МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПЛАНУВАННЯ ТА ПІДТРИМКИ АКТИВНОГО ВІДПОЧИНКУ

Здобувач освіти гр. ІН.мз-11с Владислав ДЕРЕВ’ЯНЧУК

Науковий керівник
доцент, к.т.н. Ігор ШЕЛЕХОВ

В.о. Завідувача кафедри
доцент, к.т.н. Ігор ШЕЛЕХОВ

Суми 2022

Сумський державний університет
(назва вузу)

Факультет *ЦЗДВФН* Кафедра *Комп'ютерних наук*
Спеціальність *«Комп'ютерні науки»*

Затверджую:
в.о. зав. кафедрою_____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Дерев'янчуку Владиславу Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія планування та підтримки активного відпочинку*
затверджую наказом по інституту від "____" _____ 20 ____ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз предметної області 2) Проблематика дослідження пошуку активного відпочинку в Інтернет ресурсах 3) Підхід щодо побудови мікросервісної архітектури. 4) Моделювання мікросервісної інформаційної технології. 5) Розробка програмного забезпечення пошуку та пропозицій активного відпочинку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) *Діаграми класів сервісу, послідовностей процесів взаємодії, сутностей і зв'язків, варіантів використання загальної мікросервісної архітектури.*

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх.

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналіз предметної області*	*17.10.2022 – 20.10.2022*	
2.	*Проблематика дослідження пошуку активного відпочинку в Інтернет ресурсах*	*21.10.2022 – 24.10.2022*	
3.	*Підхід побудови мікросервісної архітектури*	*25.10.2022 – 28.10.2022*	
4.	*Моделювання мікросервісної інформаційної технології*	*29.10.2022 – 10.11.2022*	
5.	*Розробка програмного забезпечення пошуку та пропозицій активного відпочинку*	*11.11.2022 – 27.11.2022*	
6.	*Оформлення пояснювальної записки до дипломної роботи*	*28.11.2022 – 02.12.2022*	

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 89 стор., 54 рис., 4 табл., 2 додатки, 15 джерел.

Мета роботи — інформаційна технологія для задоволення потреб людей у зручному пошуку і пропонуванні активного відпочинку зі зручним, зрозумілим і надійним інтерфейсом, з широким вибором різних активностей, можливістю їх порівняння, оцінки і бронювання.

Об’єкт дослідження — веб-системи у підтримці і розвитку активних видів відпочинку у соціумі.

Предмет дослідження — принципи побудови застосунків на мікросервісній архітектурі для розвитку активного відпочинку.

Методи дослідження — метод створення мікросервісних систем з використанням сучасних підходів на базі хмарних технологій.

Результати — Створена інформаційна технологія пошуку і пропонування активного відпочинку, яка включає серверну і клієнтську частини і побудована на базі фреймворків сімейства Spring, з використанням хмарних технологій з реалізацією мікросервісної архітектури. Клієнтський застосунок створений на базі платформи Angular. Розроблені схеми та діаграми для опису процесів і структур, застосованих у системі, а саме: діаграми сутностей і зв’язків; послідовностей процесів; варіантів використання; класів і загальної мікросервісної архітектури.

МІКРОСЕРВІСНА АРХІТЕКТУРА ВЕБЗАСТОСУНКІВ; МЕТОДИ ПІДТРИМКИ ТУРИЗМУ З ВИКОРИСТАННЯМ ВЕБДОДАТКІВ; ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ У СФЕРІ АКТИВНОГО ВІДПОЧИНКУ; ХМАРНІ ТЕХНОЛОГІЇ; ІНФОРМАЦІЙНО РЕЛЯЦІЙНЕ ВІДОБРАЖЕННЯ; ГРУПА ФРЕЙМВОРКІВ SPRING.

ЗМІСТ

ВСТУП

Повсякденне життя людини неможливо уявити без новітніх технологій і пристроїв, які можнуть робити майже все, що ще сто років тому, важалось фантастикою. Можна отримати або поділитися майже будь-якою інформацію за лічені хвилини. За рахунок розвитку інформаційних технологій і модернізації, майже кожен може отримати послугу або товар за лічені години або навіть хвилини чи секунди. Можна навести безліч прикладів таких систем і сервісів, які полегшують життя, надаючи майже цілодобові послуги. Мабуть, перше, що спадає на думку – це банківські застосунки, без яких найпростіші операції вимагали б від клієнта банка безпосереднього візиту в відділення. Замовлення їжі, речей, квитків у теарт або на літак, навіть складно уявити без онлайн застосунків, які доступні у будь-який час та майже у бідь-якій точці планети.

Така популярнісь застосунків є цілком зрозумілою і навіть корисною, але не тільки для споживачів, а й для бізнесу бо саме завдяки цим програмам, як звичайні підприємці так і великі корпорації можуть пропонувати людям те що їм необхідно, знаходячись навіть на іншому кінці нашої планети.

Мабуть, всім колись задавали таке питання: «Щоб ти хотів робити у своєму житті, якщо б у тебе була купа грошей», і згідно статистичних даних, багато людей відповідають, що вони б подорожували та займалися чимось цікавим, але часто, вони навіть не знають чим.

Інколи, важко сказати, що робить нас щасливими, але щоб відповісти на це питяння варто просто відібрати у людини те, що для неї є цінним: свобода, їжа, соціум, сон, відпочинок, цей список може продовжувати і продовжувати. Всі ми хоч раз у своєму житті подорожували чи щось досліджували і це було неймовірно цікаво й захоплююче. Перша прогулянка у сосновому лісі або подорож у засніжені гори чи навіть просто новий район у місті. Все це було дивовижно-цікавим. Після чого, люди публікують нові

фотографії у соціальних мережах, які доречі є також складними веборієнтованими застосунками і отримують задоволення від того, що соціум реагує на їх існування у цьому світі.

Базуючись на потребах людей, дана робота присвячена інформаційній технології, яка допомагатиме людям швидко знаходити варіанти для подорожей, активного відпочинку, екскурсій і цікавих івентів. Пропозиції будуть робити саме організатори цих активностей. Ця система, має позитивно вплинути на економіку і розвиток туристичної галузі держави, а також покращити і полегшити життя людей у сфері туризму.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сфери активного відпочинку

Задовольнити всі потреби туризму в одному застосунку дуже важко, саме тому зазвичай використовується підхід «Розділяй і володарю», що дозволяє обмежитися певною під-сферою: готелі, екскурсії, квитки на літак чи автобус, тощо. Саме такий підхід допомагає зосередити увагу на конкретній проблемі і створити систему, яка б задовольняла потреби більшості населення.

Сферу активного відпочинку, можна вважати під-сферою туризму. Туристична сфера складається з багатьох складових під-сфер, в яких працює неймовірно-велика кількість людей. Для наочності, проаналізуємо деякі компоненти туристичної сфери і виділимо ту частину, для якої буде створена інформаційна система, для подальшого розвитку і популяризації активного відпочинку.

Туризм складається з деяких наступних складових:

- Транспортні сполучення між місцями
    - Літаки
    - Потяги
    - Автобуси
    - Кораблі
    - Автостоп
    - Тощо
- Місця проживання
    - Готелі
    - Хостели
    - Кемпінги
- Екскурсії різних типів
- Заклади харчування

- Активний відпочинок
    - Кемпінг
    - Хайкінг
    - Сапи
    - Спортивна Рибалка
    - Віндсерфінг
    - Серфінг
    - Стрибки з парашутом
    - Кемпінг
    - Скелелазіння
    - Альпінізм
    - Рафтинг
    - Прогулянка на повітряній кулі
    - Прогулянки на літаком
    - Прогулянки по бездоріжжю на квадрациклі
    - Байдарки
    - Велосипедний тур
    - Кінні прогулянки
    - Лижі, сноуборд
    - Кайтбординг
    - Кайтсерфинг
    - Вейксерфінг
    - Вейкбординг
    - Ліплення з глини
    - Відвідування цікавих і визначних місць
    - Тощо

Перелічивши різні варіанти активного відпочинку, можна побачити, що сфера доволі об’ємна і включає в себе багато способів проводження часу.

Також важливим фактором є те, що з кожним роком з’являються нові й нові варіанти, як можна активно відпочивати.

Проаналізувавши сферу активного відпочинку, можна виділити необхідні складові для створення мінімально-функціонуючої інформаційної технології (Рисунок 1.1).

