recommendation, and a brief explanation of why these tours were recommended. Your response must be a JSON, that should follow this structure: private class ChatGptRecommendation{{ public List<int> recommendedTourIds {{ get; set; }} = new(); public string reason {{ get; set; }} = string.Empty; }}" },
        new { role = "user", content = JsonSerializer.Serialize(new { UserPreferences = userPreferences, AvailableTours = availableTours }) }
    },
    temperature = 0.7,
    max_tokens = 500
};

// Call ChatGPT API
var openAiUrl = configuration.GetValue<string>("OpenAiUrl");
var openAiKey = configuration.GetValue<string>("OpenAiKey");
using var cli = new HttpClient();
cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", openAiKey);
var response = await cli.PostAsync(openAiUrl,
    new StringContent(JsonSerializer.Serialize(prompt), Encoding.UTF8, "application/json"),
```

```
    stoppingToken);

if (!response.IsSuccessStatusCode)
{
    logger.LogError("Failed to get response from OpenAI API: {statusCode}", response.StatusCode);
    return new OperationResult<TourRecommendationResponseDto>(ErrorCodes.ErrorCode_FailedToGetOpenApiResponse);
}

var responseContent = await response.Content.ReadAsStringAsync(stoppingToken);
var chatGptResponse = JsonSerializer.Deserialize<ChatGptResponse>(responseContent, new JsonSerializerOptions
{
    PropertyNameCaseInsensitive = true
});

if (chatGptResponse?.Choices == null || !chatGptResponse.Choices.Any())
{
    logger.LogError("Invalid response from OpenAI API: {response}", responseContent);
    return new OperationResult<TourRecommendationResponseDto>(ErrorCodes.ErrorCode_InvalidOpenApiResponse);
}

// Parse ChatGPT response
var recommendation = JsonSerializer.Deserialize<ChatGptRecommendation>(
    chatGptResponse.Choices[0].Message.Content, new JsonSerializerOptions
    {
        PropertyNameCaseInsensitive = true
    });

if (recommendation == null)
{
    logger.LogError("Failed to parse OpenAI API response: {response}", chatGptResponse.Choices[0].Message.Content);
    return new OperationResult<TourRecommendationResponseDto>(ErrorCodes.ErrorCode_FailedToParseOpenApiResponse);
}

// Get recommended tours
var recommendedTours = await dbContext.TourInstances
    .Where(x => recommendation.RecommendedTourIds.Contains(x.TourId) && !x.IsCancelled && x.StartDate > DateTime.UtcNow)
    .Include(x => x.Tour)
    .ThenInclude(x => x.Guide)
    .ThenInclude(x => x.Account)
    .Select(x => new TourDto
```

```
        {
            Id = x.Id,
            Title = x.Tour.Title,
            Description = x.Tour.Description,
            ImageUrl = x.Tour.ImageUrl,
            Locations = x.Tour.Locations,
            Price = x.Tour.Price,
            TourType = x.Tour.TourType,
            WithGuide = x.Tour.WithGuide,
            SpecialOffers = x.Tour.SpecialOffers,
            DurationDays = x.Tour.DurationDays,
            GuideName = x.Tour.Guide.Name,
            GuideSurname = x.Tour.Guide.Surname,
            GuideAvatarUrl = x.Tour.Guide.Account.AvatarUrl,
            Classification = x.Tour.Classification
        })
        .ToListAsync(stoppingToken);

    // Order tours according to ChatGPT's recommendation
    recommendedTours = recommendedTours
        .OrderBy(x => recommendation.RecommendedTourIds.IndexOf(x.Id))
        .Take(3)
        .ToList();

    var result = new TourRecommendationResponseDto
    {
        RecommendedTours = recommendedTours,
        RecommendationReason = recommendation.Reason
    };

    return new OperationResult<TourRecommendationResponseDto>(result);
}

public async Task<OperationResult> RateTour(Account account, TourRateDto request, CancellationToken stoppingToken)
{
    // Verify the tour instance exists and has ended
    var tourInstance = await dbContext.TourInstances
        .Include(x => x.Rates)
        .FirstOrDefaultAsync(x => x.Id == request.TourInstanceId, stoppingToken);
```

```
if (tourInstance == null)
{
    logger.LogWarning("Tour instance with ID {id} not found", request.TourInstanceId);
    return new OperationResult(ErrorCodes.ErrorCode_TourNotFound);
}

if (tourInstance.GetStatus() != TourInstanceStatus.Completed)
{
    logger.LogWarning("Tour instance with ID {id} is not completed", request.TourInstanceId);
    return new OperationResult(ErrorCodes.ErrorCode_CannotRateNotCompletedTour);
}

// Check if user has already rated this tour
var existingRate = tourInstance.Rates.FirstOrDefault(x => x.TouristAccountId == account.Id);
if (existingRate != null)
{
    logger.LogWarning("User {userId} has already rated tour instance {tourInstanceId}", account.Id, request.TourInstanceId);
    return new OperationResult(ErrorCodes.ErrorCode_CannotRateRatedTour);
}

// Verify user was a participant
var booking = await dbContext.TourBookings
    .FirstOrDefaultAsync(x => x.TourInstanceId == request.TourInstanceId &&
                              x.AccountId == account.Id &&
                              x.CancellationDate == null,
                         stoppingToken);

if (booking == null)
{
    logger.LogWarning("User {userId} was not a participant in tour instance {tourInstanceId}", account.Id, request.TourInstanceId);
    return new OperationResult(ErrorCodes.ErrorCode_CannotRateNotBookedTour);
}

// Create the rating
var rate = new TourInstanceRate
{
    TourInstanceId = request.TourInstanceId,
    TouristAccountId = account.Id,
    Rate = request.Rate,
    TouristCommentary = request.Commentary,
```

```
            RatedTimeUtc = DateTime.UtcNow
        };

        await dbContext.TourInstanceRates.AddAsync(rate, stoppingToken);
        await dbContext.SaveChangesAsync(stoppingToken);

        return OperationResult.Success();
    }

    private class ChatGptResponse
    {
        public List<ChatGptChoice> Choices { get; set; } = new();
    }

    private class ChatGptChoice
    {
        public ChatGptMessage Message { get; set; } = new();
    }

    private class ChatGptMessage
    {
        public string Content { get; set; } = string.Empty;
    }

    private class ChatGptRecommendation
    {
        public List<int> RecommendedTourIds { get; set; } = new();
        public string Reason { get; set; } = string.Empty;
    }
}
```

HostedExtensions:

```
using System.Security.Claims;
using System.Text;
using System.Threading.Channels;
using Backend.Services;
using CloudinaryDotNet;
using Dipchik.BackgroundWorkers;
using Dipchik.Services;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Hosting.DbContexts;
using Microsoft.IdentityModel.Tokens;
using Shared.Model;
using Shared.Model.Dtos;
using StackExchange.Redis;

namespace Dipchik
{
```

```
    public static class HostedExtensions
    {
        public static WebApplication ConfigureServices(this WebApplicationBuilder builder)
        {
            var services = builder.Services;

            services.AddEndpointsApiExplorer();
            services.AddSwaggerGen();

            var eventsChannel = Channel.CreateUnbounded<EventDto>(new UnboundedChannelOptions
            {
                SingleReader = true
            });
            services.AddSingleton(eventsChannel);
            services.AddSingleton(eventsChannel.Writer);
            services.AddSingleton(eventsChannel.Reader);

            var sqlConnectionStr =
builder.Configuration.GetValue<string>("Databases:SqlConnection");
            var redisConnectionStr =
builder.Configuration.GetValue<string>("Databases:RedisConnection"); ;
            var cloudinaryConnectionStr =
builder.Configuration.GetValue<string>("Databases:CloudinaryConnection"); ;

            builder.Services.AddSingleton<IConnectionMultiplexer>(sp =>
                ConnectionMultiplexer.Connect(redisConnectionStr));
            services.AddSignalR().AddStackExchangeRedis(redisConnectionStr);
            services.AddDbContext<SqlContext>(options => options.UseNpgsql(sqlConnectionStr));
            services.AddScoped<AuthService>();
            services.AddScoped<LocalizationsService>();
            services.AddScoped<BookingService>();
            services.AddScoped<GuideService>();
            services.AddScoped<ToursService>();
            services.AddSingleton<Cloudinary>(x =>
            {
                var elem = new Cloudinary(cloudinaryConnectionStr);
                elem.Api.Secure = true;
                return elem;
            });
            services.AddSingleton<JwtTokenGenerator>();
            services.AddSingleton<SignalRService>();
            services.AddSingleton<CacheManager>();
            services.AddHttpContextAccessor();

            services.AddHostedService<EventProceeder>();

            var jwtKey = builder.Configuration.GetValue<string>("Auth:JwtKey");
            services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
                .AddJwtBearer(options =>
                {
                    options.TokenValidationParameters = new TokenValidationParameters
                    {
                        ValidateLifetime = true,
                        ValidateIssuer = false,
                        ValidateAudience = false,
                        ValidateIssuerSigningKey = true,
                        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtKey))
                    };
                    options.Events = new JwtBearerEvents
                    {
                        OnMessageReceived = context =>
                        {
                            var accessToken = context.Request.Query["access_token"];

                            if (!string.IsNullOrEmpty(accessToken) &&
                                context.HttpContext.Request.Path.StartsWithSegments("/messages"))
                            {
                                context.Token = accessToken;
                            }
                            return Task.CompletedTask;
                        }
                    };
                });

            services.AddAuthorization(options =>
            {
                foreach (AccountRolesEnum elem in Enum.GetValues<AccountRolesEnum>())
```

```
                {
                    if (elem == AccountRolesEnum.None)
                    {
                        continue;
                    }
                    options.AddPolicy(elem.ToString(), policy => policy.RequireAssertion(context =>
                    {
                        var rolesClaim = context.User.FindFirst(ClaimTypes.Role)?.Value;
                        if (string.IsNullOrEmpty(rolesClaim))
                            return false;
                        if (!int.TryParse(rolesClaim, out var roleFlags))
                            return false;

                        return (roleFlags & (int)elem) > 0;
                    }));
                }
            });
            services.AddSignalR();
            var webClientUrl = builder.Configuration.GetValue<string>("WebClientUrl");
            services.AddCors(options =>
            {
                options.AddPolicy("WebClient",
                    policy => policy.WithOrigins(webClientUrl)
                        .AllowAnyHeader()
                        .AllowAnyMethod()
                        .AllowCredentials());
            });

            return builder.Build();
        }
    }
}
```

Program:

```
using Serilog.Events;
using Serilog;
using Dipchik;
using Dipchik.Controllers;
using Shared.Model;

var builder = WebApplication.CreateBuilder(args);

builder.AddServiceDefaults();

Log.Logger = new LoggerConfiguration()
    .MinimumLevel.Debug()
    .MinimumLevel.Override("Microsoft", LogEventLevel.Information)
    .MinimumLevel.Override("System", LogEventLevel.Information)
    .WriteTo.Console()
    .Enrich.FromLogContext()
    .CreateLogger();

builder.Host.UseSerilog();
builder.AddRedisClient("cache");

var app = builder.ConfigureServices();

app.MapDefaultEndpoints();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseCors("WebClient");
app.MapHub<SignalRHub>("/messages");
app.UseHttpsRedirection();
app.UseAuthorization();

// Non auth controllers
var anonEPs = app.MapGroup("/").AllowAnonymous().WithOpenApi();
anonEPs.AddAuthController();
anonEPs.AddLocalizationController();
```

```
anonEPs.AddToursController();
if (app.Environment.IsDevelopment())
{
    anonEPs.AddTestController();
}

// Auth controllers
var authEPs = app.MapGroup("/").RequireAuthorization().WithOpenApi();
authEPs.AddAccountController(AccountRolesEnum.Client);
authEPs.AddBookingController(AccountRolesEnum.Client);
authEPs.AddAdminController(AccountRolesEnum.Client, AccountRolesEnum.Modify);
authEPs.AddGuideController(AccountRolesEnum.Client, AccountRolesEnum.Guide);
app.Run();
```

WebUtils:

```
namespace Dipchik;

public static class WebUtils
{
    public static string? UserId(this HttpContext context)
        => context.User.Identity?.Name;
}
```

Client:

ConfirmDialog:

```
<MudDialog>
    <DialogContent>
        <MudText>@ContentText</MudText>
    </DialogContent>
    <DialogActions>
        <MudButton OnClick="Cancel">@CancelButtonText</MudButton>
        <MudButton Color=@Color Variant="Variant.Filled"
OnClick="Submit">@ConfirmButtonText</MudButton>
    </DialogActions>
</MudDialog>

@code {
    [CascadingParameter] IMudDialogInstance MudDialog { get; set; } = null!;
    [Parameter] public string ContentText { get; set; } = "";
    [Parameter] public string ConfirmButtonText { get; set; } = "Confirm";
    [Parameter] public string CancelButtonText { get; set; } = "Cancel";
    [Parameter] public Color Color { get; set; } = Color.Error;

    void Submit() => MudDialog.Close(DialogResult.Ok(true));
    void Cancel() => MudDialog.Close(DialogResult.Cancel());
}
```

RateTourDialog:

```
@using System.Net.Http.Headers
@using Shared.Model.Dtos
@inject ISnackbar Snackbar
@inject IConfiguration Configuration

<MudDialog>
    <DialogContent>
        <MudStack Spacing="3">
            <MudText Typo="Typo.h6">@globals.Localizations["MyBookings_RateTour"]</MudText>
            <MudText
Typo="Typo.body2">@globals.Localizations["MyBookings_RateTourDescription"]</MudText>

            <MudRating @bind-SelectedValue="Rating"
                       Color="Color.Primary"
                       Size="Size.Large"
                       MaxValue="5" />

            <MudTextField @bind-Value="Comment"
                          Label="@globals.Localizations["MyBookings_RateTourComment"]"
                          Lines="3"
                          Variant="Variant.Outlined" />
        </MudStack>
    </DialogContent>
    <DialogActions>
```

```
        <MudButton OnClick="Cancel">@globals.Localizations["UI_Cancel"]</MudButton>
        <MudButton Color="Color.Primary"
                   Variant="Variant.Filled"
                   OnClick="Submit"
                   Disabled="@(Rating == 0 || string.IsNullOrWhiteSpace(Comment))">
            @globals.Localizations["UI_Submit"]
        </MudButton>
    </DialogActions>
</MudDialog>

@code {
    [CascadingParameter] IMudDialogInstance MudDialog { get; set; } = null!;
    [Parameter] public int TourInstanceId { get; set; }

    private int Rating { get; set; }
    private string Comment { get; set; } = string.Empty;

    private void Cancel() => MudDialog.Cancel();

    private async Task Submit()
    {
        try
        {
            var serverUrl = Configuration.GetValue<string>("ServerUrl");
            using var cli = new HttpClient();
            cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
globals.Token);

            var request = new TourRateDto
            {
                TourInstanceId = TourInstanceId,
                Rate = (byte)(Rating * 10), // Convert to 0-50 scale
                Commentary = Comment
            };

            var response = await cli.PostAsJsonAsync($"{serverUrl}/Tours/RateTour", request);

            if (response.IsSuccessStatusCode)
            {
                Snackbar.Add(globals.Localizations["Messages_RatingSubmitted"], Severity.Success);
                MudDialog.Close(DialogResult.Ok(true));
            }
            else
            {
                var error = await response.Content.ReadAsStringAsync();
                Snackbar.Add($"{globals.Localizations["Messages_FailedToSubmitRating"]} - {error}",
Severity.Error);
            }
        }
        catch (Exception ex)
        {
            Snackbar.Add(globals.Localizations["Messages_FailedToSubmitRating"], Severity.Error);
        }
    }
}
```

TourDialog:

```
@using Shared.Model
@using Shared.Model.Dtos
@inject HttpClient Http
@inject ISnackbar Snackbar

<MudDialog>
    <DialogContent>
        <MudForm @ref="form" @bind-IsValid="@success">
            <MudTextField T="string" Label="Title" @bind-Value="Tour.Title" Required="true"
RequiredError="Title is required!" />
            <MudTextField T="string" Label="Description" @bind-Value="Tour.Description" Lines="3"
Required="true" RequiredError="Description is required!" />
            <MudTextField T="string" Label="Image URL" @bind-Value="Tour.ImageUrl" Required="true"
RequiredError="Image URL is required!" />

            <MudNumericField T="decimal" Label="Price" @bind-Value="Tour.Price" Required="true"
RequiredError="Price is required!"
                             Min="0" Max="100000" Step="100" Format="C2" />
```

```
            <MudNumericField T="int" Label="Duration (days)" @bind-Value="Tour.DurationDays"
Required="true" RequiredError="Duration is required!"
                             Min="1" Max="60" Step="1" />

            <MudSelect T="TourTypeEnum" Label="Tour Type" @bind-Value="Tour.TourType"
Required="true" RequiredError="Tour type is required!">
                @foreach (TourTypeEnum type in Enum.GetValues(typeof(TourTypeEnum)))
                {
                    @if (type != TourTypeEnum.None)
                    {
                        <MudSelectItem Value="@type">@type</MudSelectItem>
                    }
                }
            </MudSelect>

            <MudSelect T="TourClassificationEnum" Label="Classification" @bind-
Value="Tour.Classification" Required="true" RequiredError="Classification is required!">
                @foreach (TourClassificationEnum classification in
Enum.GetValues(typeof(TourClassificationEnum)))
                {
                    <MudSelectItem Value="@classification">@classification</MudSelectItem>
                }
            </MudSelect>

            <MudSelect T="SpecialOfferEnum" Label="Special Offer" @bind-Value="Tour.SpecialOffers">
                @foreach (SpecialOfferEnum offer in Enum.GetValues(typeof(SpecialOfferEnum)))
                {
                    <MudSelectItem Value="@offer">@offer</MudSelectItem>
                }
            </MudSelect>

            <MudSwitch T="bool" @bind-Checked="Tour.WithGuide" Label="With Guide"
Color="Color.Primary" />

            <MudStack Row="true" Spacing="2" AlignItems="AlignItems.Center">
                <MudAutocomplete T="int" Label="Add Location"
                                 SearchFunc="@SearchLocations"
                                 ToStringFunc="@(id => globals.LocaleLocalizations[id].City)"
                                 @bind-Value="selectedLocationId"
                                 Placeholder="Type to search locations..." />
                <MudButton OnClick=@(() =>OnLocationSelected())
                           Disabled="@(selectedLocationId == 0)"
                           Variant="Variant.Filled"
                           Color="Color.Primary"
                           Class="mt-4">
                    Add
                </MudButton>
            </MudStack>

            <MudPaper Class="pa-2 mt-2" Style="max-height: 200px; overflow-y: auto;">
                @if (selectedLocations.Any())
                {
                    <MudList T="int" Dense="true">
                        @foreach (var locationId in selectedLocations)
                        {
                            <MudListItem>
                                <MudStack Row="true" Spacing="2" AlignItems="AlignItems.Center">

<MudText>@globals.LocaleLocalizations[locationId].City</MudText>
                                    <MudSpacer />
                                    <MudIconButton Icon="@Icons.Material.Filled.Delete"
Size="Size.Small" OnClick="@(() => RemoveLocation(locationId))" />
                                </MudStack>
                            </MudListItem>
                        }
                    </MudList>
                }
                else
                {
                    <MudText Color="Color.Secondary" Class="text-center">No locations
selected</MudText>
                }
            </MudPaper>
        </MudForm>
    </DialogContent>
    <DialogActions>
        <MudButton OnClick="Cancel">Cancel</MudButton>
```

```
        <MudButton Color="Color.Primary" OnClick="Submit" Disabled="@(!success)">Submit</MudButton>
    </DialogActions>
</MudDialog>

@code {
    [CascadingParameter] IMudDialogInstance MudDialog { get; set; } = null!;
    [Parameter] public TourDto Tour { get; set; } = new();
    [Parameter] public bool IsNew { get; set; }

    private MudForm form = null!;
    private bool success;
    private List<int> selectedLocations = new();
    private int selectedLocationId;

    protected override void OnInitialized()
    {
        if (!IsNew)
        {
            selectedLocations = Tour.Locations;
        }
    }

    private async Task<IEnumerable<int>> SearchLocations(string searchText, CancellationToken
stoppingToken)
    {
        if (string.IsNullOrWhiteSpace(searchText))
            return Array.Empty<int>();

        return globals.LocaleLocalizations
            .Where(l => l.Value.City.Contains(searchText, StringComparison.OrdinalIgnoreCase))
            .Select(l => l.Key)
            .Where(id => !selectedLocations.Contains(id));
    }

    private void OnLocationSelected()
    {
        if (!selectedLocations.Contains(selectedLocationId))
        {
            selectedLocations.Add(selectedLocationId);
        }
    }

    private void RemoveLocation(int locationId)
    {
        selectedLocations.Remove(locationId);
    }

    private void Cancel() => MudDialog.Close();

    private async Task Submit()
    {
        Tour.Locations = selectedLocations;

        try
        {
            if (IsNew)
            {
                await Http.PostAsJsonAsync("api/tour", Tour);
                Snackbar.Add("Tour created successfully", Severity.Success);
            }
            else
            {
                await Http.PutAsJsonAsync($"api/tour/{Tour.Id}", Tour);
                Snackbar.Add("Tour updated successfully", Severity.Success);
            }

            MudDialog.Close(DialogResult.Ok(true));
        }
        catch (Exception ex)
        {
            Snackbar.Add("Error saving tour", Severity.Error);
        }
    }
}
```

TourInstanceDialog:

```
@using Client.Pages
@using Shared.Model
@using Shared.Model.Dtos
@inject HttpClient Http
@inject ISnackbar Snackbar

<MudDialog>
    <DialogContent>
        <MudForm @ref="form" @bind-IsValid="@success">
            @if (IsNew)
            {
                <MudSelect T="int" Label="Tour" @bind-Value="Instance.TourId" Required="true"
RequiredError="Tour is required!">
                    @foreach (var tour in Tours)
                    {
                        <MudSelectItem Value="@tour.Id">@tour.Title</MudSelectItem>
                    }
                </MudSelect>
            }

            <MudDatePicker Label="Start Date" @bind-Date="Instance.StartDate" Required="true"
RequiredError="Start date is required!" />
            <MudDatePicker Label="End Date" @bind-Date="Instance.EndDate" Required="true"
RequiredError="End date is required!" />

            <MudNumericField T="int" Label="Max Participants" @bind-
Value="Instance.MaxParticipants" Required="true" RequiredError="Max participants is required!"
            Min="1" Max="100" Step="1" />

            <MudNumericField T="decimal" Label="Price" @bind-Value="Instance.Price" Required="true"
RequiredError="Price is required!"
            Min="0" Max="100000" Step="100" Format="C2" />

            <MudSwitch Disabled="Instance.StartDate < DateTime.UtcNow" ValueChanged=ChangeStatus
T="bool" Label="Is Active" Color="Color.Primary" Value="Instance.Status !=
TourInstanceStatus.Cancelled" />
            <MudStack Row=true>
                <MudText>Status: </MudText>
                <MudChip T="string" Color="@GuideManagement.GetStatusColor(Instance.Status)"
Size="Size.Small">
                    @globals.Localizations[$"TourStatus_{Instance.Status}"]
                </MudChip>
            </MudStack>
        </MudForm>
    </DialogContent>
    <DialogActions>
        <MudButton OnClick="Cancel" ButtonType="ButtonType.Reset">Cancel</MudButton>
        <MudButton Color="Color.Primary" OnClick="Submit" Disabled="@(!success)">Submit</MudButton>
    </DialogActions>
</MudDialog>

@code {
    [CascadingParameter] IMudDialogInstance MudDialog { get; set; } = null!;
    [Parameter] public TourDescDto Instance { get; set; } = new();
    [Parameter] public bool IsNew { get; set; }

    private MudForm form = null!;
    private bool success;
    private List<TourDto> Tours { get; set; } = new();

    protected override async Task OnInitializedAsync()
    {
        if (IsNew)
        {
            Tours = await Http.GetFromJsonAsync<List<TourDto>>("api/tour/guide") ?? new();
        }
    }

    private void Cancel() => MudDialog.Cancel();

    private async Task Submit()
    {
        try
        {
            if (IsNew)
            {
                await Http.PostAsJsonAsync("api/tour-instance", Instance);
```

```
                Snackbar.Add("Tour instance created successfully", Severity.Success);
            }
            else
            {
                await Http.PutAsJsonAsync($"api/tour-instance/{Instance.Id}", Instance);
                Snackbar.Add("Tour instance updated successfully", Severity.Success);
            }

            MudDialog.Close(DialogResult.Ok(true));
        }
        catch (Exception ex)
        {
            Snackbar.Add("Error saving tour instance", Severity.Error);
        }
    }

    private void ChangeStatus(bool val) => Instance.IsCancelled = !Instance.IsCancelled;
}
```

Globals:

```
using Microsoft.JSInterop;
using System.Text.Json;
using Shared.Model;
using Shared.Model.Dtos;

namespace Client.Dtos;

public class Globals
{
    private readonly IConfiguration configuration;
    public event Action? OnChange;

    public void Notify() => OnChange?.Invoke();

    public bool IsLoggedIn => User is not null || !string.IsNullOrEmpty(Token);
    public AccountInfoDto? User { get; private set; } = null;
    public string? Token { get; set; } = null;
    public Dictionary<string, string> Localizations { get; set; } = null!;
    public Dictionary<int, LanguageLocationsDto> LocaleLocalizations { get; set; } = null!;

    public Globals(IConfiguration configuration)
    {
        this.configuration = configuration;
    }

    public async Task LoadLocalization(LocalizationCode oldLocale, LocalizationCode newLocale)
    {
        if (oldLocale == newLocale)
        {
            return;
        }
        var serverUrl = configuration.GetValue<string>("ServerUrl");
        using (var cli = new HttpClient())
        {
            var response = await
cli.GetAsync($"{serverUrl}/localization/LoadDisplayLocalization?localeCode={newLocale.ToString()}")
;
            var result = await response.Content.ReadAsStringAsync();
            if (response.StatusCode != System.Net.HttpStatusCode.OK)
            {
                return;
            }
            Localizations = JsonSerializer.Deserialize<Dictionary<string, string>>(result, new
JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });
        }
        using (var cli = new HttpClient())
        {
            var response = await
cli.GetAsync($"{serverUrl}/localization/LoadLocationLocalization?localeCode={newLocale.ToString()}"
);
            var result = await response.Content.ReadAsStringAsync();
            if (response.StatusCode != System.Net.HttpStatusCode.OK)
            {
```

```
                return;
            }

            LocaleLocalizations = JsonSerializer.Deserialize<Dictionary<int,
LanguageLocationsDto>>(result, new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });
        }

        Notify();
    }

    public async Task Logout(IJSRuntime jsRuntime)
    {
        User = null;
        Token = null;
        await jsRuntime.InvokeAsync<string>("localStorage.setItem", "userData", null);
        await jsRuntime.InvokeAsync<string>("localStorage.setItem", "userToken", null);
        Notify();
    }

    public async Task Login(IJSRuntime jsRuntime, AccountInfoDto user, string token)
    {
        await LoadLocalization(User?.Locale ?? LocalizationCode.None, user.Locale);
        User = user;
        Token = token;
        await jsRuntime.InvokeAsync<string>("localStorage.setItem", "userData",
JsonSerializer.Serialize(user));
        await jsRuntime.InvokeAsync<string>("localStorage.setItem", "userToken", token);
        Notify();
    }

    public async Task LoadUser(IJSRuntime jsRuntime)
    {
        var userData = await jsRuntime.InvokeAsync<string>("localStorage.getItem", "userData");
        Token = await jsRuntime.InvokeAsync<string>("localStorage.getItem", "userToken");
        if (!string.IsNullOrEmpty(userData))
        {
            User = JsonSerializer.Deserialize<AccountInfoDto>(userData);
        }
        await LoadLocalization(LocalizationCode.None, User?.Locale ?? LocalizationCode.ENG);
    }
    public async Task UpdateUser(IJSRuntime jsRuntime, AccountInfoDto user)
    {
        await LoadLocalization(User.Locale, user.Locale);
        User = user;
        await jsRuntime.InvokeAsync<string>("localStorage.setItem", "userData",
JsonSerializer.Serialize(User));
        Notify();
    }
}
```

Login:

```
@page "/login"
@using System.ComponentModel.DataAnnotations
@using System.Net.Http.Headers
@using System.Text.Json
@using Shared.Model.Dtos
@inject IConfiguration Configuration
@inject IJSRuntime jsRuntime
@inject NavigationManager NavigationManager

<h3>Login</h3>
<EditForm Model="@model" OnValidSubmit="OnValidSubmit">
    <DataAnnotationsValidator />
    <MudGrid>
        <MudItem xs="12" sm="12">
            <MudCard>
                <MudCardContent>
                    <MudTextField Label="First name" HelperText="Max. 8 characters"
                    @bind-Value="model.Username" For="@(() => model.Username)" />
                    <MudTextField Label="Password" HelperText="Choose a strong password" Class="mt-
3"
                    @bind-Value="model.Password" For="@(() => model.Password)"
InputType="InputType.Password" />
```

```
                </MudCardContent>
                <MudCardActions>
                    <MudButton ButtonType="ButtonType.Button" OnClick="OnValidLogin"
Variant="Variant.Filled" Color="Color.Primary" Class="ml-auto">Login</MudButton>
                </MudCardActions>
                <MudCardActions>
                    <MudButton ButtonType="ButtonType.Submit" Variant="Variant.Filled"
Color="Color.Primary" Class="ml-auto">Register</MudButton>
                </MudCardActions>
            </MudCard>
        </MudItem>
        <MudItem xs="12">
            <MudText Typo="Typo.body2" Align="Align.Center" Style="color:red">
                @errorMsg
            </MudText>
        </MudItem>
    </MudGrid>
</EditForm>
@code
{
    RegisterAccountForm model = new RegisterAccountForm();
    bool success;
    string errorMsg;

    public class RegisterAccountForm
    {
        [Required]
        //[StringLength(8, ErrorMessage = "Name length can't be more than 8.")]
        public string Username { get; set; }

        [Required]
        //[StringLength(30, ErrorMessage = "Password must be at least 8 characters long.",
MinimumLength = 8)]
        public string Password { get; set; }

    }

    private async Task OnValidLogin()
    {
        await Authenticate("login");
    }
    private async Task OnValidSubmit(EditContext context)
    {
        await Authenticate("register");
    }

    private async Task Authenticate(string endpoint)
    {
        string serverUrl = Configuration.GetValue<string>("ServerUrl");

        var token = await GetToken(serverUrl, endpoint);
        var accountData = await GetAccountData(serverUrl, token);
        if (token is null || accountData is null)
        {
            errorMsg = "Invalid username or password.";
            return;
        }

        await globals.Login(jsRuntime, accountData, token);

        success = true;
        StateHasChanged();
        NavigationManager.NavigateTo("/");
    }

    private async Task<string> GetToken(string serverUrl, string endpoint)
    {

        using var cli = new HttpClient();
        var response = await
cli.GetAsync($"{serverUrl}/auth/{endpoint}?login={model.Username}&password={model.Password}");
        if (response.StatusCode != System.Net.HttpStatusCode.OK)
        {
            errorMsg = response.ReasonPhrase;
            return null;
        }
```

```
        return (await response.Content.ReadAsStringAsync()).Trim('"');
    }

    private async Task<AccountInfoDto> GetAccountData(string serverUrl, string token)
    {

        using var cli = new HttpClient();
        cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);
        var response = await cli.GetAsync($"{serverUrl}/Account/GetAccountInfo");
        if (response.StatusCode != System.Net.HttpStatusCode.OK)
        {
            errorMsg = response.ReasonPhrase;
            return null;
        }

        var json = await response.Content.ReadAsStringAsync();
        return JsonSerializer.Deserialize<AccountInfoDto>(json, new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });
    }
}
```

MainLayout:

```
@inherits LayoutComponentBase
@inject IJSRuntime _jsRuntime

<MudThemeProvider />
<MudPopoverProvider />
<MudDialogProvider />
<MudSnackbarProvider />

<div class="page">
    <div class="sidebar">
        <NavMenu />
    </div>

    <main>
        <div class="top-row px-4">

            @if (globals.IsLoggedIn)
            {
                if (string.IsNullOrEmpty(globals.User!.AvatarUrl))
                {
                    <MudAvatar Color="Color.Primary"
Variant="Variant.Filled">@globals.User!.Username!.First()</MudAvatar>
                }
                else
                {
                    <MudAvatar>
                        <MudImage Src=@globals.User!.AvatarUrl></MudImage>
                    </MudAvatar>
                }
                <MudText Class="ml-2">@($"{@globals.Localizations["UI_WelcomeBack"]},
{globals.User!.Username}!")</MudText>