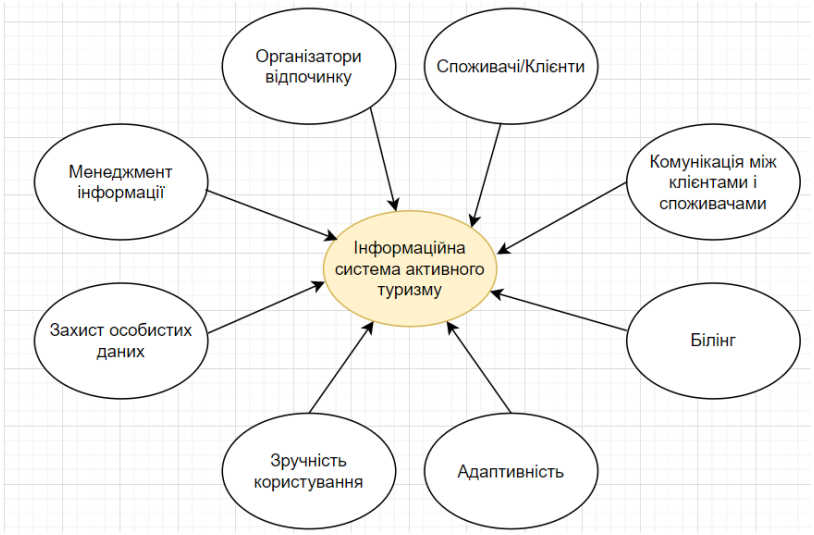


Рисунок 1.1 – Складові інформаційної технології активного відпочинку

Кожен з компонентів відіграє важливу роль для успішного функціонування системи, також, варто зауважити, що кількість компонентів може збільшуватись, зважаючи на потреби користувачів.

Організатори відпочинку мають великий вплив на систему, тому, для них має бути створений зручний і зрозумілий інтерфейс для взаємодії. Споживачі, також, мають мати зручний і доступний інтерфейс для того щоб застосунок був дійсно корисним і конкурентно-спроможним в порівнянні з іншими рішеннями.

Важливим елементом є взаємодія між клієнтом і користувачем, яка може бути реалізована різними способами і технологіями, починаючи від простого надання клієнтові або організатору контактних даних, закінчуючи власною системою листування.

Крім різного типу клієнтських інтерфейсів існують, також, такі важливі елементи, як менеджмент і обробка інформації, захист особистих даних,

адаптивність і масштабованість, а також білінг. Всі процеси мають відбуватися швидко. Зазвичай користувачі не люблять чекати поки все завантажиться. Крім того, якщо застосунок не матиме захисту то їм буде просто неможливо користуватися бо ніхто не хоче ділитися особистими даними з усіма людьми, більш того, захист особистої інформації регулюється законом.

Адаптивність і масштабованість дозволяє застосунку розроблювати нові функції і не впливати на вже існуючі функціональності. Таки чином, можна, дуже легко створювати експериментальні модулі, виявляти потреби користувачів і впроваджувати нові ідеї.

Білінг є також важливим компонентом в життєвому циклі застосунку, але даний модуль не є найважливішим і при розробці перших версій продукту може повністю буди відсутнім.

1.2 Аналіз конкурентних систем

Незважаючи на розвиток інформаційних систем, ще існує багато сфер людської життєдіяльності, які взагалі не мають інформаційних систем оптимізації або потребують покращення і розширення. Таким чином, проаналізувавши конкурентів, можна виділити найпопулярніші, схожі за деякими принципами системи у сфері туризму, які мають схожі функціональності з технологією яку представляє дана робота. До списку конкурентних систем належать:

- GetYourGuide – орієнтований на пошук екскурсій та івентів.
- Couchsurfing – орієнтований на пошук людини в якомусь місті, яка б могла провести безкоштовну екскурсію чи щось порадити у певній місцевості.
- Culture Trip – орієнтований на подорожі, в яких особлива увага приділяється життю в чужій культурі, а не в туристичній зоні, як звичайний турист.

- Tripadvisor – орієнтований на можливість спланувати поїздку до різних країни світу.
- Withlocals – орієнтований на мандрівки до місцевих людей, які пропонують приватні і персоналізовані тури.

Також існує велика кількість підприємців, які мають власний сайт і на якому вони пропонують свої послуги, але це лише декілька десятків пропозицій без реальних відгуків і оцінок, а також без можливості порівняти пропозиції конкурентів.

Незважаючи на достатню кількість різних систем для пошуку відпочинку, жодна з них не зосереджена саме на активного відпочинку і немає достатньої популярності і авторитетності в Україні. Таким чином, можна зробити висновок до проведеного аналізу конкурентів, що систему, саме для пошуку активного відпочинку, яка б інтенсивно розвивала цю сферу, поки не створили або не розвинули до достатньо популярного рівня.

1.3 Проблематика дослідження системи

У сучасному світі, можна знайти інформацію майже про все, починаючи від вирощування огірків, закінчуючи будівництвом ракет, але проблема полягає в тому, що на знаходження потрібної інформації може знадобитися дуже багато часу і сил. Враховуючи той факт, що мозок людини влаштований таким чином, щоб заощаджувати енергію, зазвичай, багатьом не вистачає терпіння знайти необхідну інформацію, і саме це стає причиною прийняття не найкращого рішення або вибору продукції чи послуг. Таким чином, однією з проблем, яку має вирішувати розроблена система є надання користувачу широкого вибору, який підкріплюється зворотнім зв’язком інший користувачів, що дозволяє іншим туристам краще зрозуміти рівень сервісу, який надається організатором і прийняти вигідне рішення і гарно провести час.

Зважаючи на той фактор, що існує достатня кількість підприємців, які працюють у сфері туризму, майже кожен з них має власний сайт або сторінку в соцмережах, де вони продають свої послуги активного відпочинку, такий підхід ведення бізнесу у даній сфері є складним, як для існуючих так і нових організаторів івентів бо потребує значного інвестування часу. Підприємці мають слідкувати за багатьма чинниками, такими як: актуальність контенту, привабливість інформації про відпочинок і вигляд сайту в цілому, а також інколи є потреба у кваліфікованій допомозі з боку фахівців зі сфери інформаційних технологій, що спричиняє додаткові витрати часу і коштів. Враховуючи підприємців, які тільки починають ріст у сфері туризму, їм дуже важко конкурувати з існуючими підприємцями, хоч їх сервіс може бути набагато кращим. Таким чином, другу проблему, яку має вирішувати система є зручність ведення бізнесу у даній сфері, за рахунок зручного і розширюваного інтерфейсу, а також «здорової» конкуренції між різними за досвідом і сервісом підприємцями. Організатори можуть надавати детальну інформацію, а також, розширювати або редагувати її, що відрізняє дану систему від статичних форм представлення інформації іншими локальними сайтами підприємців.

Інколи, буває так, що людина сама не знає чого хоче, а коли невідомо, що шукати то дуже складно це знайти. Саме так буває, коли люди хочуть спробувати, щось нове, але не знають де і як це знайти. Таким чином, розроблювана інформаційна технологія, направлена на ознайомлення і пропонування клієнтам нових і різноманітних видів відпочинку, що має підвищувати їх інтерес до активного проводження часу і задовольняти потреби у оригінальному відпочинку.

2 ПОСТАНОВКА ЗАВДАННЯ

2.1 Визначення мети дослідження системи

Беручи до уваги декілька проведених опитувань серед людей різної вікової категорії і різного рівня заможності, можна зазначити, що активний відпочинок є більш популярним ніж перегляд фільмів чи просто відпочинок вдома (Рисунок 2.1).

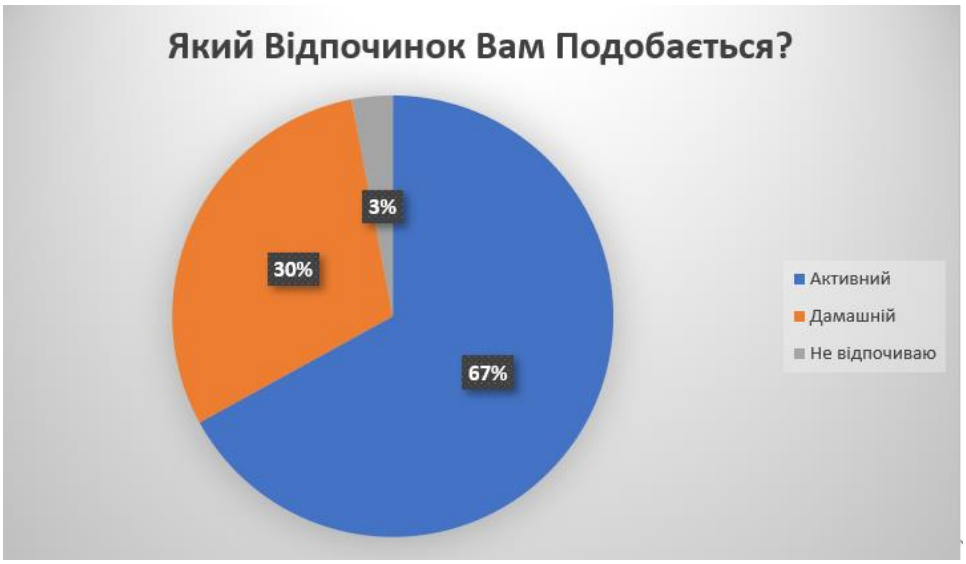


Рисунок 2.1 – Опитування про улюблений вид відпочинку

Пізнання світу є невід’ємною частиною всього людського життя. Для виявлення обізнаності людей, було проведено опитування на знання деяких видів активно відпочинку (Рисунок 2.2).