                <NavLink class="nav-link" href="accountedit" Match="NavLinkMatch.All">
                    <span class="bi bi-house-door-fill-nav-menu" aria-hidden="true"></span>
@globals.Localizations["UI_Edit"]
                </NavLink>
                <MudLink OnClick="Logout">@globals.Localizations["UI_Logout"]</MudLink>
            }
            else
            {
                <NavLink class="nav-link" href="login" Match="NavLinkMatch.All">
                    <span class="bi bi-house-door-fill-nav-menu" aria-hidden="true"></span>
@globals.Localizations["UI_Login"]
                </NavLink>
            }
        </div>

        <article class="content px-4">
            @Body
        </article>
    </main>
</div>
```

```
@code {
    protected override void OnInitialized()
    {
        globals.OnChange += StateHasChanged;
    }

    public void Dispose()
    {
        globals.OnChange -= StateHasChanged;
    }


    private async Task Logout()
    {
        await globals.Logout(_jsRuntime);
    }
}
```

NavMenu:

```
@using Shared.Model
<div class="top-row ps-3 navbar navbar-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="">Client</a>
        <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
            <span class="navbar-toggler-icon"></span>
        </button>
    </div>
</div>

<div class="@NavMenuCssClass nav-scrollable" @onclick="ToggleNavMenu">
    <nav class="flex-column">
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="tours">
                <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span>
@globals.Localizations["Navbar_Tours"]
            </NavLink>
        </div>
        @if (globals.IsLoggedIn)
        {
            <div class="nav-item px-3">
                <NavLink class="nav-link" href="my-bookings">
                    <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span>
@globals.Localizations["Navbar_MyBookings"]
                </NavLink>
            </div>
            if (globals.User!.HasRole(AccountRolesEnum.Modify))
            {
                <div class="nav-item px-3">
                    <NavLink class="nav-link" href="localizations">
                        <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span>
@globals.Localizations["Navbar_ChangeLocalization"]
                    </NavLink>
                </div>
            }
            if (globals.User!.HasRole(AccountRolesEnum.Guide))
            {
                <div class="nav-item px-3">
                    <NavLink class="nav-link" href="guide-management">
                        <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span>
@globals.Localizations["Navbar_GuideManagement"]
                    </NavLink>
                </div>
            }
        }
    </nav>
</div>

@code {
    private bool collapseNavMenu = true;

    private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

    private void ToggleNavMenu()
    {
        collapseNavMenu = !collapseNavMenu;
```

```
    }

    protected override void OnInitialized()
    {
        globals.OnChange += StateHasChanged;
    }

    public void Dispose()
    {
        globals.OnChange -= StateHasChanged;
    }
}
```

AccountEdit:

```
@page "/accountedit"
@using System.Net.Http.Headers
@using System.Text.Json
@using System.Web
@using Shared.Model
@using Shared.Model.Dtos
@inject NavigationManager Navigation
@inject ISnackbar Snackbar
@inject IConfiguration Configuration
@inject IJSRuntime jsRuntime

<MudPaper Class="max-w-xl mx-auto pa-10" Elevation="6">
    <MudText Typo="Typo.h6" Align="Align.Center" Class="mb-
4">@globals.Localizations["AccountEdit_Title"]</MudText>

    <div>

        <MudForm @ref="_form">
            <div class="d-flex flex-grow-1">
                <div class="mr-15">
                    <label for="avatarUpload" class="block">
                        <MudAvatar @onmouseenter=@(() => showOverlay = true) @onmouseleave=@(() =>
showOverlay = false) Size="Size.Large" Image="@AvatarPreviewUrl" Style="height:200px;width:200px"
Class="w-full h-full rounded-full cursor-pointer object-cover">
                            <MudImage Src="@AvatarPreviewUrl" Size="Size.Large" Class="w-full h-
full rounded-full cursor-pointer"/>
                            <MudOverlay Visible="showOverlay" Absolute=true Class="cursor-pointer"
LightBackground/>
                            @if (showOverlay)
                            {
                                <MudIcon @bind-Visible="showOverlay"
Icon="@Icons.Material.Filled.Add" Color="Color.Dark" Size="Size.Large" Class="absolute cursor-
pointer z-100"/>
                            }
                        </MudAvatar>
                    </label>
                    <InputFile id="avatarUpload" style="display:none;"
OnChange="HandleAvatarUpload" accept="image/x-png,image/jpeg"/>
                </div>
                <div class="flex-1">
                    <MudTextField Label=@globals.Localizations["AccountEdit_Username"]  @bind-
Value="Username" Required="true"
RequiredError=@globals.Localizations["AccountEdit_UsernameIsRequired"]/>

                    <MudSelect @bind-Value="Locale" Class="mt-3"
Label=@globals.Localizations["AccountEdit_SelectLanguage"]
OpenIcon="@Icons.Material.Filled.Language" AdornmentColor="Color.Secondary">
                        @foreach (var item in Enum.GetValues<LocalizationCode>())
                        {
                            if (item == LocalizationCode.None)
                                continue;
                            <MudSelectItem Value="@item">@item.ToString()</MudSelectItem>
                        }
                    </MudSelect>
                </div>
            </div>

            <MudButton Variant="Variant.Filled" Color="Color.Primary" Class="mt-4"
OnClick="HandleValidSubmit" Type="Submit">
                @globals.Localizations["UI_SaveChanges"]
            </MudButton>
```

```
        </MudForm>
    </div>


    <MudDivider />

    <MudButton Variant="Variant.Outlined" Color="Color.Error" Class="mt-4"
OnClick="OpenDeleteDialog">
        @globals.Localizations["AccountEdit_DeleteAccount"]
    </MudButton>
</MudPaper>

<MudDialog @bind-Visible="_deleteDialogOpen">
    <DialogContent>
        <MudText Typo="Typo.h6">@globals.Localizations["AccountEdit_PromptDeleteMsg"]</MudText>
    </DialogContent>
    <DialogActions>
        <MudButton Color="Color.Default"
OnClick="CloseDeleteDialog">@globals.Localizations["UI_Cancel"]</MudButton>
        <MudButton Color="Color.Error"
OnClick="DeleteAccount">@globals.Localizations["UI_Delete"]</MudButton>
    </DialogActions>
</MudDialog>
@code {
    private bool showOverlay = false;

    private MudForm _form;
    private IBrowserFile? _selectedFile;
    private InputFile? _fileInput;

    private string Username;
    private LocalizationCode Locale = LocalizationCode.ENG;
    private string AvatarPreviewUrl =
"https://res.cloudinary.com/da7kbbrap/image/upload/360_F_303981738_1s8t2JvUDyfBKsHUmR01LZhEJBsJTgML
_itthcj.jpg";
    private byte[] CustomAvatarBytes = [];
    private bool _deleteDialogOpen = false;

    protected override async Task OnInitializedAsync()
    {
        Username = globals.User!.Username;
        Locale = globals.User.Locale;
        if (globals.User.AvatarUrl is not null)
            AvatarPreviewUrl = globals.User.AvatarUrl;
    }

    private async Task HandleAvatarUpload(InputFileChangeEventArgs e)
    {
        var file = e.File;
        await using var stream = file.OpenReadStream();
        CustomAvatarBytes = new byte[file.Size];
        await stream.ReadAsync(CustomAvatarBytes);
        var imageStr = Convert.ToBase64String(CustomAvatarBytes);

        AvatarPreviewUrl = $"data:{file.ContentType};base64,{imageStr}";
        StateHasChanged();
    }

    private async Task HandleValidSubmit()
    {
        var serverUrl = Configuration.GetValue<string>("ServerUrl");
        using var cli = new HttpClient();
        cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
globals.Token);
        var response = await
cli.PostAsJsonAsync($"{serverUrl}/Account/UpdateUser?Username={Username}&localeCode={Locale}",
CustomAvatarBytes);
        var result = await response.Content.ReadAsStringAsync();
        if (response.StatusCode != System.Net.HttpStatusCode.OK)
        {
            Snackbar.Add($"{@globals.Localizations["Messages_ErrorOnUpdate"]} - {result}",
Severity.Error);
            return;
        }
        var userData = JsonSerializer.Deserialize<AccountInfoDto>(result, new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
```

```
            });
        AvatarPreviewUrl = HttpUtility.UrlDecode(userData.AvatarUrl).Trim('\"');
        await globals.UpdateUser(jsRuntime, userData);
        Snackbar.Add(@globals.Localizations["Messages_AccountSettingsUpdated"], Severity.Success);
    }

    private void OpenDeleteDialog() => _deleteDialogOpen = true;
    private void CloseDeleteDialog() => _deleteDialogOpen = false;

    private async Task DeleteAccount()
    {
        var serverUrl = Configuration.GetValue<string>("ServerUrl");
        using var cli = new HttpClient();
        cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
globals.Token);
        var response = await cli.GetAsync($"{serverUrl}/Account/DeleteAccount");
        var result = await response.Content.ReadAsStringAsync();
        if (response.StatusCode != System.Net.HttpStatusCode.OK)
        {
            Snackbar.Add($"{@globals.Localizations["Messages_AccountDeletionError"]}- {result}",
Severity.Error);
            return;
        }
        Snackbar.Add(@globals.Localizations["Messages_AccountDeleted"], Severity.Warning);
        globals.Logout(jsRuntime);
        Navigation.NavigateTo("/");
    }
}
```

GuideManagement:

```
@page "/guide-management"

@using System.Net.Http.Headers

@using System.Text.Json

@using Client.Components

@using Shared.Model

@using Shared.Model.Dtos

@inject HttpClient Http

@inject ISnackbar Snackbar

@inject IDialogService DialogService

@inject IConfiguration Configuration


<MudContainer MaxWidth="MaxWidth.Large" Class="mt-4">

  <MudPaper Class="pa-4">

    <MudText Typo="Typo.h4" Class="mb-4">Guide Management</MudText>


    <MudTabs Elevation="0" Rounded="true" ApplyEffectsToContainer="true" Class="mt-4">

      <MudTabPanel Text="My Tours">

        <MudButton Variant="Variant.Filled" Color="Color.Primary" OnClick="OpenCreateTourDialog" Class="mb-4">

          <MudIcon Icon="@Icons.Material.Filled.Add" Class="mr-2" />Create New Tour

        </MudButton>


        <MudTable Items="@Tours" Dense="true" Hover="true" Bordered="true" Striped="true" Loading="@_loading"

            LoadingProgressColor="Color.Info" Filter="new Func<TourDto,bool>(FilterFunc)">

          <HeaderContent>
```

```
                <MudTh><MudTableSortLabel SortBy="new Func<TourDto, object>(x=>x.Title)">Title</MudTableSortLabel></MudTh>

                <MudTh><MudTableSortLabel SortBy="new Func<TourDto, object>(x=>x.Price)">Price</MudTableSortLabel></MudTh>

                <MudTh><MudTableSortLabel SortBy="new Func<TourDto,
object>(x=>x.DurationDays)">Duration</MudTableSortLabel></MudTh>

                <MudTh><MudTableSortLabel SortBy="new Func<TourDto, object>(x=>x.TourType)">Type</MudTableSortLabel></MudTh>

                <MudTh><MudTableSortLabel SortBy="new Func<TourDto,
object>(x=>x.Classification)">Classification</MudTableSortLabel></MudTh>

                <MudTh>Locations</MudTh>

                <MudTh>Status</MudTh>

                <MudTh>Instances</MudTh>

                <MudTh>Actions</MudTh>

            </HeaderContent>

            <RowTemplate>

                <MudTd DataLabel="Title">@context.Title</MudTd>

                <MudTd DataLabel="Price">$@context.Price</MudTd>

                <MudTd DataLabel="Duration">@context.DurationDays days</MudTd>

                <MudTd DataLabel="Type">@context.TourType</MudTd>

                <MudTd DataLabel="Classification">@context.Classification</MudTd>

                <MudTd DataLabel="Locations">

                    @string.Join(", ", context.Locations.Select(l => globals.LocaleLocalizations[l].City))

                </MudTd>

                <MudTd DataLabel="Instances">

                    @GetTourInstanceCount(context.Id)

                </MudTd>

                <MudTd>

                    <MudStack Row="true" Spacing="2">

                        <MudIconButton Icon="@Icons.Material.Filled.Edit" OnClick="@(() => OpenEditTourDialog(context))" />

                        <MudIconButton Icon="@Icons.Material.Filled.Delete"

                                OnClick="@(() => DeleteTour(context))"

                                Disabled="@(HasTourInstances(context.Id))" />

                        <MudIconButton Icon="@Icons.Material.Filled.Add"

                                OnClick="@(() => OpenCreateInstanceDialog(context))" />

                    </MudStack>

                </MudTd>

            </RowTemplate>

            <PagerContent>

                <MudTablePager />

            </PagerContent>

        </MudTable>

    </MudTabPanel>
```

```
        <MudTabPanel Text="Tour Instances">
            <MudTable Items="@TourInstances" Dense="true" Hover="true" Bordered="true" Striped="true" Loading="@_loading"
                      LoadingProgressColor="Color.Info" Filter="new Func<TourDescDto,bool>(FilterInstanceFunc)">
                <HeaderContent>
                    <MudTh><MudTableSortLabel SortBy="new Func<TourDescDto, object>(x=>x.Title)">Tour</MudTableSortLabel></MudTh>
                    <MudTh><MudTableSortLabel SortBy="new Func<TourDescDto, object>(x=>x.StartDate)">Start
Date</MudTableSortLabel></MudTh>
                    <MudTh><MudTableSortLabel SortBy="new Func<TourDescDto, object>(x=>x.EndDate)">End
Date</MudTableSortLabel></MudTh>
                    <MudTh><MudTableSortLabel SortBy="new Func<TourDescDto,
object>(x=>x.CurrentParticipants)">Participants</MudTableSortLabel></MudTh>
                    <MudTh><MudTableSortLabel SortBy="new Func<TourDescDto,
object>(x=>x.Status)">Status</MudTableSortLabel></MudTh>
                    <MudTh>Actions</MudTh>
                </HeaderContent>
                <RowTemplate>
                    <MudTd DataLabel="Tour">@context.Title</MudTd>
                    <MudTd DataLabel="Start Date">@context.StartDate.Value.ToShortDateString()</MudTd>
                    <MudTd DataLabel="End Date">@context.EndDate.Value.ToShortDateString()</MudTd>
                    <MudTd DataLabel="Participants">@context.CurrentParticipants/@context.MaxParticipants</MudTd>
                    <MudTd DataLabel="Status">
                        <MudChip T="string" Color="@GetStatusColor(context.Status)" Size="Size.Small">
                            @globals.Localizations[$"TourStatus_{context.Status}"]
                        </MudChip>
                    </MudTd>
                    <MudTd>
                        <MudStack Row="true" Spacing="2">
                            @if (context.Status == TourInstanceStatus.Scheduled || context.Status == TourInstanceStatus.Cancelled)
                            {
                                <MudIconButton Icon="@Icons.Material.Filled.Edit" OnClick="@(() => OpenEditInstanceDialog(context))" />
                                <MudIconButton Icon="@Icons.Material.Filled.Cancel" OnClick="@(() => CancelInstance(context))" />
                            }
                        </MudStack>
                    </MudTd>
                </RowTemplate>
                <PagerContent>
                    <MudTablePager />
                </PagerContent>
            </MudTable>
        </MudTabPanel>
    </MudTabs>
</MudPaper>
```

```
</MudContainer>

@code {
    private bool _loading = true;
    private List<TourDto> Tours { get; set; } = new();
    private List<TourDescDto> TourInstances { get; set; } = new();
    private TourDto? _selectedTour;
    private TourDescDto? _selectedInstance;

    protected override async Task OnInitializedAsync()
    {
        await LoadData();
    }

    private async Task LoadData()
    {
        try
        {
            _loading = true;
            var serverUrl = Configuration.GetValue<string>("ServerUrl");
            using (var cli = new HttpClient())
            {
                cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", globals.Token);
                var response = await cli.GetAsync($"{serverUrl}/Guide/GetTours");
                var result = await response.Content.ReadAsStringAsync();
                if (response.StatusCode != System.Net.HttpStatusCode.OK)
                {
                    Snackbar.Add($"{@globals.Localizations["Messages_FailedToGetData"]} - {result}", Severity.Error);
                    return;
                }
                Tours = JsonSerializer.Deserialize<List<TourDto>>(result, new JsonSerializerOptions
                {
                    PropertyNameCaseInsensitive = true
                }) ?? new();
            }
            using (var cli = new HttpClient())
            {
                cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", globals.Token);
                var response = await cli.GetAsync($"{serverUrl}/Guide/GetTourInstances");
                var result = await response.Content.ReadAsStringAsync();
```