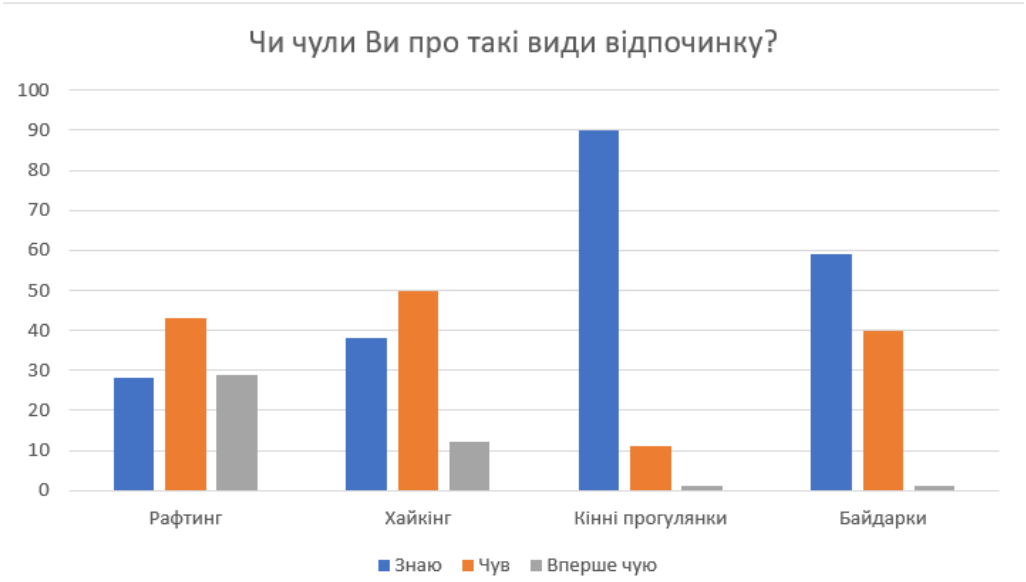


Рисунок 2.2 – Обізнаність людей у деяких видах активного відпочинку

Важливим фактором будь-якої системи є визначення її цінності для населення. Тому, було проведено опитування для виявлення корисності головних ідей системи (Рисунок 2.3).

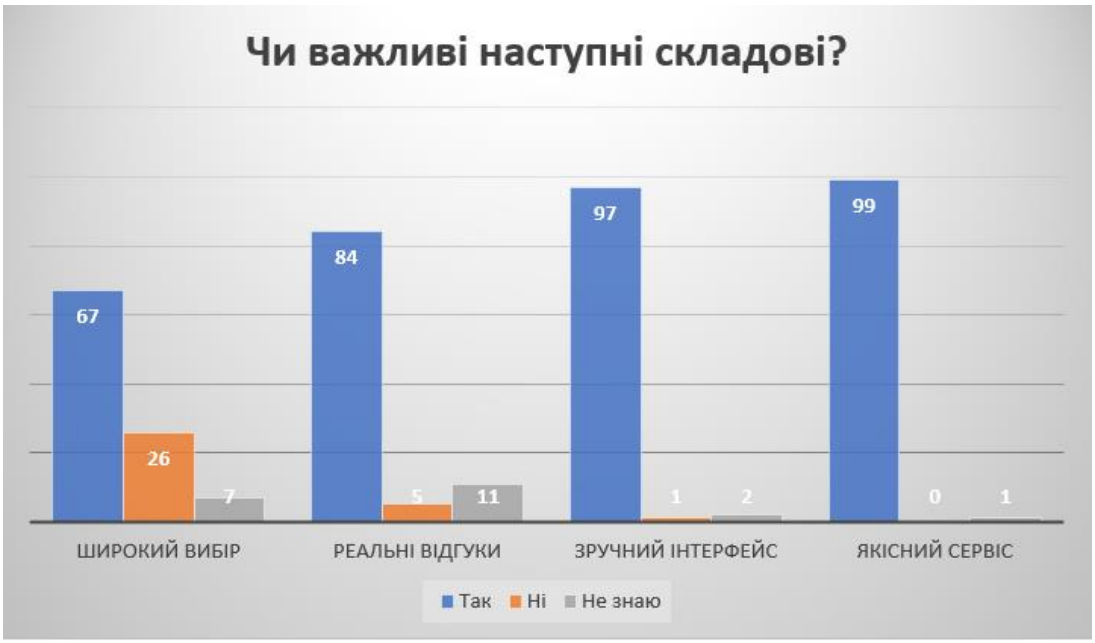


Рисунок 2.3 – Опитування важливості складових системи

Базуючись на проведених опитуваннях можна сформувати й визначити мету даної системи.

Мета – задоволення потреб людей у зручному пошуку і пропонуванні відпочинку, за рахунок системи, яка представляє зручний, зрозумілий і

надійний інтерфейс, як для туристів так і організаторів, а також надає користувачу широкий вибір різних активностей, можливість їх порівняти, оцінити і забронювати.

2.2 Постановка завдання дослідження

Завданням дослідження є: створення інтелектуальної технології, що дозволить виконувати проєктування і розроблення системи, серверна частина якої, буде базуватиметеся на мікросервісній архітектурі, а також клієнтський інтерфейс для взаємодії користувачів через браузер. Перед етапом розробки провести етап планування, який включатиме схематичну побудову архітектури, внутрішніх і зовнішніх процесів. Розробити моделі бази даних, створивши діаграми сутностей і зв’язків. Провести аналіз сучасних технологій і рішень для подальшого використання. При розробці системи, базуватися на вирішенні актуальних проблем у даній сфері. Для подальшого дослідження необхідно розробити такі функціональності для користувачів:

- Реєстрація і авторизація двох типів користувачів, а саме туристів і гідів.
- Створення і менеджмент пропозицій активного відпочинку організаторами.
- Пошук по критеріях пропозицій.
- Бронювання і додавання турів до улюблених.
- Написання відгуків.

Спроєктувати і розробити гнучку мікросервісну архітектуру, а також інтерфейс для взаємодії клієнтських застосунків, за допомогою якого відбуватиметься комунікація з серверною частиною. Для цього необхідно розробити такі мікросервіси :

- Конфігурацій – для централізованого доступу і налаштування інших інтерфейсів

- Навантаження і моніторингу даних – для моніторингу і розподілення навантаження.
- Шлюзу – для надавання точки вході клієнтським застосункам через один інтерфейс.
- Організаторів турів – для оброблення інформації про організаторів турів.
- Подорожей – для оброблення інформації про подорожі.
- Туристів – для оброблення інформації про туристів.
- Коментарів – для оброблення коментарів.
- Аутентифікації – для реєстрації.

3 ІНФОРМАЦІЙНИЙ ОГЛЯД МЕТОДІВ РІШЕННЯ

3.1 Технології мікросервісної архітектури

Одним з найважливіших факторів сучасної інформаційної системи є її масштабування функціональних можливостей. Розширення функцій і розподілення навантаження стає можливим завдяки слабкому зв'язку компонентів програми. Слабка залежність може бути досягнена завдяки мікросервісній архітектурі [1]. Проаналізувавши існуючі методи створення сучасних застосунків на базі мікросервісної архітектури, можна виділити, доволі конкурентний фреймворк Spring Boot Cloud у зв'язці з іншими під-фреймворками сімейства Spring.

Мікросервіс – це найпростіший елемент, який приймає і оброблює вхідні запити для здійснення певних операції. Мікросервісом може бути програма на сервері, яка доступна цілодобово та без вихідних, або функція, яка викликається, коли відбувається подія. Тобто, мікросервіс – це функція або набір функцій, доступних через певний інтерфейс у мережі [8]. На відміну від монолітного застосунку, мікросервісний додаток, побудований, як набір відносно невеликих і незалежних сервісів, які комунікують через комп'ютерну мережу за певними протоколами передачі даних (Рисунок 3.1).

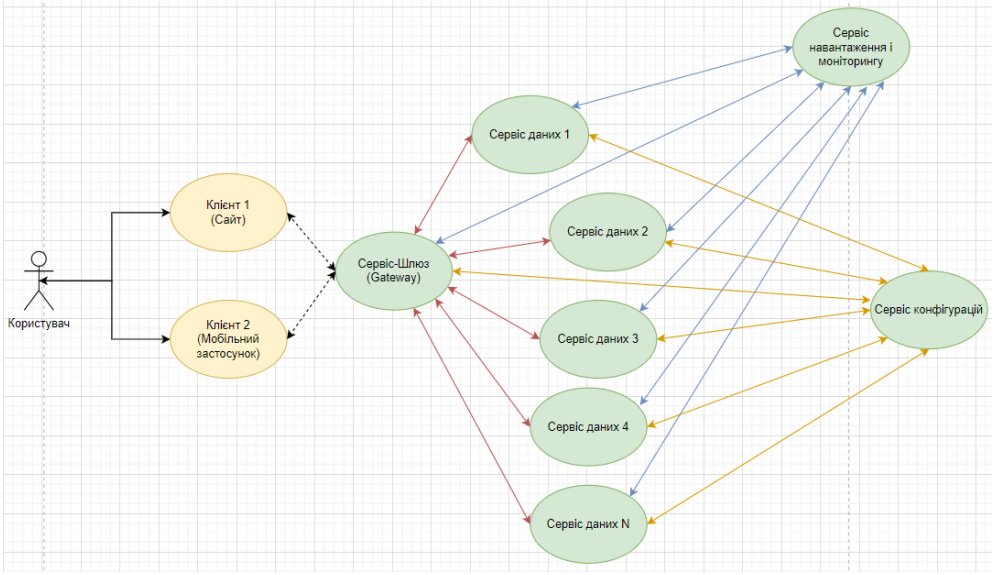


Рисунок 3.1 – Шаблон мікросервісної архітектури

Можна сказати, мікросервіси – це ті самі логічні модулі монолітного додатка, які розподілені у комп’ютерній мережі, замість того, щоб працювати в рамках одного процесу [2].

Складовими компонентами мікросервісної архітектури для веб-застосунку є:

- Клієнти, які використовують API – програмний інтерфейс застосунку. Зазвичай клієнт використовує HTTP протокол, але можливе застосування й інших, такого, як наприклад GRPC в залежності від реалізації логіки мікросервіса та клієнта.
- Мікросервіс Шлюз або Gateway. Саме через цей сервіс відбувається контроль доступу і комунікація клієнтів та сервісів даних. Оскільки, мікросервіси даних зазвичай захищені від зовнішнього і прямого доступу. Таким чином покращується захист сервісів, які мають доступ до даних і полегшується розробка і налаштування безпеки доступу до кожного сервісу [3].
- Мікросервіси даних – сервіси які мають в бізнес-логіку і можуть мати доступ до сховищ даних.
- Мікросервіс конфігурацій. Відповідальний за надання налаштувань іншим мікросервісам при їх завантаженні, а також в процесі виконання. При зміні конфігураційних файлів в репозиторії налаштувань, також, цей сервіс може оновлювати налаштування інших сервісів динамічно.
- Сервіс навантаження і моніторингу. Даний мікросервіс реєструє сервіси і стежить за їх станом і завантаженістю. Він може автоматично створювати нові сутності певного мікросервіса та розподіляти навантаження між ними.
- Сховища даних. В залежності від вимог бізнесу використовуються різні бази даних, наприклад, такі, як SQL або

NOSQL. Гарною практикою, вважається, коли тільки один мікросервіс має доступ до певних баз даних. Інші мікросервіси отримують інформацію іншого сервісу через його інтерфейс.

3.2 Веб-сервіси на базі фреймворку Spring Boot

Існує безліч підходів і технологій для створення веб-застосунків, з широким вибором допоміжних фреймворків і бібліотек. Фреймворк Spring Boot є одним з найпопулярніших фреймворків для побудови складних веб-застосунків і не тільки. Група фреймворків Spring є ідеальним варіантом для розробки мікросервісних застосунків, так як має велику кількість вже готових модулів, які потрібно тільки правильно налаштувати, і, звісно, розробити власну бізнес логіку [4]. Зазвичай, для налаштування використовуються спеціальні файли з розширенням properties або yml (Рисунок 3.2).

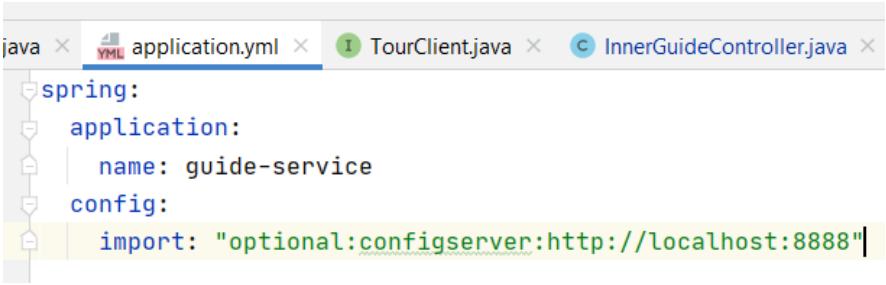


Рисунок 3.2 – Файл конфігурації за замовчанням для сервісу гідів

Ці файли містять як критичні, так і допоміжні налаштування, такі, як: порт, який буде використовувати програма; ім’я сервісу; дані доступу до бази даних тощо.

Важливим елементом системи є доступ до сховища даних. Для швидкої імплементації і взаємодії з реляційною базою даних, можна використати фреймворк Spring Data JPA у зв’язці з бібліотекою Hibernate, що дозволяє скористатися всіма перевагами технології ORM – технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов

програмування, створюючи «віртуальну об'єктну базу даних» [9]. Таким чином, можна створити певний клас, який описуватиме якусь сутність. У випадку даного проекту, сутностями є: турист, гід, тур тощо.

Описавши клас і його складові за допомогою анотацій на етапі компіляції, проаналізуються створені налаштування і створяться таблиці у базі даних, відповідно до налаштувань. Таким чином, непотрібно реалізовувати базові запити до бази даних, бо фреймворк знає про структуру бази і може виконувати, так звані, CRUD операції, базуючись на налаштуваннях класів, навіть, якщо між таблицями наявні певні відносини, як один до багатьох або навіть багато до багатьох. Через спеціальні інтерфейси Repository, здійснюються запити до сховища. Ці інтерфейси містять базові операції створення, пошуку, оновлення і видалення. Також, наявна можливість створення додаткових спеціальних запитів, необхідних для бізнес логіки. Існує декілька шляхів створення таких спеціальних запитів. Один зі способів – це написання запитів майже вручну. Іншим способом, зазвичай більш раціональним, є використання назви методу і його параметрів для створення запиту. Назва методу описує що і як треба зробити, а параметри задають обмеження пошуку(Рисунок 3.3).

```
@Repository
public interface GuideRepository extends JpaRepository<Guide, UUID> {
    Optional<Guide> findByEmail(String email);
}
```

Рисунок 3.3 – Опис запиту до сховища за допомогою методу

Інколи, запити можуть бути дуже складними. В такому випадку, доцільніше буде написати запит власноруч. Також, вже є інтегрована можливість використовувати пагінацію і сортування. Для цього необхідно налаштувати додатковий параметр методу.

Для взаємодії з нереляційною базою даних, існує спеціальний під-фреймворк – Spring Data MongoDB, який дозволяє використовувати сховище

MongoDB. У даній системі, цей підхід використовується для зберігання зображень у бінарному вигляді, безпосередньо, у базі даних. Розглядаючи даний підхід на рівні імплементації, то можна зазначити, що він дуже схожий з технологіями використаними у фреймворку Spring Data JPA. Ключова відмінність полягає у тому, що немає повної можливості налаштувати структуру сутності, що насправді регламентується сховищами NOSQL.

Робота з даними є дуже важливим елементом будь-якої системи, але не меш важливим є захищеність системи, від несанкціонованого доступу. Спеціально для забезпечення швидкого і спрощеного налаштування захисту сервісів, існує фреймворк Spring Security, який дозволяє налаштувати дуже гнучкі правила доступу, ураховуючи різні ролі користувачів і обмеження за різними параметрами. У зв’язці з цим підходом, також можна використовувати такі технології захисту, як сесії, JWT, oAuth2 тощо. JWT - це стандарт токена доступу на основі JSON, стандартизованого в RFC 7519. Як правило, використовується для передачі даних для аутентифікації в клієнт-серверних програмах. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який надалі використовує цей токен для підтвердження своєї особи [10].

3.3 Хмарні технології

З кожним роком все більше й більше популярності набирають хмарні технології. Саме тому з кожним роком з’являються нові підходи побудови систем таких типів.

Spring Cloud надає розробникам інструменти для швидкого створення деяких загальних шаблонів у розподілених системах, наприклад: керування конфігураціями, виявлення служб, автоматичні вимикачі, інтелектуальна маршрутизація, мікро-проксі, керуюча шина, одноразові токени, глобальні блокування, вибори керівництва, розподілені сесії, стан кластера. Координація розподілених систем призводить до шаблонів, і за допомогою

Spring Cloud розробники можуть швидко створити служби та програми, які реалізують ці шаблони. Вони добре працюватимуть у будь-якому розподіленому середовищі, включаючи власний ноутбук розробника, центри обробки даних і керован і платформи, такі як Cloud Foundry [11].