```
            if (response.StatusCode != System.Net.HttpStatusCode.OK)
            {
                Snackbar.Add($"{@globals.Localizations["Messages_FailedToGetData"]} - {result}", Severity.Error);
                return;
            }
            TourInstances = JsonSerializer.Deserialize<List<TourDescDto>>(result, new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            }) ?? new();
        }
    }
    catch (Exception ex)
    {
        Snackbar.Add("Error loading data", Severity.Error);
    }
    finally
    {
        _loading = false;
    }
}

private bool FilterFunc(TourDto tour)
{
    if (string.IsNullOrWhiteSpace(_searchString))
        return true;

    return tour.Title.Contains(_searchString, StringComparison.OrdinalIgnoreCase)
        || tour.Locations.Any(l => globals.LocaleLocalizations[l].City.Contains(_searchString, StringComparison.OrdinalIgnoreCase));
}

private bool FilterInstanceFunc(TourDescDto instance)
{
    if (string.IsNullOrWhiteSpace(_searchString))
        return true;

    return instance.Title.Contains(_searchString, StringComparison.OrdinalIgnoreCase);
}

private string _searchString = "";
```

```
public static Color GetStatusColor(TourInstanceStatus status) => status switch
{
    TourInstanceStatus.Scheduled => Color.Info,
    TourInstanceStatus.Cancelled => Color.Error,
    TourInstanceStatus.Completed => Color.Success,
    _ => Color.Default
};

private int GetTourInstanceCount(int tourId)
{
    return TourInstances.Count(i => i.TourId == tourId);
}

private bool HasTourInstances(int tourId)
{
    return TourInstances.Any(i => i.TourId == tourId);
}

private async Task OpenCreateTourDialog()
{
    var parameters = new DialogParameters
    {
        ["Tour"] = new TourDto(),
        ["IsNew"] = true
    };

    var dialog = await DialogService.ShowAsync<TourDialog>("Create New Tour", parameters);
    var result = await dialog.Result;

    if (!result.Canceled)
    {
        await LoadData();
    }
}

private async Task OpenEditTourDialog(TourDto tour)
{
    var parameters = new DialogParameters
    {
        ["Tour"] = tour,
```

```
        ["IsNew"] = false
    };

    var dialog = await DialogService.ShowAsync<TourDialog>("Edit Tour", parameters);
    var result = await dialog.Result;

    if (!result.Canceled)
    {
        await LoadData();
    }
}

private async Task OpenCreateInstanceDialog(TourDto tour)
{
    var parameters = new DialogParameters
    {
        ["Instance"] = new TourDescDto
        {
            TourId = tour.Id,
            Title = tour.Title,
            StartDate = DateTime.Today.AddDays(1),
            EndDate = DateTime.Today.AddDays(1 + tour.DurationDays),
            MaxParticipants = 20,
            Price = tour.Price,
            IsCancelled = false
        },
        ["IsNew"] = true
    };

    var dialog = await DialogService.ShowAsync<TourInstanceDialog>("Create New Tour Instance", parameters);
    var result = await dialog.Result;

    if (!result.Canceled)
    {
        await LoadData();
    }
}

private async Task OpenEditInstanceDialog(TourDescDto instance)
{
```

```
    var parameters = new DialogParameters
    {
        ["Instance"] = instance,
        ["IsNew"] = false
    };

    var dialog = await DialogService.ShowAsync<TourInstanceDialog>("Edit Tour Instance", parameters);
    var result = await dialog.Result;

    if (!result.Canceled)
    {
        await LoadData();
    }
}

private async Task DeleteTour(TourDto tour)
{
    if (TourInstances.Any(i => i.TourId == tour.Id))
    {
        Snackbar.Add(globals.Localizations["Messages_CannotDeleteTourWithInstances"], Severity.Warning);
        return;
    }

    var parameters = new DialogParameters
    {
        ["ContentText"] = $"Are you sure you want to delete the tour '{tour.Title}'?",
        ["ButtonText"] = "Delete",
        ["Color"] = Color.Error
    };

    var dialog = await DialogService.ShowAsync<ConfirmDialog>("Delete Tour", parameters);
    var result = await dialog.Result;

    if (!result.Canceled)
    {
        try
        {
            var serverUrl = Configuration.GetValue<string>("ServerUrl");
            using (var cli = new HttpClient())
            {
```

```
            cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", globals.Token);
            var response = await cli.DeleteAsync($"{serverUrl}/Guide/DeleteTour/{tour.Id}");
            var res = await response.Content.ReadAsStringAsync();
            if (response.StatusCode != System.Net.HttpStatusCode.OK)
            {
                Snackbar.Add($"{@globals.Localizations["Messages_FailedToDelete"]} - {res}", Severity.Error);
                return;
            }
        }
        await LoadData();
        Snackbar.Add(globals.Localizations["Messages_TourDeleted"], Severity.Success);
    }
    catch (Exception ex)
    {
        Snackbar.Add(globals.Localizations["Messages_FailedToDelete"], Severity.Error);
    }
  }
}

private async Task CancelInstance(TourDescDto instance)
{
  var parameters = new DialogParameters
  {
    ["ContentText"] = $"Are you sure you want to cancel this tour instance?",
    ["ButtonText"] = "Cancel Tour",
    ["Color"] = Color.Warning
  };

  var dialog = await DialogService.ShowAsync<ConfirmDialog>("Cancel Tour Instance", parameters);
  var result = await dialog.Result;

  if (!result.Canceled)
  {
    try
    {
      await Http.PutAsync($"api/tour-instance/{instance.Id}/cancel", null);
      await LoadData();
      Snackbar.Add("Tour instance cancelled successfully", Severity.Success);
    }
    catch (Exception ex)
```

```
        {
            Snackbar.Add("Error cancelling tour instance", Severity.Error);
        }
    }
  }
}
```

LocalizationManager:

```
@page "/localizations"
@using System.Net.Http.Headers
@using System.Text.Json
@using Shared.Model.Dtos
@inject HttpClient Http
@inject ISnackbar Snackbar
@inject IConfiguration Configuration

<MudPaper Class="p-4 max-w-7xl mx-auto mt-4 shadow-md">
    <MudText Typo="Typo.h5">@globals.Localizations["LocalizationManager_Title"]</MudText>

    <MudTable Items="@Entries" Hover="true" Bordered="true" Class="mt-4">
        <HeaderContent>
            <MudTh>@globals.Localizations["LocalizationManager_Placeholder"]</MudTh>
            @foreach (var lang in Languages)
            {
                <MudTh>@lang.ToUpper()</MudTh>
            }
            <MudTh></MudTh>
        </HeaderContent>
        <RowTemplate>
            <MudTd>
                <MudTextField T="string" @bind-Value="context.Placeholder" Immediate="true" />
            </MudTd>
            @foreach (var lang in Languages)
            {
                <MudTd>
                    <MudTextField T="string" @bind-Value="context.Translations[lang]"
Immediate="true" />
                </MudTd>
            }
            <MudTd>
                <MudIconButton Icon="@Icons.Material.Filled.Delete" Color="Color.Error"
OnClick="@(() => RemoveRow(context))" />
            </MudTd>
        </RowTemplate>
    </MudTable>

    <MudButton Class="mt-4" Variant="Variant.Outlined" OnClick="AddRow">
        <MudIcon Icon="@Icons.Material.Filled.Add" />
@globals.Localizations["LocalizationManager_AddRow"]
    </MudButton>

    <MudButton Class="mt-4 ml-2" Variant="Variant.Filled" Color="Color.Primary" OnClick="SaveAll">
        <MudIcon Icon="@Icons.Material.Filled.Save" /> @globals.Localizations["UI_SaveChanges"]
    </MudButton>
</MudPaper>

@code {
    private List<LocalizationManagerDto> Entries = [];

    private string[] Languages = [];

    protected override async Task OnInitializedAsync()
    {
        var serverUrl = Configuration.GetValue<string>("ServerUrl");
        using var cli = new HttpClient();
        cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
globals.Token);
        var response = await cli.GetAsync($"{serverUrl}/admin/getAllDisplayLocalizations");
        var result = await response.Content.ReadAsStringAsync();
        if (response.StatusCode != System.Net.HttpStatusCode.OK)
        {
```

```
            Snackbar.Add($"{@globals.Localizations["Messages_FailedToGetData"]} - {result}",
Severity.Error);
            return;
        }
        Entries = JsonSerializer.Deserialize<List<LocalizationManagerDto>>(result, new
JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });
        Languages = Entries.First().Translations.Keys.ToArray();
        StateHasChanged();
    }

    private void AddRow()
    {
        var entry = new LocalizationManagerDto
            {
                Placeholder = "",
                Translations = Languages.ToDictionary(l => l, l => "")
            };
        Entries.Add(entry);
    }

    private void RemoveRow(LocalizationManagerDto managerDto)
    {
        Entries.Remove(managerDto);
    }

    private async Task SaveAll()
    {
        var serverUrl = Configuration.GetValue<string>("ServerUrl");
        using var cli = new HttpClient();
        cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
globals.Token);
        var response = await cli.PostAsJsonAsync($"{serverUrl}/admin/updateDisplayLocalizations",
Entries);
        if (response.IsSuccessStatusCode)
        {
            Snackbar.Add(@globals.Localizations["Messages_ChangesSaved"], Severity.Success);
        }
        else
        {
            var result = await response.Content.ReadAsStringAsync();
            Snackbar.Add($"{@globals.Localizations["Messages_FailedToSaveChanges"]} - {result}",
Severity.Error);
        }
    }
}
```

MyBookings:

```
@page "/my-bookings"
@using System.Net.Http.Headers
@using System.Text.Json
@using Client.Components
@using Shared.Model.Dtos
@inject IConfiguration Configuration
@inject IDialogService DialogService
@inject ISnackbar Snackbar

<PageTitle>My Bookings</PageTitle>

<MudContainer MaxWidth="MaxWidth.Large" Class="mt-4">
    <MudText Typo="Typo.h4" Class="mb-4">@globals.Localizations["MyBookings_Title"]</MudText>

    @if (_bookings == null)
    {
        <MudProgressCircular Color="Color.Primary" Indeterminate="true" />
    }
    else if (!_bookings.Any())
    {
        <MudAlert
Severity="Severity.Info">@globals.Localizations["MyBookings_NoBookings"]</MudAlert>
    }
    else
    {
```

```
        <MudTable Items="@_bookings" Dense="true" Hover="true" Bordered="true" Striped="true">
            <HeaderContent>
                <MudTh>@globals.Localizations["MyBookings_TourTitle"]</MudTh>
                <MudTh>@globals.Localizations["MyBookings_StartDate"]</MudTh>
                <MudTh>@globals.Localizations["MyBookings_EndDate"]</MudTh>
                <MudTh>@globals.Localizations["MyBookings_Price"]</MudTh>
                <MudTh>@globals.Localizations["MyBookings_Status"]</MudTh>
                <MudTh>@globals.Localizations["MyBookings_Actions"]</MudTh>
            </HeaderContent>
            <RowTemplate>
                <MudTd DataLabel="@globals.Localizations["MyBookings_TourTitle"]">
                    <NavLink href=@($@"/tour/{context.TourInstanceId}")>
                        @context.TourTitle
                    </NavLink>
                </MudTd>
                <MudTd
DataLabel="@globals.Localizations["MyBookings_StartDate"]">@context.StartDate.ToLocalTime().ToShort
DateString()</MudTd>
                <MudTd
DataLabel="@globals.Localizations["MyBookings_EndDate"]">@context.EndDate.ToLocalTime().ToShortDate
String()</MudTd>
                <MudTd
DataLabel="@globals.Localizations["MyBookings_Price"]">@context.TotalPrice.ToString("C")</MudTd>
                <MudTd DataLabel="@globals.Localizations["MyBookings_Status"]">
                    <MudChip T="string" Color="@GetStatusColor(context)" Size="Size.Small">
                        @GetStatusText(context)
                    </MudChip>
                </MudTd>
                <MudTd DataLabel="@globals.Localizations["MyBookings_Actions"]">
                    <MudStack Row=true Spacing="2">
                        @if (!context.IsCancelled && context.StartDate > DateTime.UtcNow)
                        {
                            <MudButton Variant="Variant.Text"
                                       Color="Color.Error"
                                       Size="Size.Small"
                                       OnClick="@(() => CancelBooking(context))"
                                       Disabled="@(context.StartDate <= DateTime.UtcNow.AddDays(1))">
                                @globals.Localizations["MyBookings_Cancel"]
                            </MudButton>
                        }
                        @if (context.EndDate < DateTime.UtcNow && !context.IsCancelled &&
!context.HasRated)
                        {
                            <MudButton Variant="Variant.Text"
                                       Color="Color.Primary"
                                       Size="Size.Small"
                                       OnClick="@(() => RateTour(context))">
                                @globals.Localizations["MyBookings_Rate"]
                            </MudButton>
                        }
                        @if (context.HasRated)
                        {
                            <MudIcon Icon="@Icons.Material.Filled.Star" Color="Color.Warning" />
                        }
                    </MudStack>
                </MudTd>
            </RowTemplate>
        </MudTable>
    }
</MudContainer>

@code {
    private List<BookingDto>? _bookings;

    protected override async Task OnInitializedAsync()
    {
        await LoadBookings();
    }

    private async Task LoadBookings()
    {
        try
        {
            var serverUrl = Configuration.GetValue<string>("ServerUrl");
            using var cli = new HttpClient();
            cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
globals.Token);
```

```
            var response = await cli.GetAsync($"{serverUrl}/Booking/GetMyBookings");
            var result = await response.Content.ReadAsStringAsync();
            if (response.StatusCode == System.Net.HttpStatusCode.OK)
            {
                _bookings = JsonSerializer.Deserialize<List<BookingDto>>(result, new
JsonSerializerOptions
                {
                    PropertyNameCaseInsensitive = true
                });
                StateHasChanged();
            }
            else
            {
                Snackbar.Add(globals.Localizations["Messages_FailedToLoad"], Severity.Error);
            }
        }
        catch (Exception ex)
        {
            Snackbar.Add(globals.Localizations["Messages_FailedToLoad"], Severity.Error);
        }
    }

    private async Task CancelBooking(BookingDto booking)
    {
        var parameters = new DialogParameters
            {
                ["ContentText"] = globals.Localizations["MyBookings_CancelConfirmation"],
                ["ConfirmButtonText"] = globals.Localizations["UI_Yes"],
                ["CancelButtonText"] = globals.Localizations["UI_No"]
            };
        var dialog = await
DialogService.ShowAsync<ConfirmDialog>(globals.Localizations["MyBookings_CancelBooking"],
parameters);
        var result = await dialog.Result;

        if (!result.Canceled)
        {
            try
            {
                var serverUrl = Configuration.GetValue<string>("ServerUrl");
                using var cli = new HttpClient();
                cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
globals.Token);
                var response = await
cli.GetAsync($"{serverUrl}/Booking/CancelBooking?id={booking.Id}");
                if (response.StatusCode == System.Net.HttpStatusCode.OK)
                {
                    await LoadBookings();
                    Snackbar.Add(globals.Localizations["Messages_BookingCancelled"],
Severity.Success);
                }
                else
                {
                    var error = await response.Content.ReadAsStringAsync();
                    Snackbar.Add($"{globals.Localizations["Messages_FailedToCancel"]} - {error}",
Severity.Error);
                }
            }
            catch (Exception ex)
            {
                Snackbar.Add(globals.Localizations["Messages_FailedToCancel"], Severity.Error);
            }
        }
    }

    private async Task RateTour(BookingDto booking)
    {
        var parameters = new DialogParameters
        {
            ["TourInstanceId"] = booking.TourInstanceId
        };

        var dialog = await
DialogService.ShowAsync<RateTourDialog>(globals.Localizations["MyBookings_RateTour"], parameters);
        var result = await dialog.Result;

        if (!result.Canceled)
```

```
        {
            await LoadBookings();
        }
    }

    private Color GetStatusColor(BookingDto context)
    {
        if (context.IsCancelled)
            return Color.Error;
        if (context.EndDate < DateTime.UtcNow)
            return Color.Primary;
        if (context.StartDate < DateTime.UtcNow)
            return Color.Warning;

        return Color.Success;
    }

    private string GetStatusText(BookingDto context)
    {
        if (context.IsCancelled)
            return globals.Localizations["MyBookings_StatusCancelled"];
        if (context.EndDate < DateTime.UtcNow)
            return globals.Localizations["MyBookings_StatusPassed"];
        if (context.StartDate < DateTime.UtcNow)
            return globals.Localizations["MyBookings_StatusRunning"];

        return globals.Localizations["MyBookings_StatusActive"];
    }
}
```

TourDesc:

```
@page "/tour/{InstanceId:int}"
@using System.Text.Json
@using System.Net.Http.Headers
@using Shared.Model.Dtos
@inject Globals globals
@inject NavigationManager Navigation
@inject IConfiguration Configuration
@inject ISnackbar Snackbar

<PageTitle>@globals.Localizations["Tour_Details_Title"]</PageTitle>

<MudContainer MaxWidth="MaxWidth.Large" Class="mt-6">
    @if (IsLoading)
    {
        <MudProgressCircular Indeterminate Color="Color.Primary" />
    }
    else if (tour == null)
    {
        <MudText Typo="Typo.h5">@globals.Localizations["Tour_NotFound"]</MudText>
    }
    else
    {
        <MudGrid GutterSize="3">
            <MudItem xs="12" md="6">
                <MudImage Height="400" Src="@(string.IsNullOrEmpty(tour.ImageUrl) ?
"https://via.placeholder.com/600x400?text=Tour" : tour.ImageUrl)" Alt="@tour.Title"
Style="width:100%; border-radius:12px;" />
            </MudItem>
            <MudItem xs="12" md="6">
                <MudText Typo="Typo.h4" Class="mb-2">@tour.Title</MudText>
                <MudStack Row=true AlignItems="AlignItems.Center" Spacing="1" Class="mb-2">
                    <MudText Typo="Typo.body2" Class="mb-
1"><b>@globals.Localizations["Tour_Guide"]:</b></MudText>
                    <MudAvatar Size="Size.Small">
                        <MudImage Src=@tour.GuideAvatarUrl></MudImage>
                    </MudAvatar>
                    <MudText Typo="Typo.subtitle2">@tour.GuideName @tour.GuideSurname</MudText>
                </MudStack>
                <MudText Typo="Typo.body1" Class="mb-2">@tour.Description</MudText>
                <MudText Typo="Typo.body2" Class="mb-
1"><b>@globals.Localizations["Tour_Locations"]:</b> @string.Join(", ", tour.Locations.Select(l =>
globals.LocaleLocalizations[l].City))</MudText>
                <MudText Typo="Typo.body2" Class="mb-
1"><b>@globals.Localizations["Tour_Dates"]:</b> @tour.StartDate.Value.ToString("dd MMM yyyy") -
@tour.EndDate.Value.ToString("dd MMM yyyy")</MudText>
```

```
                <MudText Typo="Typo.body2" Class="mb-
1"><b>@globals.Localizations["Tour_Duration"]:</b> @((tour.EndDate - tour.StartDate).Value.Days)
@globals.Localizations["Tours_Filter_Days"]</MudText>
                <MudText Typo="Typo.body2" Class="mb-
1"><b>@globals.Localizations["Tours_Filter_TourType"]:</b>
@globals.Localizations[$"TourType_{tour.TourType}"]</MudText>
                <MudText Typo="Typo.body2" Class="mb-
1"><b>@globals.Localizations["Tour_Price"]:</b> $@tour.Price</MudText>
                <MudStack Row=true AlignItems="AlignItems.Center" Spacing="1" Class="mb-2">
                    <MudText Typo="Typo.body2">@globals.Localizations["Tour_Rating"]:</MudText>
                    <MudRating SelectedValue=@((int)tour.Rating) MaxValue="5" ReadOnly
Size="Size.Small" />
                    <MudText Typo="Typo.caption">@tour.Rating.ToString("0.0")</MudText>
                </MudStack>
                <MudDivider Class="my-2" />
                <MudButton Variant="Variant.Filled" Color="Color.Primary"
Disabled="@(!globals.IsLoggedIn)" OnClick="BookTour">
                    @globals.Localizations["Tour_BookButton"]
                </MudButton>
                @if (!globals.IsLoggedIn)
                {
                    <MudText Typo="Typo.caption"
Color="Color.Error">@globals.Localizations["Tour_LoginToBook"]</MudText>
                }
            </MudItem>
        </MudGrid>
    }
</MudContainer>

@code {
    [Parameter]
    public int InstanceId { get; set; }

    private TourDescDto? tour = new TourDescDto();
    private bool IsLoading = true;
    private bool IsBooking = false;
    private int SelectedParticipants = 1;

    protected override async Task OnInitializedAsync()
    {
        await LoadData();
    }

    private async Task LoadData()
    {
        IsLoading = true;
        try
        {
            var serverUrl = Configuration.GetValue<string>("ServerUrl");
            using var cli = new HttpClient();
            cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
globals.Token);

            var response = await cli.GetAsync($"{serverUrl}/Tours/GetById/{InstanceId}");
            var result = await response.Content.ReadAsStringAsync();

            if (response.StatusCode != System.Net.HttpStatusCode.OK)
            {
                Snackbar.Add($"{globals.Localizations["Messages_FailedToGetData"]} - {result}",
Severity.Error);
                return;
            }

            tour = JsonSerializer.Deserialize<TourDescDto>(result, new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

            if (tour == null)
            {
                Snackbar.Add(globals.Localizations["Messages_TourNotFound"], Severity.Error);
                Navigation.NavigateTo("/tours");
                return;
            }

            SelectedParticipants = Math.Min(SelectedParticipants, tour.MaxParticipants -
tour.CurrentParticipants);
```

```
        }
        catch (Exception ex)
        {
            Snackbar.Add($"{globals.Localizations["Messages_FailedToGetData"]} - {ex.Message}",
Severity.Error);
        }
        finally
        {
            IsLoading = false;
        }
    }

    private async Task BookTour()
    {
        if (tour == null) return;

        IsBooking = true;
        try
        {
            var serverUrl = Configuration.GetValue<string>("ServerUrl");
            using var cli = new HttpClient();
            cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
globals.Token);

            var response = await cli.GetAsync($"{serverUrl}/Booking/BookTour?id={InstanceId}");
            var result = await response.Content.ReadAsStringAsync();

            if (response.StatusCode != System.Net.HttpStatusCode.OK)
            {
                Snackbar.Add($"{globals.Localizations["Messages_FailedToBookTour"]} - {result}",
Severity.Error);
                return;
            }

            Snackbar.Add(globals.Localizations["Messages_TourBookedSuccessfully"],
Severity.Success);
            await LoadData(); // Refresh data to update current participants
        }
        catch (Exception ex)
        {
            Snackbar.Add($"{globals.Localizations["Messages_FailedToBookTour"]} - {ex.Message}",
Severity.Error);
        }
        finally
        {
            IsBooking = false;
        }
    }
}
```

Tours:

```
@page "/tours"

@using System.Net.Http.Headers

@using System.Text.Json

@using Common.Model.Dtos

@using Shared.Model

@using Shared.Model.Dtos

@inject Globals globals

@inject NavigationManager Navigation

@inject IConfiguration Configuration

@inject ISnackbar Snackbar


<PageTitle>Tours</PageTitle>

<MudContainer MaxWidth="MaxWidth.False" Class="p-0">
```

```
<div style="position: relative; height: 800px;">
    <MudImage ObjectPosition="ObjectPosition.Top"
    ObjectFit="ObjectFit.Cover"
    Style="width:100%; height:100%"
    Src="https://res.cloudinary.com/da7kbbrap/image/upload/v1746448800/ChatGPT_Image_May_5_2025_03_39_39_PM_kbvhla.png"
    Class="rounded-lg" />

    <div class="d-flex flex-column align-items-center justify-center z-100 absolute"
    style="top: 0; left: 0; right: 0; bottom: 0; color: white">
        <MudText Typo="Typo.h3" Class="mb-4">@globals.Localizations["Tours_Hero_Title"]</MudText>

        <MudPaper Class="d-flex align-center gap-2 p-3 rounded" Style="position: relative; overflow: visible;">
            <!-- Destination Search -->
            <MudAutocomplete T="int"
                        Placeholder="@globals.Localizations["Tours_Filter_SearchDestinations"]"
                        SearchFunc="SearchDestinations"
                        Style="width: 200px;"
                        @bind-Value="SelectedDestination"
                        ToStringFunc="@(id => globals.LocaleLocalizations[id].City)" />

            <!-- Date Picker -->
            <MudDatePicker @bind-Date="Filters.StartDate" Placeholder="@globals.Localizations["Tours_Filter_StartDate"]" />

            <!-- Fake Dropdown Toggle -->
            <MudButton Variant="Variant.Outlined"
                    OnClick="@ToggleFilter"
                    EndIcon=@(ShowFilter ? Icons.Material.Filled.ExpandLess : Icons.Material.Filled.ExpandMore)
                    Disabled="@IsLoading">
                @if (ShowFilter)
                {
                    <div>@globals.Localizations["Tours_Filter_HideFilters"]</div>
                }
                else
                {
                    <div>@globals.Localizations["Tours_Filter_ShowFilters"]</div>
                }
            </MudButton>

            <!-- Dropdown Content -->
            @if (ShowFilter)
```

```
        {
            <MudPaper Class="p-4 mt-2 absolute z-100" Style="top: 100%; left: 0; width: 100%; box-shadow: var(--mud-elevation-8);
background: white;">
                @if (IsLoading)
                {
                    <MudProgressCircular Indeterminate Color="Color.Primary" />
                }
                else
                {
                    <MudStack Spacing="3" Class="p-4">
                        <MudStack Direction="Row" Justify="Justify.SpaceBetween" AlignItems="AlignItems.Center">
                            <MudLink Class="text-primary" OnClick="ClearAll">@globals.Localizations["Tours_Filter_ClearAll"]</MudLink>
                        </MudStack>

                        <!-- Duration -->
                        <MudStack>
                            <MudText Typo="Typo.subtitle2">@globals.Localizations["Tours_Filter_Duration"]</MudText>
                            <MudRangeSlider Min="@MinPossibleDuration"
                                            Max="@MaxPossibleDuration"
                                            Step="1"
                                            @bind-Value="Filters.FromDurationDays"
                                            @bind-UpperValue="Filters.ToDurationDays"
                                            Range="true"
                                            Immediate="true"
                                            MinDistance="1" />
                            <MudText Typo="Typo.caption">@Filters.FromDurationDays – @Filters.ToDurationDays
@globals.Localizations["Tours_Filter_Days"]</MudText>
                        </MudStack>

                        <!-- Price Range -->
                        <MudStack>
                            <MudText Typo="Typo.subtitle2">@globals.Localizations["Tours_Filter_Price"]</MudText>
                            <MudRangeSlider Min="@MinPossiblePrice"
                                            Max="@MaxPossiblePrice"
                                            Step="50"
                                            @bind-Value="Filters.FromPrice"
                                            @bind-UpperValue="Filters.ToPrice"
                                            Range="true"
                                            Immediate="true"
                                            MinDistance="50" />
                            <MudText Typo="Typo.caption">$@Filters.FromPrice – $@Filters.ToPrice</MudText>
```

```
                </MudStack>

                <!-- Number of Destinations -->
                <MudText Typo="Typo.subtitle2">@globals.Localizations["Tours_Filter_NumberOfDestinations"]</MudText>
                <MudRadioGroup T="DestinationCountEnum" @bind-SelectedOption="Filters.DestinationsCount">
                    <MudRadio T="DestinationCountEnum"
Option="DestinationCountEnum.Single">@globals.Localizations["Tours_Filter_SingleLocation"]</MudRadio>
                    <MudRadio T="DestinationCountEnum"
Option="DestinationCountEnum.TwoToThree">@globals.Localizations["Tours_Filter_2_3Locations"]</MudRadio>
                    <MudRadio T="DestinationCountEnum"
Option="DestinationCountEnum.FourPlus">@globals.Localizations["Tours_Filter_4PlusLocations"]</MudRadio>
                </MudRadioGroup>

                <!-- Tour Type -->
                <MudText Typo="Typo.subtitle2">@globals.Localizations["Tours_Filter_TourType"]</MudText>
                <MudSelect MultiSelectionTextFunc="MultiSelectionTextFunc"
                    T="TourTypeEnum"
                    MultiSelection="true"
                    @bind-SelectedValues="Filters.TourTypes">
                  @foreach (TourTypeEnum type in Enum.GetValues(typeof(TourTypeEnum)))
                  {
                    @if (type != TourTypeEnum.None)
                    {
                      <MudSelectItem Value="@type">@globals.Localizations[$"TourType_{type}"]</MudSelectItem>
                    }
                  }
                </MudSelect>

                <!-- Rating -->
                <MudStack>
                  <MudText Typo="Typo.subtitle2">@globals.Localizations["Tours_Filter_Rating"]</MudText>
                  <MudSlider Min="1.0"
                      Max="5.0"
                      Step="0.1"
                      @bind-Value="Filters.MinRating"/>
                  <MudText Typo="Typo.caption">@Filters.MinRating.ToString("0.0")
@globals.Localizations["Tours_Filter_Stars"]</MudText>
                </MudStack>

                <!-- Format Options -->
                <MudText Typo="Typo.subtitle2">@globals.Localizations["Tours_Filter_TourFormat"]</MudText>
```

```
                        <MudCheckBox T=bool @bind-Checked="Filters.WithGuide" Label="@globals.Localizations["Tours_Filter_WithGuide"]"
/>
                        <MudCheckBox T=bool @bind-Checked="Filters.PrivateTour"
Label="@globals.Localizations["Tours_Filter_PrivateTour"]" />
                        <MudCheckBox T=bool @bind-Checked="Filters.GroupTour" Label="@globals.Localizations["Tours_Filter_GroupTour"]"
/>

                        <!-- Special Offers -->
                        <MudText Typo="Typo.subtitle2">@globals.Localizations["Tours_Filter_SpecialOffers"]</MudText>
                        <MudCheckBox T=bool @bind-Checked="Filters.OnSale" Label="@globals.Localizations["Tours_Filter_OnSale"]" />
                        <MudCheckBox T=bool @bind-Checked="Filters.StartsSoon" Label="@globals.Localizations["Tours_Filter_StartsSoon"]"
/>
                        <MudCheckBox T=bool @bind-Checked="Filters.SpecialDiscount"
Label="@globals.Localizations["Tours_Filter_SpecialDiscount"]" />
                    </MudStack>
                }
            </MudPaper>
        }
    </MudPaper>
</div>
</div>

<!-- Featured Tours -->
<MudContainer MaxWidth="MaxWidth.Large" Class="mt-6">
    <MudText Typo="Typo.h5" Class="mb-4">@globals.Localizations["Tours_FeaturedTours"]</MudText>
    @if (IsLoadingRecommendations)
    {
        <MudProgressCircular Indeterminate Color="Color.Primary" />
    }
    else if (RecommendedTours.Any())
    {
        <MudGrid GutterSize="3">
            @foreach (var tour in RecommendedTours)
            {
                <MudItem xs="12" sm="6" md="4">
                    @TourInstanceCard(tour)
                </MudItem>
            }
        </MudGrid>
        @if (!string.IsNullOrEmpty(RecommendationReason))
        {
            <MudText Typo="Typo.body2" Class="mt-2 text-muted">@RecommendationReason</MudText>
```

```
            }
        }
        else
        {
            <MudGrid GutterSize="3">
                @foreach (var tour in AllTours.Take(3))
                {
                    <MudItem xs="12" sm="6" md="4">
                        @TourInstanceCard(tour)
                    </MudItem>
                }
            </MudGrid>
        }
    </MudContainer>
</MudContainer>

<MudContainer MaxWidth="MaxWidth.Large" Class="mt-4">
    <MudGrid>
        <!-- Tours Display -->
        <MudItem xs="12">
            <MudText Typo="Typo.h5" Class="mb-3">@globals.Localizations["Tours_SearchResults"]</MudText>
            @if (IsLoading)
            {
                <MudProgressCircular Indeterminate Color="Color.Primary" />
            }
            else if (!AllTours.Any())
            {
                <MudText Typo="Typo.subtitle1">@globals.Localizations["Tours_NoResultsFound"]</MudText>
            }
            else
            {
                <MudGrid GutterSize="3">
                    @foreach (var tour in AllTours)
                    {
                        <MudItem xs="12" sm="6" md="4">
                            @TourInstanceCard(tour)
                        </MudItem>
                    }
                </MudGrid>
                <MudTablePager PageSizeOptions="new int[] {6, 12, 24}"
```

```
                    PageSize="@PageSize"
                    PageIndex="@PageIndex"
                    RowsPerPageString=""
                    OnPageChanged="OnPageChanged"
                    RowsCount="@TotalCount" />
            }
        </MudItem>
    </MudGrid>
</MudContainer>