3.4 Автоматична збірка проектів

Сучасну систему майже неможливо уявити без функіональності, яка дозволяє автоматично, завантажувати необхідні бібліотеки, компілювати і збирати проєкт, а також всановлювати критично необхідні параметри для всієї програми. Для складних систем, використання таких технологій надзвичайно важливе, так як дозволяє пришвидшити час розробки, що в свою чергу, заощаджує час та гроші розробників і замовників відповідно. Для систем, розроблюваних на мові програмування Java з використанням фреймворків із сімейства Spring, найпопулярнішими системами автоматичної збірки є Gradle і Maven. Кожна з них має свої переваги і недоліки, як і будь-що в нашому світі. І Gradle, і Maven забезпечують конвенцію над конфігурацією. Maven надає дуже жорстку модель, яка робить налаштування випробовуючим, а іноді й неможливим. Хоча це може полегшити розуміння будь-якої збірки Maven, якщо у вас немає особливих вимог, це також робить її непридатною для багатьох проблем автоматизації. Gradle, з іншого боку, створено з урахуванням уповноваженого та відповідального користувача [12].

3.5 Підходи щодо побудови клієнтських застосунків

Для побудови багатофункціональних веб-клієнтів, які можуть бути розширенні, завдяки гнучким підходам розробки і проєктування, найкраще використовувати фреймворки, які містять багато розроблених модулей і бібліотек, що в сукупності спрощують і тим самим, розширюють можливості інтегрування різних функцій у складну систему. Гарним прикладом є такі платформи, як Angular, React, Vue.js. Вони дозволяють створювати

застосунок вже з готових елементів, розширюючи його власною конфігурацією і логікою. Вони містять вже готові і оптимізовані елементи для надсилання HTTP запитів до серверної частини, створення логічних модулей, використання одного коду в різних місцях. Фреймворк Angular складається з трьох основних компонентів: моделі; представлення і контролери. Однак структура програми Angular є фіксованою та складною. Angular дозволяє розробникам розділяти частини застосунку на окремі файли, полегшуючи повторне використання шаблонів і кодових баз в інших проектах [13]. Вбудована утиліта, дозволяє користувачам створювати та керувати проектами з командного рядка. Це автоматизує такі завдання, як створення проектів, додавання нових контролерів, модулів, інтерфейсів, тощо [5]. Angular - це безкоштовна платформа веб-додатків із відкритим вихідним кодом на основі TypeScript, очолювана командою Angular у Google і великою спільнотою розробників [7].

Створення веб-додатків, які можуть задовольнити потреби користувачів, не є тривіальним завданням. Якість і складність застосунків постійно зростає, як і очікування користувачів. Angular існує, щоб допомогти розробникам доставляти програми відповідати цим вимогам [6].

Використовуючи підхід односторінкоих застосунків можна динамічно підвантажувати данні, відповідно до певних дій користувача. Такий підхід, зменшує навантаження на серверну частину, щляхом зменшення даних, що передаються по мережі. Також, це дозволяє зменшити заткимки, утримати користувача на сторінці і надати задоволення клієнту, від користування застосунком за рахунок його шивдкості і плавності.

4 МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

4.1 Архітектурна модель системи

Будь-який проєкт повинен починатися з моделювання, щоб урахувати різні фактори і елементи розробки, і цим самим уникнути проблем при імплементації. Розроблена модель мікросервісної архітектури для даної технології представляє собою невід'ємну складову для якісної реалізації.

Важливим компонентом являється сервіс Шлюз або Gateway, який в подальшому буде виконуватиме роль роутера для клієнтських запитів(Рисунок 4.1). Саме за допомогою цього мікросервісу, клієнти будуть здійснювати доступ до інших – внутрішніх мікросервісів.

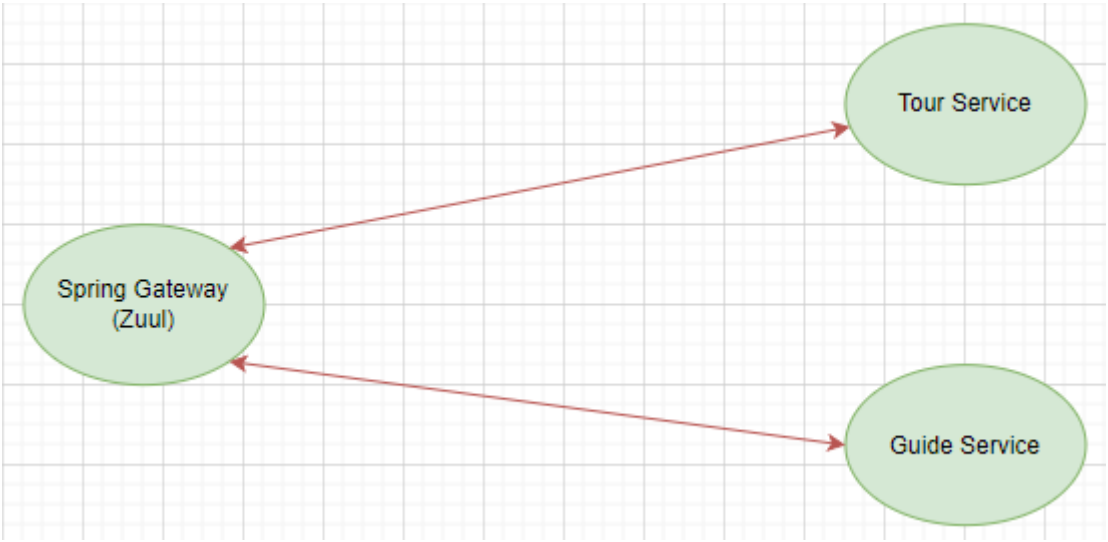


Рисунок 4.1 – Шлюз з сервісами даних, а саме турів і гідів

Конфігурація системи є однією з найважливіших складових, тому, мікросервіс налаштувань, який надає конфігурацію всім іншим сервісам є невід'ємною складовою даної системи (Рисунок 4.2). Без нього інші мікросервіси не запустяться, або запустяться у обмеженому режимі. Також, він надає можливість динамічної зміни некритичних параметрів, що надає змогу змінити налаштування сервісу без його зупинення, більш того, централізоване зберігання налаштувань зменшує час налаштування всієї системи.

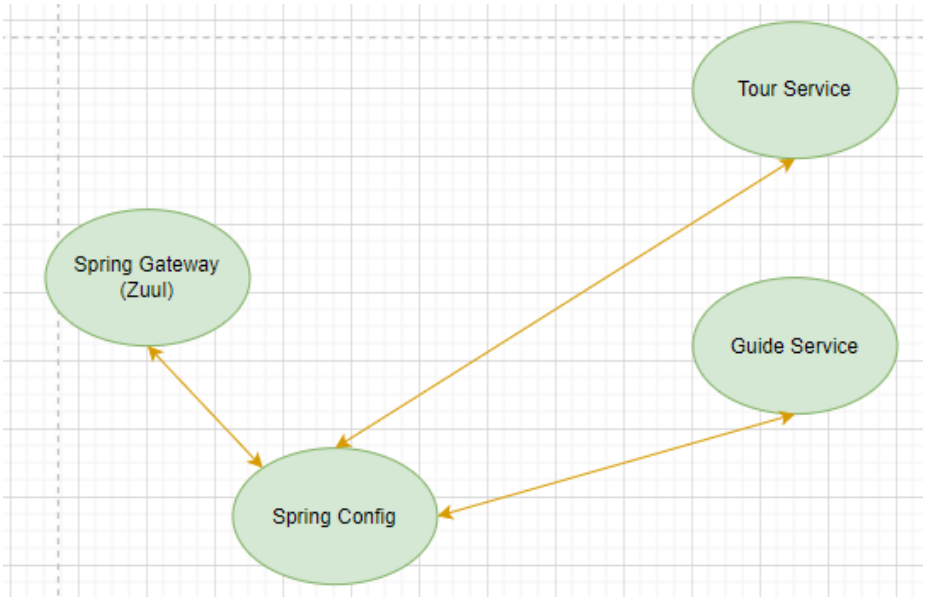


Рисунок 4.2 – Сервіс конфігурацій з деякими сервісами, що потребують налаштувань

Ще одним важливим елементом є сервіс реєстрації і моніторингу, який відповідає за моніторинг всіх інших мікросервісів (Рисунок 4.3).