@code {
    // Filter state
    private bool ShowFilter;
    private int SelectedDestination;
    private TourFiltersDto Filters = new TourFiltersDto
    {
        TourTypes = new List<TourTypeEnum>(),
        FromDurationDays = 1,
        ToDurationDays = int.MaxValue,
        FromPrice = 0M,
        ToPrice = decimal.MaxValue,
        MinRating = 1.0d
    };
    private bool IsLoading = true;

    // Range values
    private int MinPossibleDuration;
    private int MaxPossibleDuration;
    private decimal MinPossiblePrice;
    private decimal MaxPossiblePrice;

    // Data
    private List<TourDto> AllTours = new();
    private int TotalCount;
    private int PageIndex = 0;
    private int PageSize = 6;
    private string SortBy = "price";

    private List<TourDto> RecommendedTours = new();
    private string RecommendationReason = string.Empty;
```

```
private bool IsLoadingRecommendations = true;

protected override async Task OnInitializedAsync()
{
    await Task.WhenAll(LoadData(), LoadRecommendations());
}

private async Task LoadData()
{
    IsLoading = true;
    try
    {
        var serverUrl = Configuration.GetValue<string>("ServerUrl");
        using var cli = new HttpClient();

        // Fetch filtered tours
        var response = await cli.PostAsJsonAsync($"{serverUrl}/Tours/GetAll?page={PageIndex + 1}&pageSize={PageSize}", Filters);
        var result = await response.Content.ReadAsStringAsync();

        if (response.StatusCode != System.Net.HttpStatusCode.OK)
        {
            Snackbar.Add($"{globals.Localizations["Messages_FailedToGetData"]} - {result}", Severity.Error);
            return;
        }

        var toursResponse = JsonSerializer.Deserialize<PagedResult<TourDto>>(result, new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });

        AllTours = toursResponse.Items;
        TotalCount = toursResponse.TotalCount;

        MinPossibleDuration = toursResponse.MinDuration;
        MaxPossibleDuration = toursResponse.MaxDuration;
        MinPossiblePrice = toursResponse.MinPrice;
        MaxPossiblePrice = toursResponse.MaxPrice;
        Filters.FromDurationDays = MinPossibleDuration;
        Filters.ToDurationDays = MaxPossibleDuration;
        Filters.FromPrice = MinPossiblePrice;
```

```
        Filters.ToPrice = MaxPossiblePrice;
        Filters.SortBy = SortBy;
        Filters.SelectedDestination = SelectedDestination;
    }
    catch (Exception ex)
    {
        Snackbar.Add($"{globals.Localizations["Messages_FailedToGetData"]} - {ex.Message}", Severity.Error);
    }
    finally
    {
        IsLoading = false;
    }
}

private async Task LoadRecommendations()
{
    if (string.IsNullOrEmpty(globals.Token))
    {
        IsLoadingRecommendations = false;
        return;
    }

    try
    {
        var serverUrl = Configuration.GetValue<string>("ServerUrl");
        using var cli = new HttpClient();
        cli.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", globals.Token);

        var response = await cli.GetAsync($"{serverUrl}/Tours/GetRecommendations");
        var result = await response.Content.ReadAsStringAsync();

        if (response.StatusCode == System.Net.HttpStatusCode.OK)
        {
            var recommendation = JsonSerializer.Deserialize<TourRecommendationResponseDto>(result, new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

            if (recommendation != null)
            {
```

```
            RecommendedTours = recommendation.RecommendedTours;
            RecommendationReason = recommendation.RecommendationReason;
        }
    }
  }
  catch (Exception)
  {
    RecommendedTours.Clear();
    RecommendationReason = string.Empty;
  }
  finally
  {
    IsLoadingRecommendations = false;
  }
}

string MultiSelectionTextFunc(List<string?>? arg)
  => string.Join(", ", (Filters.TourTypes ?? new List<TourTypeEnum>()).Select(x => globals.Localizations[$"TourType_{x}"]));

private void ToggleFilter() => ShowFilter = !ShowFilter;

private void ClearAll()
{
  SelectedDestination = 0;
  Filters = new TourFiltersDto
  {
    TourTypes = new List<TourTypeEnum>(),
    FromDurationDays = 1,
    ToDurationDays = int.MaxValue,
    FromPrice = 0M,
    ToPrice = decimal.MaxValue,
    MinRating = 1.0d
  };
  ApplyFilters();
}

private async Task<IEnumerable<int>> SearchDestinations(string value, CancellationToken stoppingToken)
{
  if (string.IsNullOrWhiteSpace(value))
    return globals.LocaleLocalizations.Keys;
```

```
    return globals.LocaleLocalizations
        .Where(x => x.Value.City.Contains(value, StringComparison.InvariantCultureIgnoreCase))
        .Select(x => x.Key);
}

private void ApplyFilters()
{
    PageIndex = 0;
    _ = LoadData();
}

private void OnPageChanged(int page)
{
    PageIndex = page;
    _ = LoadData();
}

private void OnSortChanged(string value)
{
    SortBy = value;
    ApplyFilters();
}

// Helper for displaying a tour instance card
private RenderFragment TourInstanceCard(TourDto tour) => __builder =>
{
    <MudCard>
        <MudCardMedia Image="@(string.IsNullOrEmpty(tour.ImageUrl) ? "https://via.placeholder.com/400x200?text=Tour" : tour.ImageUrl)"
Height="180" />
        <MudCardContent>
            <MudText Typo="Typo.subtitle1">@tour.Title</MudText>
            <MudText Typo="Typo.body2">@string.Join(", ", tour.Locations.Select(l => globals.LocaleLocalizations[l].City))</MudText>
            <MudText Typo="Typo.caption">@tour.DurationDays @globals.Localizations["Tours_Filter_Days"]</MudText>
            <MudText Typo="Typo.caption">$@tour.Price</MudText>
            <MudDivider Class="my-2" />
            <MudStack Row=true AlignItems="AlignItems.Center" Spacing="2">
                <MudAvatar Size="Size.Small">
                    <MudImage Src=@tour.GuideAvatarUrl></MudImage>
                </MudAvatar>
                <MudText Typo="Typo.caption">@tour.GuideName @tour.GuideSurname</MudText>
            </MudStack>
```

```
            </MudCardContent>
            <MudCardActions>
                <MudIconButton Icon="@Icons.Material.Filled.FavoriteBorder" Disabled />
                <MudButton Variant="Variant.Text" Color="Color.Primary" OnClick="() =>
NavigateToTour(tour.Id)">@globals.Localizations["UI_Details"]</MudButton>
            </MudCardActions>
        </MudCard>;
    };

    private Color GetStatusColor(TourInstanceStatus status) => status switch
    {
        TourInstanceStatus.Scheduled => Color.Info,
        TourInstanceStatus.Completed => Color.Secondary,
        TourInstanceStatus.Cancelled => Color.Error,
        _ => Color.Default
    };

    private void NavigateToTour(int tourId)
    {
        Navigation.NavigateTo($"/tour/{tourId}");
    }
}
```

Program:

```
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;
using Client;
using Client.Dtos;
using MudBlazor.Services;
using Microsoft.JSInterop;
using MudExtensions.Services;

var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new
Uri(builder.HostEnvironment.BaseAddress) });

var globals = new Globals(builder.Configuration);
builder.Services.AddSingleton<Globals>(x => globals);

builder.Services.AddMudServices();
builder.Services.AddMudExtensions();

var host = builder.Build();
var jsRuntime = host.Services.GetRequiredService<IJSRuntime>();
await globals.LoadUser(jsRuntime);

await host.RunAsync();
```

Common:

Account:

```
using Shared.Model;
using System.ComponentModel.DataAnnotations;
```

```
using System.ComponentModel.DataAnnotations.Schema;

namespace Common.Model.Entities;

public class Account
{
    [Key]
    public int Id { get; set; }
    public bool IsDeleted { get; set; }
    public DateTime? UtcDeleted { get; set; }
    public string AccountId { get; set; }
    public string Login { get; set; }
    public string Username { get; set; }
    public string PasswordHash { get; set; }
    public DateTime UtcCreated { get; set; }
    public AccountRolesEnum Roles { get; set; }
    public LocalizationCode Locale { get; set; }
    public string? AvatarUrl { get; set; }


    public Account() { }
    public Account(string passwordHash, string login)
    {
        IsDeleted = false;
        PasswordHash = passwordHash;
        Login = login;
        Username = login;
        AccountId = Guid.NewGuid().ToString();
        Locale = LocalizationCode.ENG;
        UtcCreated = DateTime.UtcNow;
        Roles = AccountRolesEnum.Client;
    }
}
```

AppEvent:

```
using System.ComponentModel.DataAnnotations;
using Shared.Model;
using Shared.Model.Dtos;

namespace Common.Model.Entities;

public class AppEvent
{
    [Key]
    public int Id { get; set; }
    public DateTime UtcCreated { get; set; }
    public EventTypeEnum EventType { get; set; }
    public string EventJsonData { get; set; }

    public AppEvent() { }

    public AppEvent(EventDto dto, string eventJsonData)
    {
        UtcCreated = dto.UtcCreated;
        EventType = dto.EventType;
        EventJsonData = eventJsonData;
    }
}
```

Guide:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Common.Model.Entities;

public class Guide
{
    [Key]
    public int Id { get; set; }

    public string Name { get; set; }
    public string Surname { get; set; }
    public byte Age { get; set; }
    public DateTime CareerStartUtc { get; set; }
    public bool IsActive { get; set; }
```

```
    public int AccountId { get; set; }
    [ForeignKey("AccountId")]
    public Account Account { get; set; }
}
```

Language:

```
using Shared.Model;
using System.ComponentModel.DataAnnotations;

namespace Common.Model.Entities;

public class Language
{
    [Key]
    public LocalizationCode Locale { get; set; }
    public string CitiesAndCountriesJson { get; set; }
    public string DisplayLocalizationsJson { get; set; }

    public Language(LocalizationCode locale, string citiesAndCountriesJson, string
displayLocalizationsJson)
    {
        Locale = locale;
        CitiesAndCountriesJson = citiesAndCountriesJson;
        DisplayLocalizationsJson = displayLocalizationsJson;
    }
}
```

Location:

```
using System.ComponentModel.DataAnnotations;

namespace Common.Model.Entities;

public class Location
{
    [Key]
    public int GeoId { get; set; }
    public string Name { get; set; }
    public string Country { get; set; }
}
```

Tour:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Shared.Model;

namespace Common.Model.Entities;

public class Tour
{
    [Key]
    public int Id { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public string ImageUrl { get; set; }
    public List<int> Locations { get; set; } = new();
    public decimal Price { get; set; }
    public TourTypeEnum TourType { get; set; }
    public bool WithGuide { get; set; }
    public SpecialOfferEnum SpecialOffers { get; set; }
    public int DurationDays { get; set; }
    public TourClassificationEnum Classification { get; set; }

    public int GuideId { get; set; }
    [ForeignKey("GuideId")]
    public Guide Guide { get; set; }

    // Template status
    public bool IsActive { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime? LastModifiedAt { get; set; }

    public List<TourInstance> Instances { get; set; }
}
```

TourBookings:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Common.Model.Entities;

public class TourBooking
{
    [Key]
    public int Id { get; set; }

    [Required]
    public int AccountId { get; set; }
    [ForeignKey(nameof(AccountId))]
    public Account Account { get; set; } = null!;

    [Required]
    public int TourInstanceId { get; set; }
    [ForeignKey(nameof(TourInstanceId))]
    public TourInstance TourInstance { get; set; } = null!;

    [Required]
    public DateTime BookedUtc { get; set; }

    [Required]
    [Column(TypeName = "decimal(18,2)")]
    public decimal TotalPrice { get; set; }

    public bool IsCancelled => CancellationDate is not null;
    public string? CancellationReason { get; set; }
    public DateTime? CancellationDate { get; set; }
}
```

TourInstance:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Shared.Model;

namespace Common.Model.Entities;

public class TourInstance
{
    [Key]
    public int Id { get; set; }

    public int TourId { get; set; }
    [ForeignKey("TourId")]
    public Tour Tour { get; set; }

    // Instance-specific dates
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }

    // Instance status
    public TourInstanceStatus GetStatus()
    {
        if (IsCancelled)
            return TourInstanceStatus.Cancelled;
        if (EndDate <= DateTime.UtcNow)
            return TourInstanceStatus.Completed;
        return TourInstanceStatus.Scheduled;
    }
    public static TourInstanceStatus GetStatus(bool isCancelled, DateTime endDate)
    {
        if (isCancelled)
            return TourInstanceStatus.Cancelled;
        if (endDate <= DateTime.UtcNow)
            return TourInstanceStatus.Completed;
        return TourInstanceStatus.Scheduled;
    }

    public bool IsCancelled { get; set; }
    public int MaxParticipants { get; set; }

    // Navigation property for rates
```

```
    public List<TourInstanceRate> Rates { get; set; } = new();
    public List<TourBooking> Bookings { get; set; } = new();

    // Computed property for average rating
    public double? Rating => Rates.Any() ? Rates.Average(r => r.Rate / 10.0) : null;
}
```

TourInstanceRate:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Common.Model.Entities;

public class TourInstanceRate
{
    [Key]
    public int Id { get; set; }
    public byte Rate { get; set; } // from 10 to 50, to be represented as scale from 1.0 to 5.0
    public string TouristCommentary { get; set; }
    public DateTime RatedTimeUtc { get; set; }

    public int TouristAccountId { get; set; }
    [ForeignKey("TouristAccountId")]
    public Account TouristAccount { get; set; }

    public int TourInstanceId { get; set; }
    [ForeignKey("TourInstanceId")]
    public TourInstance TourInstance { get; set; }
}
```

Constants:

```
namespace Shared.Model;

public class Constants
{
    public static readonly string LOCATION_LOCALIZATION_CACHE_KEY = "LocationLocalizations";
    public static readonly string DISPLAY_LOCALIZATION_CACHE_KEY = "DisplayLocalizations";
    public const int MaxPageSize = 100;
    public const int STARTS_SOON_DAYS = 14;


    // Default filter values
    public const decimal DefaultMinPrice = 0;
    public const decimal DefaultMaxPrice = 5000;
}
```

Enums:

```
namespace Shared.Model;

[Flags]
public enum AccountRolesEnum
{
    None = 0,
    Client = 1 << 0,
    Modify = 1 << 1,
    Guide = 1 << 2
}

public enum GeoFeatureCode
{
    None = 0,
    PPL = 1,
    PPLC = 2,
    PPLL = 3,
}

public enum LocalizationCode
{
    None = 0,
    UKR = 1,
    ENG = 2,
    RUS = 3
}
```

```
public enum EventTypeEnum
{
    None = 0,
    LocationsLocalizationsUpdated = 1,
    DisplayLocalizationsUpdated = 2,
}

public enum TourTypeEnum
{
    None = 0,
    Recreational = 1,
    Hiking = 2,
    Mixed = 3,
    Recovery = 4,
    Sightseeing = 5,
}

[Flags]
public enum SpecialOfferEnum
{
    None = 0,
    OnSale = 1 << 0,
    StartsSoon = 1 << 1,
    SpecialDiscount = 1 << 2
}

public enum DestinationCountEnum
{
    None = 0,
    Single = 1,
    TwoToThree = 2,
    FourOrMore = 3
}

public enum TourInstanceStatus
{
    None = 0,
    Scheduled = 1,
    Completed = 2,
    Cancelled = 3
}
public enum TourClassificationEnum
{
    Private,
    Group
}

public static class EnumSwitches
{
    public static LocalizationCode GetLocalizationCode(string code) => code switch
    {
        "uk" => LocalizationCode.UKR,
        "en" => LocalizationCode.ENG,
        "ru" => LocalizationCode.RUS,
        _ => LocalizationCode.None
    };

    public static GeoFeatureCode GetGeoFeatureCode(string code) => code switch
    {
        "PPLC" => GeoFeatureCode.PPLC,
        "PPLL" => GeoFeatureCode.PPLL,
        "PPL" => GeoFeatureCode.PPL,
        _ => GeoFeatureCode.None
    };
}
```