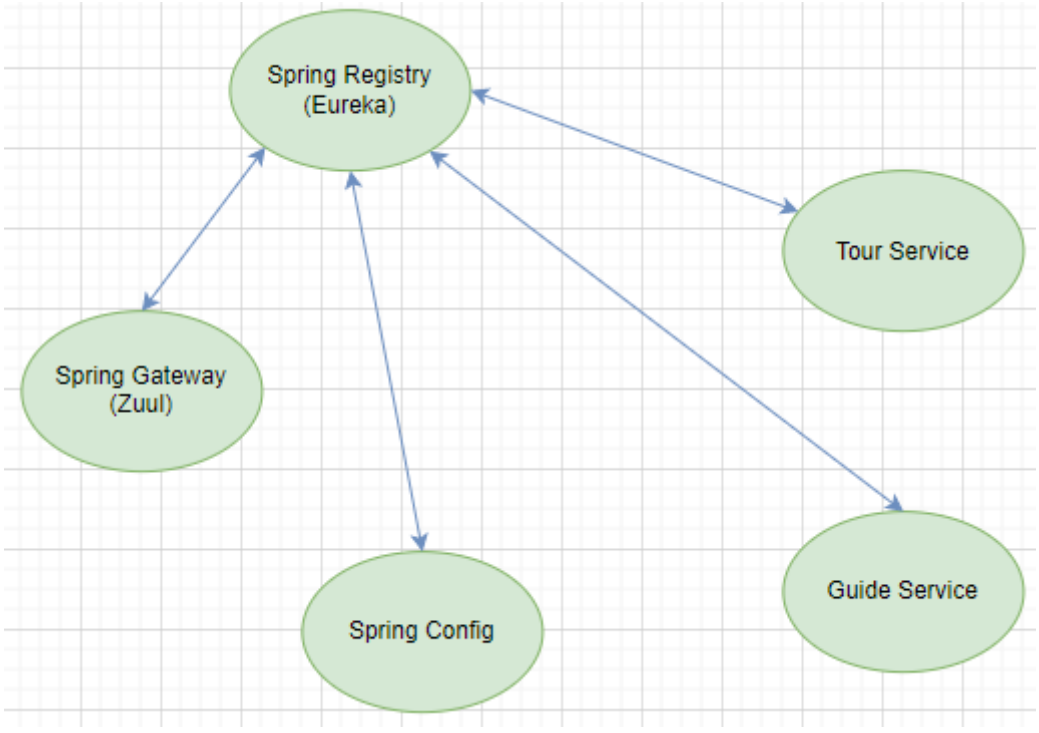


Рисунок 4.3 – Сервіс реєстрації і моніторингу з деякими іншими сервісами

Поміж критично важливих комунікацій, таких, як з сервісом конфігурацій, існують також внутрішні комунікації між сервісами

даних (Рисунок 4.4). Зв’язки між ними також є дуже важливими для злагодженого функціонування всієї системи, але не критичними. У випадку коли один мікросервіс вийде з ладу він не зупинить функціонування всієї системи.

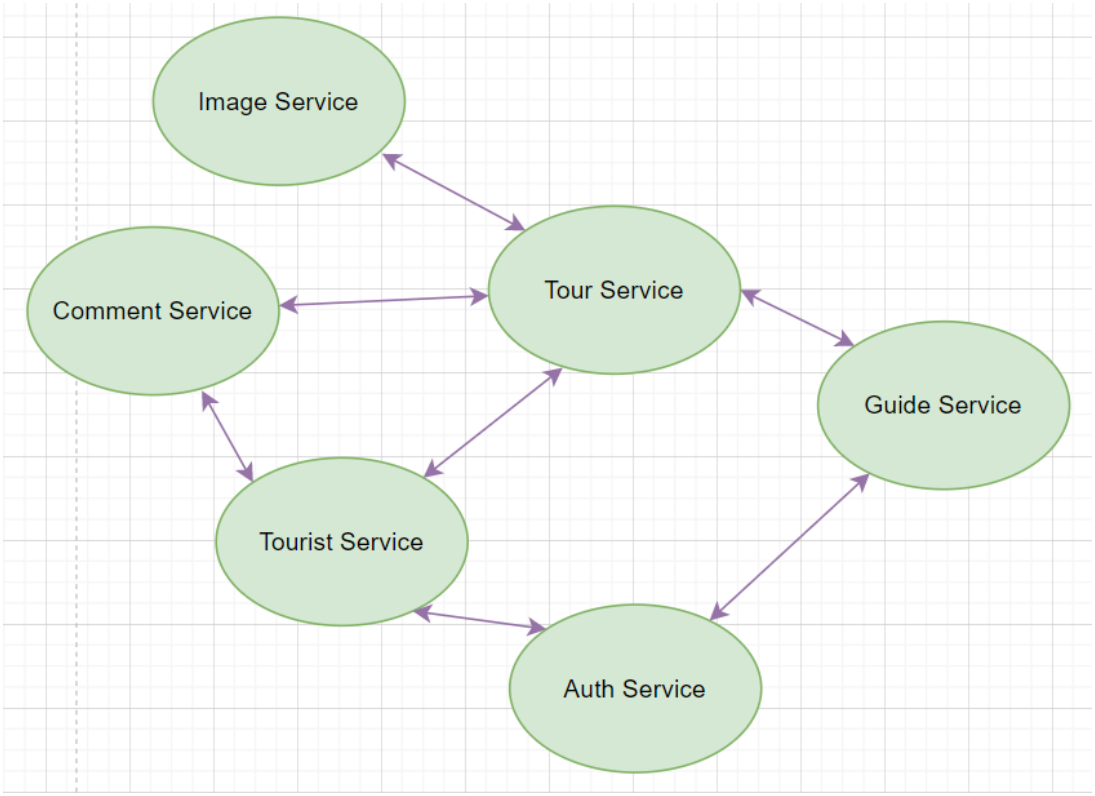


Рисунок 4.4 – Взаємодія між сервісами даних

Майже кожен сервіс даних, який має потребу у збереженні і відтворенні даних і повинен мати зв’язок з власною базою даних, а обмін даними між сервісами повинен відбуватися за рахунок API мікросервісів (Рисунок 4.5).

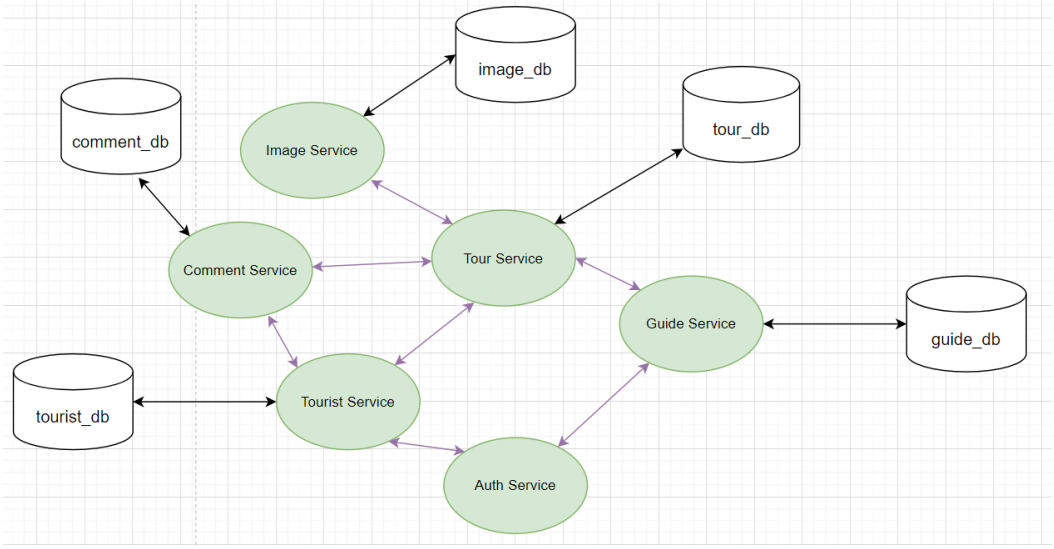


Рисунок 4.5 – Взаємодія між сервісами даних і сховищами даних

Для взаємодії користувачів з системою, необхідно складовою є клієнтський застосунок, через який клієнт взаємодіє з серверною частиною через певний інтерфейс (Рисунок 4.6).

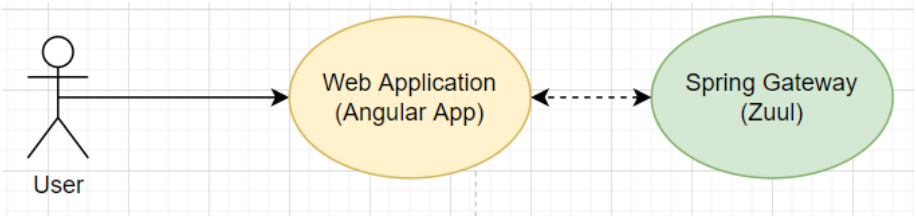


Рисунок 4.6 – Взаємодія користувача з системою через клієнтський інтерфейс

Розділивши загальну схему на складові, можна краще зрозуміти внутрішні процеси і полегшити розробку системи, але також важливо мати загальну схему мікросервісної архітектури для розуміння всієї системи в цілому (Рисунок 4.7).