ErrorCodes:

```
namespace Shared.Model;

public class ErrorCodes
{
    // General
    public const string ErrorCode_PageMustBeBiggerThanZero = "ErrorCode_PageMustBeBiggerThanZero";
    public const string ErrorCode_PageSizeNotAllowed = "ErrorCode_PageSizeNotAllowed";
```

```
    // Account
    public const string ErrorCode_Unauthorized = "ErrorCode_Unauthorized";
    public const string ErrorCode_AccountNotFound = "ErrorCode_AccountNotFound";
    public const string ErrorCode_TriedToRegisterExistingAccount =
"ErrorCode_TriedToRegisterExistingAccount";
    public const string ErrorCode_AccountWrongPassword = "ErrorCode_AccountWrongPassword";

    // Tours
    public const string ErrorCode_TourNotFound = "ErrorCode_TourNotFound";
    public const string ErrorCode_TourIsNotAvailableForBooking =
"ErrorCode_TourIsNotAvailableForBooking";
    public const string ErrorCode_TourIsFullyBooked = "ErrorCode_TourIsFullyBooked";
    public const string ErrorCode_TourIsBookedByAccount = "ErrorCode_TourIsBookedByAccount";
    public const string ErrorCode_CannotRateRatedTour = "ErrorCode_CannotRateRatedTour";
    public const string ErrorCode_CannotRateNotCompletedTour =
"ErrorCode_CannotRateNotCompletedTour";
    public const string ErrorCode_CannotRateNotBookedTour = "ErrorCode_CannotRateNotBookedTour";

    // Bookings
    public const string ErrorCode_BookingNotFound = "ErrorCode_BookingNotFound";
    public const string ErrorCode_BookingIsCancelled = "ErrorCode_BookingIsCancelled";
    public const string ErrorCode_TooLateToCancelBooking = "ErrorCode_TooLateToCancelBooking";

    // Localization
    public const string ErrorCode_LocaleNotFound = "ErrorCode_LocaleNotFound";
    public const string ErrorCode_LocaleNotSupported = "ErrorCode_LocaleNotSupported";

    // AI
    public const string ErrorCode_FailedToGetOpenApiResponse =
"ErrorCode_FailedToGetOpenApiResponse";
    public const string ErrorCode_InvalidOpenApiResponse = "ErrorCode_InvalidOpenApiResponse";
    public const string ErrorCode_FailedToParseOpenApiResponse =
"ErrorCode_FailedToParseOpenApiResponse";
}
```

OperationResult:

```
namespace Shared.Model;

public class OperationResult<T>
{
    private T Data { get; set; }
    public string? ErrorMessageCode { get; set; } = null!;
    public bool IsSuccess => string.IsNullOrEmpty(ErrorMessageCode);

    public OperationResult(string errorMessageCode)
    {
        ErrorMessageCode = errorMessageCode;
    }

    public OperationResult(T obj)
    {
        Data = obj;
    }

    public bool TryGetValue(out T obj)
    {
        obj = default!;
        if (IsSuccess)
        {
            obj = Data;
            return true;
        }

        return false;
    }
}

public class OperationResult
{
    public string ErrorMessageCode { get; set; } = null!;
    public bool IsSuccess => string.IsNullOrEmpty(ErrorMessageCode);
    public OperationResult(string errorMessageCode)
    {
        ErrorMessageCode = errorMessageCode;
    }
    private OperationResult()
```

```
    {

    }
    public static OperationResult Success()
    {
        return new OperationResult();
    }
}
```

ServiceDefaults:

SqlContext:

```
using Common.Model.Entities;
using Microsoft.EntityFrameworkCore;
using Shared.Model;
using Shared.Model.Dtos;
using System.Text.Json;

namespace Microsoft.Extensions.Hosting.DbContexts;

public class SqlContext : DbContext
{
    public SqlContext(DbContextOptions<SqlContext> options)
        : base(options)
    {
        Database.EnsureCreated();
    }

    public DbSet<Account> Accounts { get; set; }
    public DbSet<Language> Languages { get; set; }
    public DbSet<AppEvent> Events { get; set; }
    public DbSet<Location> Locations { get; set; }
    public DbSet<Tour> Tours { get; set; }
    public DbSet<TourInstance> TourInstances { get; set; }
    public DbSet<TourInstanceRate> TourInstanceRates { get; set; }
    public DbSet<Guide> Guides { get; set; }
    public DbSet<TourBooking> TourBookings { get; set; }


    protected override void OnModelCreating(ModelBuilder mb)
    {
        mb.Entity<Account>().Property(x => x.Login).HasMaxLength(100);
        mb.Entity<Account>().Property(x => x.AccountId).HasMaxLength(50);
        mb.Entity<Account>().HasIndex(x => x.Login).IsUnique();
        mb.Entity<Account>().HasIndex(x => x.AccountId).IsUnique();
        mb.Entity<Account>().Property(x => x.Username).HasMaxLength(100);
        mb.Entity<Account>().HasQueryFilter(x => !x.IsDeleted);

        mb.Entity<Language>().Property(x =>
x.DisplayLocalizationsJson).HasColumnName("DisplayLocalizationsJson");

        base.OnModelCreating(mb);
    }
}
```

Tests:

GuideServiceTests:

```
using Backend.Services;
using Microsoft.Extensions.Logging;
using NSubstitute;
using Shared.Model;
using Xunit;

namespace Tests.UnitTests;

public class GuideServiceTests : IClassFixture<TestFixture>
{
    private readonly TestFixture _fixture;
    private readonly GuideService _service;
    private readonly ILoggerFactory _loggerFactory;

    public GuideServiceTests(TestFixture fixture)
    {
```

```
        _fixture = fixture;
        _loggerFactory = Substitute.For<ILoggerFactory>();
        var logger = Substitute.For<ILogger<GuideService>>();
        _loggerFactory.CreateLogger(Arg.Any<string>()).Returns(logger);

        _service = new GuideService(_loggerFactory, _fixture.DbContext);
    }

    [Fact]
    public async Task GetGuideTours_WithValidGuideId_ReturnsGuideTours()
    {
        // Arrange
        var guideAccountId = "test_guide1"; // John Doe's account

        // Act
        var result = await _service.GetGuideTours(guideAccountId, CancellationToken.None);

        // Assert
        Assert.True(result.TryGetValue(out var data));
        Assert.NotNull(data);
        Assert.Equal(2, data.Count); // Both instances of Historical City Tour
        Assert.All(data, tour =>
        {
            Assert.Equal("Historical City Tour", tour.Title);
            Assert.Equal("John", tour.GuideName);
            Assert.Equal("Doe", tour.GuideSurname);
            Assert.Equal("https://test.com/avatar1.jpg", tour.GuideAvatarUrl);
        });
    }

    [Fact]
    public async Task GetGuideTours_WithInvalidGuideId_ReturnsEmptyList()
    {
        // Arrange
        var invalidGuideId = "non_existent_guide";

        // Act
        var result = await _service.GetGuideTours(invalidGuideId, CancellationToken.None);

        // Assert
        Assert.True(result.TryGetValue(out var data));
        Assert.NotNull(data);
        Assert.Empty(data);
    }

    [Fact]
    public async Task GetGuideTourInstances_WithValidGuideId_ReturnsGuideTourInstances()
    {
        // Arrange
        var guideAccountId = "test_guide1"; // John Doe's account

        // Act
        var result = await _service.GetGuideTourInstances(guideAccountId, CancellationToken.None);

        // Assert
        Assert.True(result.TryGetValue(out var data));
        Assert.NotNull(data);
        Assert.Equal(2, data.Count); // Both instances of Historical City Tour
        Assert.All(data, tour =>
        {
            Assert.Equal("Historical City Tour", tour.Title);
            Assert.Equal("John", tour.GuideName);
            Assert.Equal("Doe", tour.GuideSurname);
            Assert.Equal("https://test.com/avatar1.jpg", tour.GuideAvatarUrl);
            Assert.False(tour.IsCancelled);
        });

        // Verify specific instance details
        var firstInstance = data.First();
        Assert.Equal(1, firstInstance.Id);
        Assert.Equal(5, firstInstance.Rating);
        Assert.Equal(20, firstInstance.MaxParticipants);
        Assert.Equal(1, firstInstance.CurrentParticipants); // One booking exists
        Assert.True(firstInstance.StartDate > DateTime.UtcNow);
        Assert.True(firstInstance.EndDate > firstInstance.StartDate);
    }
```

```
    [Fact]
    public async Task GetGuideTourInstances_WithDifferentGuideId_ReturnsDifferentGuideTours()
    {
        // Arrange
        var guideAccountId = "test_guide2"; // Jane Smith's account

        // Act
        var result = await _service.GetGuideTourInstances(guideAccountId, CancellationToken.None);

        // Assert
        Assert.True(result.TryGetValue(out var data));
        Assert.NotNull(data);
        Assert.Single(data); // Only the Nature Adventure tour
        var tour = data.First();
        Assert.Equal("Nature Adventure", tour.Title);
        Assert.Equal("Jane", tour.GuideName);
        Assert.Equal("Smith", tour.GuideSurname);
        Assert.Equal("https://test.com/avatar2.jpg", tour.GuideAvatarUrl);
        Assert.Equal(150.00m, tour.Price);
        Assert.Equal(10, tour.MaxParticipants);
        Assert.Equal(0, tour.CurrentParticipants); // No bookings for this tour
    }

    [Fact]
    public async Task GetGuideTourInstances_WithInvalidGuideId_ReturnsEmptyList()
    {
        // Arrange
        var invalidGuideId = "non_existent_guide";

        // Act
        var result = await _service.GetGuideTourInstances(invalidGuideId, CancellationToken.None);

        // Assert
        Assert.True(result.TryGetValue(out var data));
        Assert.NotNull(data);
        Assert.Empty(data);
    }
}
```

ToursServiceTests:

```
using Backend.Services;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;
using NSubstitute;
using Shared.Model;
using Shared.Model.Dtos;
using System.Text.Json;
using Xunit;

namespace Tests.UnitTests;

public class ToursServiceTests : IClassFixture<TestFixture>
{
    private readonly TestFixture _fixture;
    private readonly ToursService _service;
    private readonly ILoggerFactory _loggerFactory;
    private readonly IConfiguration _configuration;

    public ToursServiceTests(TestFixture fixture)
    {
        _fixture = fixture;
        _loggerFactory = Substitute.For<ILoggerFactory>();
        var logger = Substitute.For<ILogger<ToursService>>();
        _loggerFactory.CreateLogger(Arg.Any<string>()).Returns(logger);

        _configuration = Substitute.For<IConfiguration>();
        _configuration.GetValue<string>("ServerUrl").Returns("http://test-server");

        _service = new ToursService(_loggerFactory, _fixture.DbContext, _configuration);
    }

    [Fact]
    public async Task GetAllTours_WithNoFilters_ReturnsAllActiveTours()
    {
        // Arrange
        var filters = new TourFiltersDto
```

```
        {
            ToPrice = int.MaxValue,
            ToDurationDays = int.MaxValue
        };
        var page = 1;
        var pageSize = 10;

        // Act
        var result = await _service.GetAllTours(page, pageSize, filters, CancellationToken.None);

        // Assert
        Assert.True(result.TryGetValue(out var data));
        Assert.NotNull(data);
        Assert.Equal(3, data.TotalCount); // We have 3 active tour instances
        Assert.Equal(3, data.Items.Count);
        Assert.Equal(1, data.TotalPages);
        Assert.Equal(100.00m, data.MinPrice);
        Assert.Equal(150.00m, data.MaxPrice);
        Assert.Equal(3, data.MinDuration);
        Assert.Equal(5, data.MaxDuration);
    }

    [Fact]
    public async Task GetAllTours_WithPriceFilter_ReturnsFilteredTours()
    {
        // Arrange
        var filters = new TourFiltersDto
        {
            FromPrice = 120.00m,
            ToPrice = 200.00m,
            ToDurationDays = int.MaxValue
        };
        var page = 1;
        var pageSize = 10;

        // Act
        var result = await _service.GetAllTours(page, pageSize, filters, CancellationToken.None);

        // Assert
        Assert.True(result.TryGetValue(out var data));
        Assert.NotNull(data);
        Assert.Equal(1, data.TotalCount); // Only the Nature Adventure tour
        Assert.Single(data.Items);
        Assert.Equal("Nature Adventure", data.Items[0].Title);
    }

    [Fact]
    public async Task GetAllTours_WithTourTypeFilter_ReturnsFilteredTours()
    {
        // Arrange
        var filters = new TourFiltersDto
        {
            TourTypes = new List<TourTypeEnum> { TourTypeEnum.Hiking },
            ToPrice = int.MaxValue,
            ToDurationDays = int.MaxValue
        };
        var page = 1;
        var pageSize = 10;

        // Act
        var result = await _service.GetAllTours(page, pageSize, filters, CancellationToken.None);
        var success = result.TryGetValue(out var data);

        // Assert
        Assert.True(success);
        Assert.NotNull(data);
        Assert.Equal(2, data.TotalCount); // Both instances of Historical City Tour
        Assert.Equal(2, data.Items.Count);
        Assert.All(data.Items, tour => Assert.Equal(TourTypeEnum.Hiking, tour.TourType));
    }

    [Fact]
    public async Task GetTourById_WithValidId_ReturnsTourDetails()
    {
        // Act
        var result = await _service.GetTourById(1, CancellationToken.None);
```

```
        // Assert
        Assert.True(result.TryGetValue(out var data));
        Assert.NotNull(data);
        Assert.Equal(1, data.Id);
        Assert.Equal("Historical City Tour", data.Title);
        Assert.Equal(100.00m, data.Price);
        Assert.Equal(5, data.Rating);
        Assert.Equal(20, data.MaxParticipants);
        Assert.Equal(1, data.CurrentParticipants); // One booking exists
        Assert.False(data.IsCancelled);
    }

    [Fact]
    public async Task GetTourById_WithInvalidId_ReturnsError()
    {
        // Act
        var result = await _service.GetTourById(999, CancellationToken.None);

        // Assert
        Assert.False(result.TryGetValue(out var data));
        Assert.Null(data);
    }

    [Fact]
    public async Task GetAllTours_WithSpecialOffersFilter_ReturnsFilteredTours()
    {
        // Arrange
        var filters = new TourFiltersDto
        {
            OnSale = true,
            ToPrice = int.MaxValue,
            ToDurationDays = int.MaxValue
        };
        var page = 1;
        var pageSize = 10;

        // Act
        var result = await _service.GetAllTours(page, pageSize, filters, CancellationToken.None);

        // Assert
        Assert.True(result.TryGetValue(out var data));
        Assert.NotNull(data);
        Assert.Equal(2, data.TotalCount); // Both instances of Historical City Tour
        Assert.Equal(2, data.Items.Count);
        Assert.All(data.Items, tour => Assert.True((tour.SpecialOffers & SpecialOfferEnum.OnSale) >
0));
    }

    [Fact]
    public async Task GetAllTours_WithClassificationFilter_ReturnsFilteredTours()
    {
        // Arrange
        var filters = new TourFiltersDto
        {
            PrivateTour = true,
            ToPrice = int.MaxValue,
            ToDurationDays = int.MaxValue
        };
        var page = 1;
        var pageSize = 10;

        // Act
        var result = await _service.GetAllTours(page, pageSize, filters, CancellationToken.None);

        // Assert
        Assert.True(result.TryGetValue(out var data));
        Assert.NotNull(data);
        Assert.Equal(1, data.TotalCount); // Only the Nature Adventure tour
        Assert.Single(data.Items);
        Assert.Equal(TourClassificationEnum.Private, data.Items[0].Classification);
    }
}
```

TestFixture:

using Common.Model.Entities;

```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Hosting.DbContexts;
using Shared.Model;
using Shared.Model.Dtos;
using System.Text.Json;

namespace Tests;

public class TestFixture : IDisposable
{
    public SqlContext DbContext { get; private set; }
    public TestFixture()
    {
        var options = new DbContextOptionsBuilder<SqlContext>()
            .UseInMemoryDatabase(databaseName: "TestDatabase")
            .Options;

        DbContext = new SqlContext(options);

        // Add test data
        var accounts = new List<Account>
        {
            new Account
            {
                Id = 1,
                AccountId = "test_guide1",
                Login = "guide1@test.com",
                Username = "guide1@test.com",
                PasswordHash = "dummy_hash",
                UtcCreated = DateTime.UtcNow.AddDays(-100),
                Roles = AccountRolesEnum.Guide,
                Locale = LocalizationCode.ENG,
                AvatarUrl = "https://test.com/avatar1.jpg",
                IsDeleted = false
            },
            new Account
            {
                Id = 2,
                AccountId = "test_guide2",
                Login = "guide2@test.com",
```

```
        Username = "guide2@test.com",
        PasswordHash = "dummy_hash",
        UtcCreated = DateTime.UtcNow.AddDays(-90),
        Roles = AccountRolesEnum.Guide,
        Locale = LocalizationCode.ENG,
        AvatarUrl = "https://test.com/avatar2.jpg",
        IsDeleted = false
    },
    new Account
    {
        Id = 3,
        AccountId = "test_user1",
        Login = "user1@test.com",
        Username = "user1@test.com",
        PasswordHash = "dummy_hash",
        UtcCreated = DateTime.UtcNow.AddDays(-50),
        Roles = AccountRolesEnum.Client,
        Locale = LocalizationCode.ENG,
        AvatarUrl = "https://test.com/avatar3.jpg",
        IsDeleted = false
    }
};

var guides = new List<Guide>
{
    new Guide
    {
        Id = 1,
        AccountId = accounts[0].Id,
        Account = accounts[0],
        Name = "John",
        Surname = "Doe",
        Age = 35,
        CareerStartUtc = DateTime.UtcNow.AddYears(-5),
        IsActive = true
    },
    new Guide
    {
        Id = 2,
        AccountId = accounts[1].Id,
```

```
        Account = accounts[1],
        Name = "Jane",
        Surname = "Smith",
        Age = 28,
        CareerStartUtc = DateTime.UtcNow.AddYears(-3),
        IsActive = true
    }
};

var tours = new List<Tour>
{
    new Tour
    {
        Id = 1,
        GuideId = guides[0].Id,
        Guide = guides[0],
        Title = "Historical City Tour",
        Description = "Explore the rich history of our city",
        ImageUrl = "https://test.com/tour1.jpg",
        Locations = new List<int> { 1, 2 },
        Price = 100.00m,
        TourType = TourTypeEnum.Hiking,
        WithGuide = true,
        SpecialOffers = SpecialOfferEnum.OnSale,
        DurationDays = 3,
        Classification = TourClassificationEnum.Group,
        IsActive = true,
        CreatedAt = DateTime.UtcNow.AddDays(-30),
        LastModifiedAt = DateTime.UtcNow.AddDays(-15)
    },
    new Tour
    {
        Id = 2,
        GuideId = guides[1].Id,
        Guide = guides[1],
        Title = "Nature Adventure",
        Description = "Experience the beauty of nature",
        ImageUrl = "https://test.com/tour2.jpg",
        Locations = new List<int> { 3, 4, 5 },
        Price = 150.00m,
```

```
        TourType = TourTypeEnum.Recovery,
        WithGuide = true,
        SpecialOffers = SpecialOfferEnum.SpecialDiscount,
        DurationDays = 5,
        Classification = TourClassificationEnum.Private,
        IsActive = true,
        CreatedAt = DateTime.UtcNow.AddDays(-25),
        LastModifiedAt = DateTime.UtcNow.AddDays(-10)
    }
};

var tourInstances = new List<TourInstance>
{
    new TourInstance
    {
        Id = 1,
        TourId = tours[0].Id,
        Tour = tours[0],
        StartDate = DateTime.UtcNow.AddDays(7),
        EndDate = DateTime.UtcNow.AddDays(10),
        MaxParticipants = 20,
        IsCancelled = false
    },
    new TourInstance
    {
        Id = 2,
        TourId = tours[1].Id,
        Tour = tours[1],
        StartDate = DateTime.UtcNow.AddDays(14),
        EndDate = DateTime.UtcNow.AddDays(19),
        MaxParticipants = 10,
        IsCancelled = false
    },
    new TourInstance
    {
        Id = 3,
        TourId = tours[0].Id,
        Tour = tours[0],
        StartDate = DateTime.UtcNow.AddDays(30),
        EndDate = DateTime.UtcNow.AddDays(33),
```

```
        MaxParticipants = 15,
        IsCancelled = false
    }
};

// Add rates for some instances
var rates = new List<TourInstanceRate>
{
    new TourInstanceRate
    {
        Id = 1,
        TourInstanceId = tourInstances[0].Id,
        TourInstance = tourInstances[0],
        TouristAccountId = accounts[2].Id,
        TouristAccount = accounts[2],
        Rate = 50, // 5.0 rating
        TouristCommentary = "Excellent tour!",
        RatedTimeUtc = DateTime.UtcNow.AddDays(-1)
    }
};

// Add bookings for some instances
var bookings = new List<TourBooking>
{
    new TourBooking
    {
        Id = 1,
        TourInstanceId = tourInstances[0].Id,
        TourInstance = tourInstances[0],
        AccountId = accounts[2].Id,
        Account = accounts[2],
        BookedUtc = DateTime.UtcNow.AddDays(-5),
        TotalPrice = 100.00m
    }
};

// Add test languages with localizations
var languages = new List<Language>
{
    new Language(LocalizationCode.ENG, JsonSerializer.Serialize(new List<LanguageLocationsDto>
```

```
{
    new(1, "Kyiv", "Ukraine"),
    new(2, "Lviv", "Ukraine"),
    new(3, "New York", "United States"),
    new(4, "Los Angeles", "United States"),
    new(5, "London", "United Kingdom")
}), JsonSerializer.Serialize(new Dictionary<string, string>
{
    { "Tour_Details_Title", "Tour Details" },
    { "Tour_NotFound", "Tour not found" },
    { "Tour_Guide", "Guide" },
    { "Tour_Locations", "Locations" },
    { "Tour_Dates", "Dates" },
    { "Tour_Duration", "Duration" },
    { "Tour_Price", "Price" },
    { "Tour_Rating", "Rating" },
    { "Tour_BookButton", "Book Tour" },
    { "Tour_LoginToBook", "Please login to book a tour" },
    { "TourType_Hiking", "Hiking" },
    { "TourType_Adventure", "Adventure" },
    { "TourType_Recovery", "Recovery" },
    { "Tours_Filter_Days", "days" },
    { "Tours_Filter_TourType", "Tour Type" },
    { "Messages_FailedToGetData", "Failed to get data" },
    { "Messages_TourNotFound", "Tour not found" },
    { "Messages_FailedToBookTour", "Failed to book tour" },
    { "Messages_TourBookedSuccessfully", "Tour booked successfully" }
})),
new Language(LocalizationCode.UKR, JsonSerializer.Serialize(new List<LanguageLocationsDto>
{
    new(1, "Київ", "Україна"),
    new(2, "Львів", "Україна"),
    new(3, "Нью-Йорк", "Сполучені Штати"),
    new(4, "Лос-Анджелес", "Сполучені Штати"),
    new(5, "Лондон", "Велика Британія")
}), JsonSerializer.Serialize(new Dictionary<string, string>
{
    { "Tour_Details_Title", "Деталі туру" },
    { "Tour_NotFound", "Тур не знайдено" },
    { "Tour_Guide", "Гід" },
```

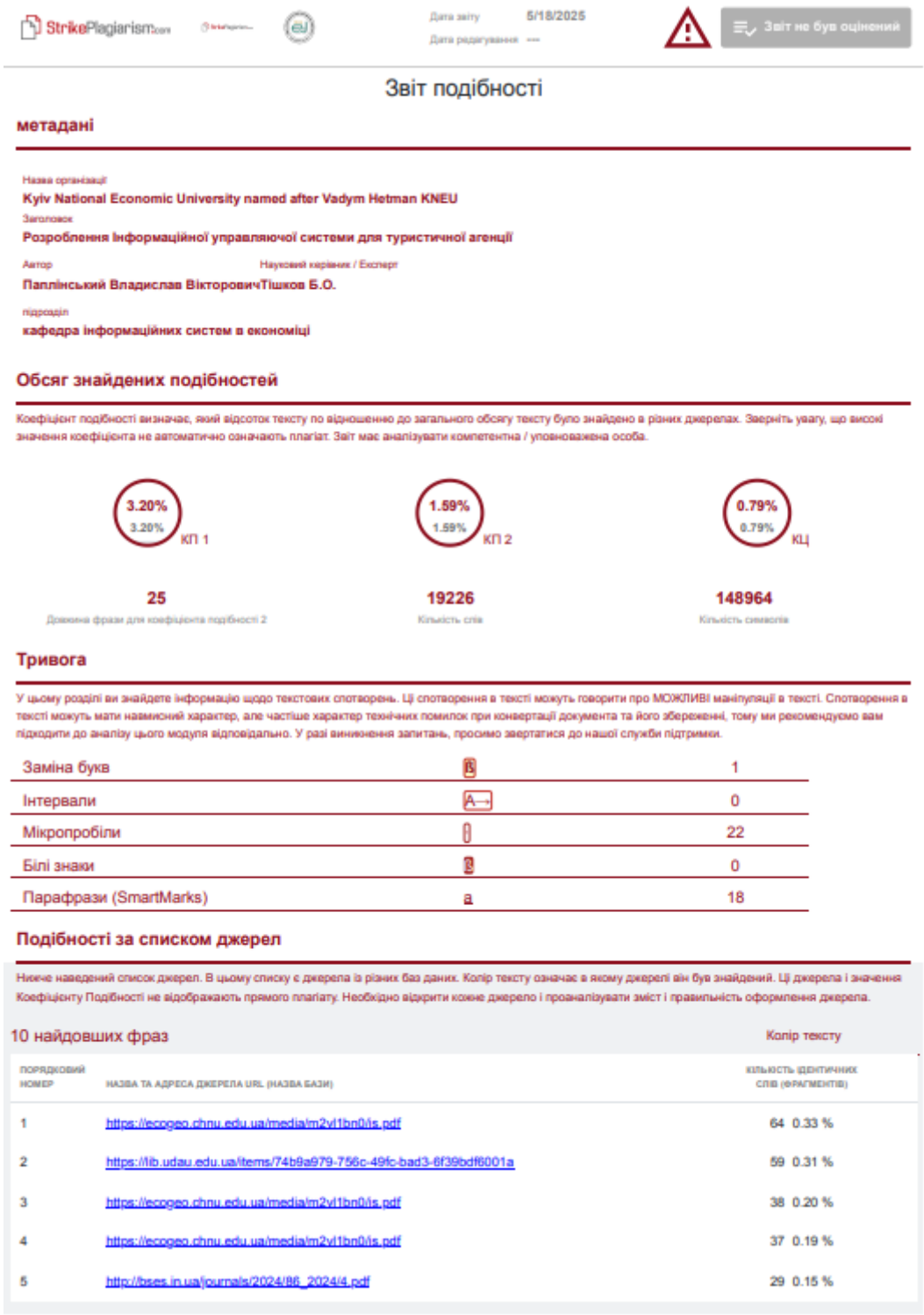
```
            { "Tour_Locations", "Локації" },
            { "Tour_Dates", "Дати" },
            { "Tour_Duration", "Тривалість" },
            { "Tour_Price", "Ціна" },
            { "Tour_Rating", "Рейтинг" },
            { "Tour_BookButton", "Забронювати тур" },
            { "Tour_LoginToBook", "Будь ласка, увійдіть, щоб забронювати тур" },
            { "TourType_Hiking", "Похід" },
            { "TourType_Adventure", "Пригода" },
            { "TourType_Recovery", "Відпочинок" },
            { "Tours_Filter_Days", "днів" },
            { "Tours_Filter_TourType", "Тип туру" },
            { "Messages_FailedToGetData", "Не вдалося отримати дані" },
            { "Messages_TourNotFound", "Тур не знайдено" },
            { "Messages_FailedToBookTour", "Не вдалося забронювати тур" },
            { "Messages_TourBookedSuccessfully", "Тур успішно забронньовано" }
        }))
    };

    DbContext.Accounts.AddRange(accounts);
    DbContext.Guides.AddRange(guides);
    DbContext.Tours.AddRange(tours);
    DbContext.TourInstances.AddRange(tourInstances);
    DbContext.TourInstanceRates.AddRange(rates);
    DbContext.TourBookings.AddRange(bookings);
    DbContext.Languages.AddRange(languages);

    DbContext.SaveChanges();
    DbContext.Database.EnsureCreated();
  }

  public void Dispose()
  {
    DbContext.Database.EnsureDeleted();
    DbContext.Dispose();
  }
}
```

Додаток В



StrikePlagiarism.com

Дата звіту 5/18/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kyiv National Economic University named after Vadym Hetman KNEU
Заголовок
Розроблення інформаційної управляючої системи для туристичної агенції
Автор
Паплінський Владислав Вікторович
Науковий керівник / Експерт
Тішков Б.О.
підрозділ
кафедра інформаційних систем в економіці

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

3.20% 3.20% КП 1

1.59% 1.59% КП 2

0.79% 0.79% КЦ

25
Довжина фрази для коефіцієнта подібності 2

19226
Кількість слів

148964
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		1
Інтервали		0
Мікропробіли		22
Білі знаки		0
Парафрази (SmartMarks)		18

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	https://ecogeo.chnu.edu.ua/media/m2vl1bn0/is.pdf	64	0.33 %
2	https://lib.udau.edu.ua/items/74b9a979-756c-49fc-bad3-6f39bdf6001a	59	0.31 %
3	https://ecogeo.chnu.edu.ua/media/m2vl1bn0/is.pdf	38	0.20 %
4	https://ecogeo.chnu.edu.ua/media/m2vl1bn0/is.pdf	37	0.19 %
5	http://bses.in.ua/journals/2024/86_2024/4.pdf	29	0.15 %

6	https://gozavtrak.blogspot.com/2020/06/blog-post_823.html	28 0.15 %
7	https://ecogeo.chnu.edu.ua/media/m2vl1bn0/is.pdf	26 0.14 %
8	https://ecogeo.chnu.edu.ua/media/m2vl1bn0/is.pdf	25 0.13 %
9	https://ecogeo.chnu.edu.ua/media/m2vl1bn0/is.pdf	22 0.11 %
10	Система електронного документообігу на основі приватного блокчейну [redacted] 5/30/2019 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	22 0.11 %

з бази даних RefBooks (0.03 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
джерело: Paperity		
1	HIGHER EDUCATION DEVELOPMENT TRENDS IN THE CONTEXT OF EUROPEAN INTEGRATION PROCESSES Наталія Мачинська;	5 (1) 0.03 %

з домашньої бази даних (0.66 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Система електронного документообігу на основі приватного блокчейну [redacted] 5/30/2019 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	77 (5) 0.40 %
2	Комп'ютерізація процесів підтримки прийняття рішень з управління віртуальним офісом [redacted] 6/22/2020 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	43 (5) 0.22 %
3	КМР Єлизаров [redacted] 11/27/2024 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра менеджменту)	7 (1) 0.04 %

з програми обміну базами даних (0.32 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Системи баз даних. Комп'ютерний практикум [redacted] 3/15/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)	21 (2) 0.11 %
2	дипломна робота Єсаян Л.В. [redacted] 5/29/2024 Interregional Academy of Personnel Management (Навчально-науковий інститут управління, економіки та бізнесу)	12 (1) 0.06 %
3	ФКНТ_2024_121_магістр_Добродуб М.А. [redacted] 11/21/2024 Ukrainian national aviation university (ФКНТ Кафедра інженерії програмного забезпечення)	11 (1) 0.06 %

4	КОНКУРЕНТОСПРОМОЖНІСТЬ ТА СТАЛИЙ РОЗВИТОК СУБ'ЄКТІВ ТУРИСТИЧНОГО БІЗНЕСУ [redacted] 6/4/2024 National University "Zaporizhzhia Polytechnic" (Кафедра "Туристичний, готельний та ресторанний бізнес")	10 (1) 0.05 %
5	Краснокутська Ю.В., Сегеда І.В., Безпалова А.С.docx [redacted] 11/28/2023 Publishing House "Helvetica" (Видавничий дім "Гельветика")	8 (1) 0.04 %

з Інтернету (2.19 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://ecogeo.chnu.edu.ua/media/m2vl1bn0/lis.pdf	257 (9) 1.34 %
2	https://lib.udau.edu.ua/items/74b9a979-756c-49fc-bad3-6f39bdf6001a	59 (1) 0.31 %
3	http://bses.in.ua/journals/2024/86_2024/4.pdf	29 (1) 0.15 %
4	https://gozavtrak.blogspot.com/2020/06/blog-post_823.html	28 (1) 0.15 %
5	https://ir.kneu.edu.ua/bitstreams/98d9b446-789f-4812-bc2a-7e6669494e60/download	27 (2) 0.14 %
6	https://speka.media/kiberbezpeka/ci-postrazdala-it-industriya-vid-kiberatak-analiz-ta-poradi-poradi-biznesu-vr8009	12 (1) 0.06 %
7	https://card-file.onlu.edu.ua/items/5b553bd4-518a-4311-94d6-984e9a37c306/full	10 (1) 0.05 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)