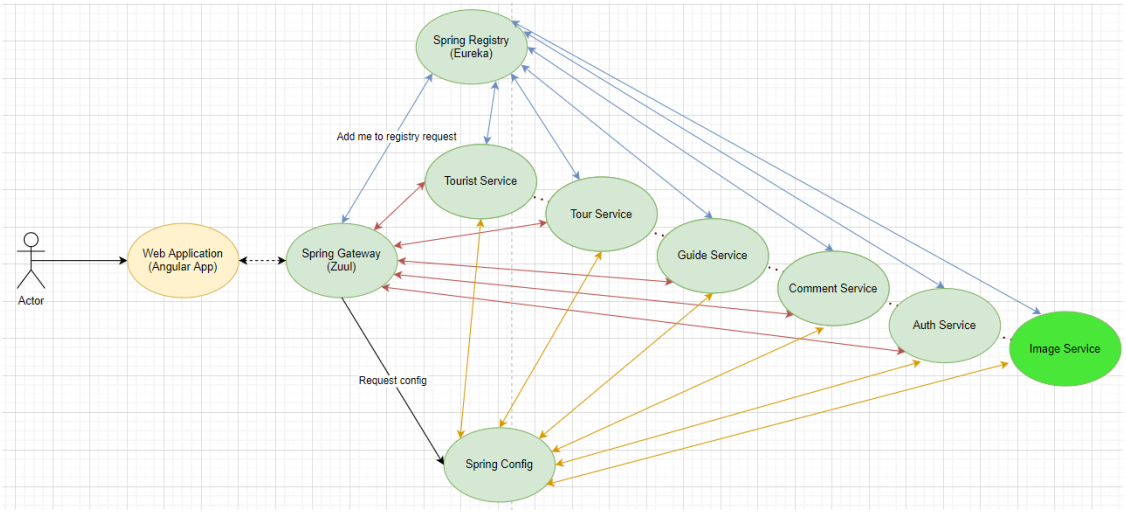


Рисунок 4.7 – Загальна мікросервісна архітектура системи

4.2 Розробка баз даних

Невід'ємною складовою будь-якої системи є дані які треба обробляти і зберігати в певному вигляді. У даній системі також використовуються два різних сховища даних для збереження структурованої інформації, наприклад, про туристів чи активний відпочинок, тощо і сховище для зображень які може завантажити організатор для евентів, які він створює.

Таким чином, для зберігання інформації про туристів, гідів, екскурсії, тощо, було обрано реляційну базу даних і систему управління базами даних MySQL. Враховуючи особливості і специфіку підібраних технологій, таких як Spring Data і Hibernate, які в свою чергу використовують підхід ORM – об'єктно-реляційного відображення, вони дозволяють автоматично створити таблиці бази даних базуючись на класах – моделях, які містять необхідні конфігурації, для автоматичної генерації і валідації структури таблиць у сховищі даних. Класи моделі – це класи які описують основні сутності системи, такі як турист, гід, тур, коментар, тощо. Hibernate не потребує спеціальних таблиць або полів бази даних і генерує більшу частину SQL під час ініціалізації системи, а не під час виконання [14].

Так як, мікросервісна архітектура, передбачає, використання кожним мікросервісом власного сховища даних, то кожна база зберігає тільки ті дані,

які обробляються певним мікросервісом. Сховище даних турив містить в собі інформацію про самі заходи і те з чим вони мають безпосередній зв’язок, а саме локація і тип туру (Рисунок 4.8).

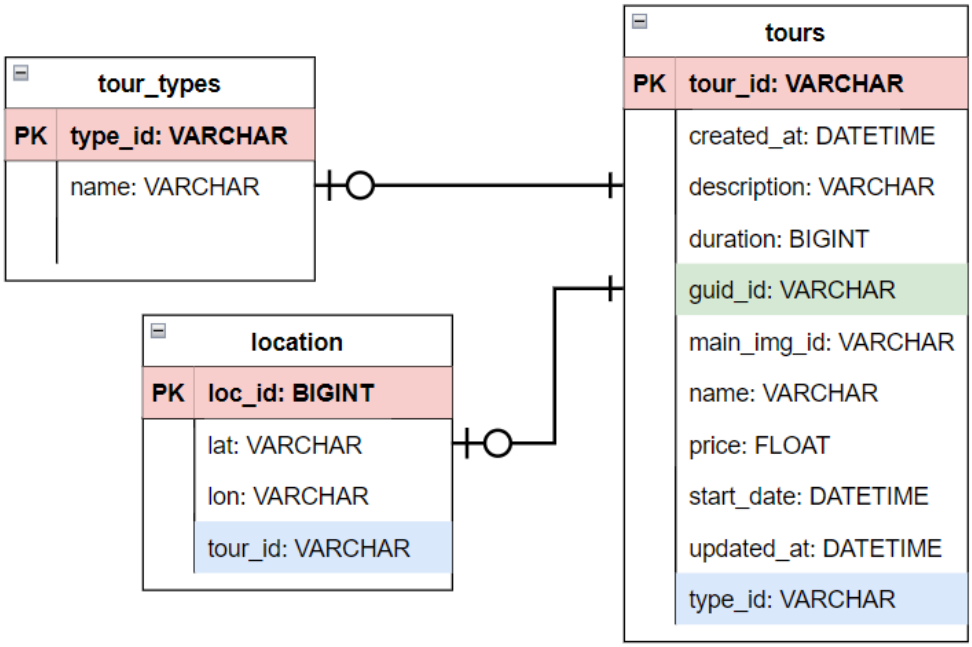


Рисунок 4.8 – Діаграма сховища даних турів

Важливу роль у системі відіграють користувачі, це значить, що необхідно зберігати інформацію про них і їх зв’язки з іншими компонентами. На даному етапі у системі використовуються два типи користувачів, а саме туристи і організатори турів. Кожен вид користувачів має власний мікросервіс для обробки даних, і відповідно власне сховище, де, наприклад, для туристів зберігається інформація про самих туристів і тури, які були заброньовані чи вподобані певним користувачем (Рисунок 4.9).

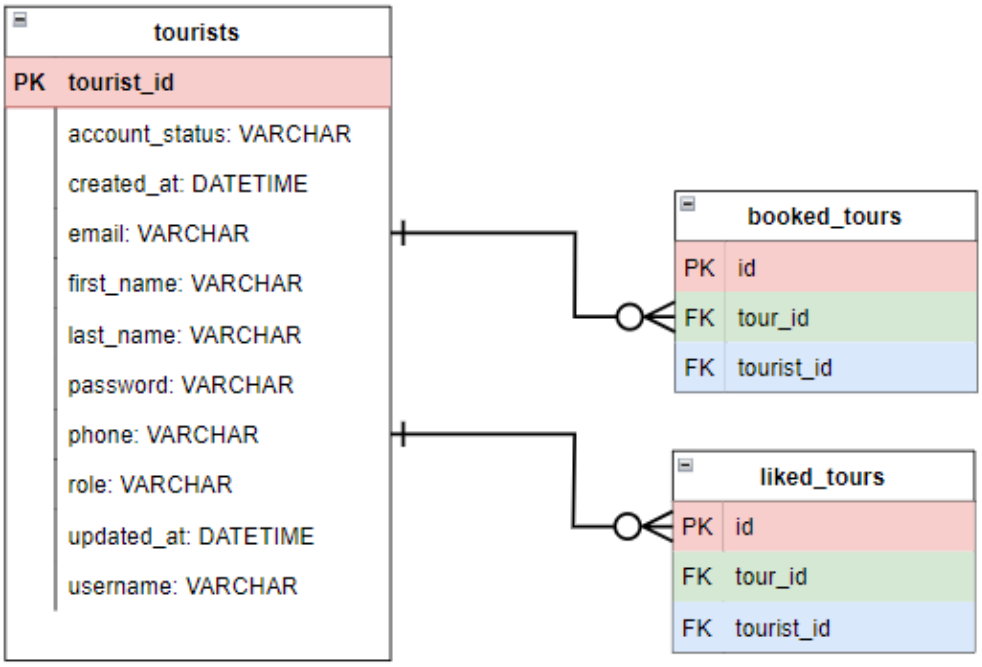


Рисунок 4.9 – Діаграма сховища даних туристів

Організатори турів мають також велике значення у системі і інформація про них також зберігається у реляційному сховищі з певною структурою (Рисунок 4.10).

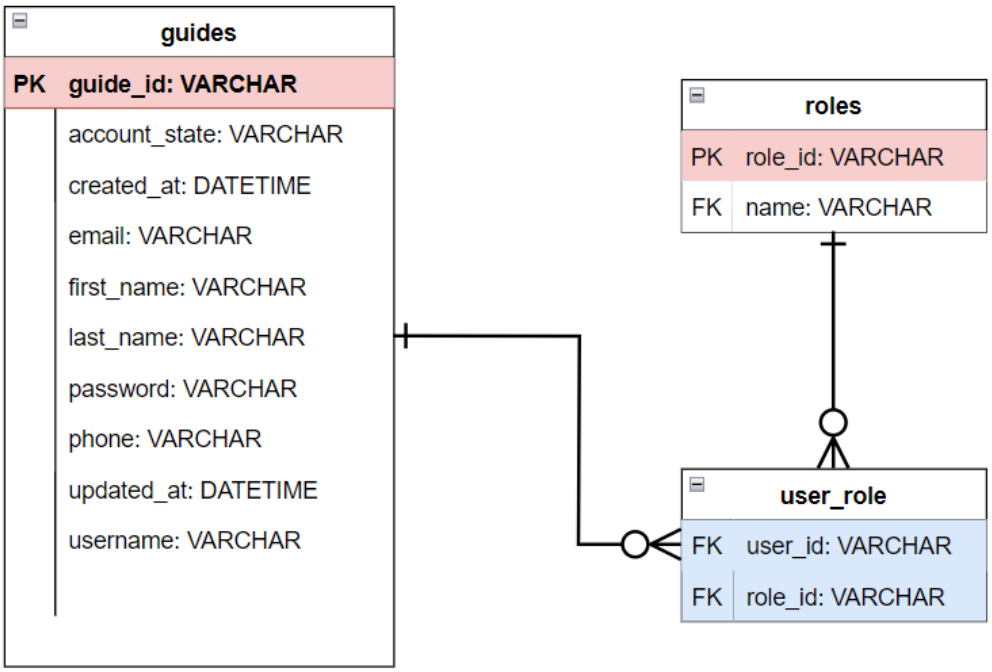


Рисунок 4.10 – Діаграма сховища даних гідів

Неймовірно корисною є інформація про досвід інших клієнтів, як для інших туристів так і для організаторів. Тому створюємо діаграму сутностей і зв’язків для сервісу коментарів (Рисунок 4.11) і рейтингу (Рисунок 4.12).

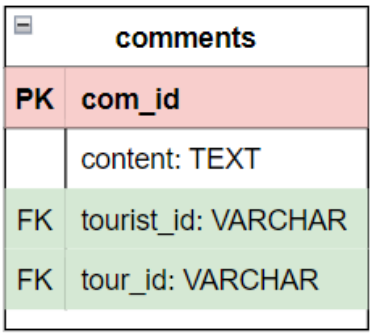


Рисунок 4.11 – Діаграма сховища даних коментарів

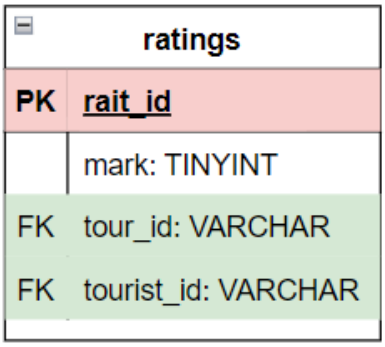


Рисунок 4.12 – Діаграма сховища даних рейтингів

Для зберігання і відтворення зображень турів використовується документо-орієнтована база даних MongoDB. Завдяки даному підходу можна легко зберігати зображення і бінарному вигляді безпосередньо в сховищі, а також зберігати гнучкість у структурі елементів сховища. Кожне зображення має ідентифікатор, назву, тип, розмір і сам конвертований файл (Рисунок 4.13).

Рисунок 4.13 – Фото туру збережене у базі даних

4.3 Представлення потоків даних

Безумовно важливим етапом є планування потоків даних, а саме, як компоненти системи взаємодіють одна з одною у певних сценаріях. Це необхідно для попереднього розуміння, як система буде себе поводити у тому чи іншому сценарію взаємодії з даними і запитами від клієнта.

Вхід користувача у систему, як у ролі гіда чи туриста є дуже важливим бо саме ця точка програми є відкритою і веде до персональних даних всіх користувачі, тому послідовність дій має бути спланована заздалегідь, щоб уникнути несанкціонованого доступу до даних (Рисунок 4.14).

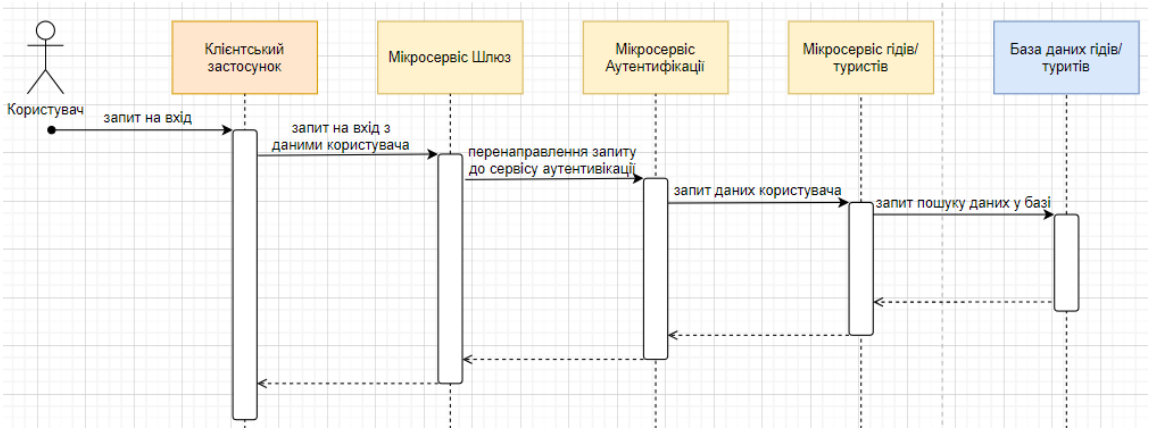


Рисунок 4.14 – Процес входу в систему користувачем

Для створення нового туру користувачеві – організатору необхідно зареєструватися в системі. Для цього процесу була створена спеціальна діаграма послідовностей дій системи (Рисунок 4.15).

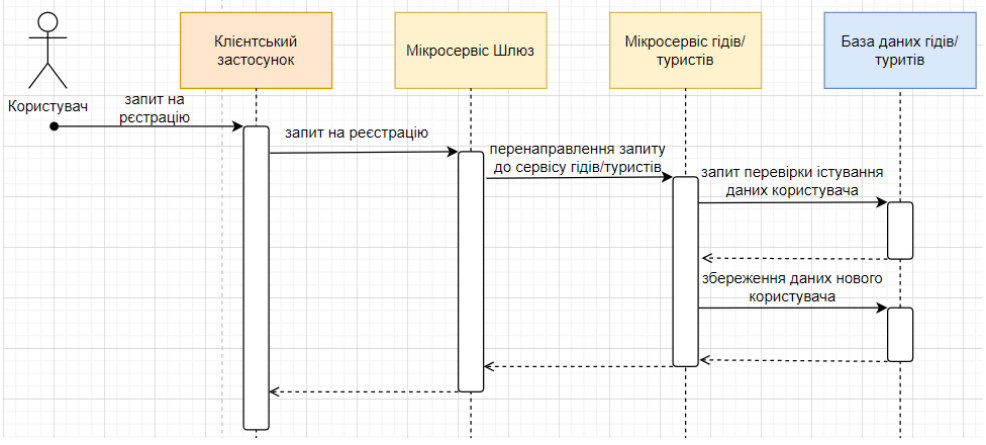


Рисунок 4.15 – Процес реєстрації користувачів

Після реєстрації акаунту бізнес користувача, він може створювати, редагувати і видаляти пропозиції активного відпочинку. Для складного процесу створення нового туру була створена діаграма послідовностей (Рисунок 4.16).

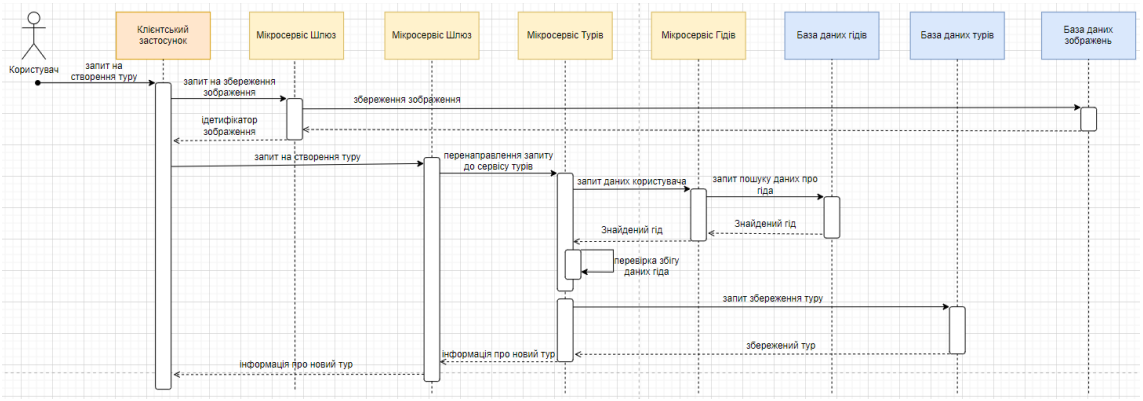


Рисунок 4.16 – Процес створення нового туру

Процес редагування пропозицій є складнішим за процес створення, тому було створено діаграму послідовностей для опису даного процесу (Рисунок 4.17).

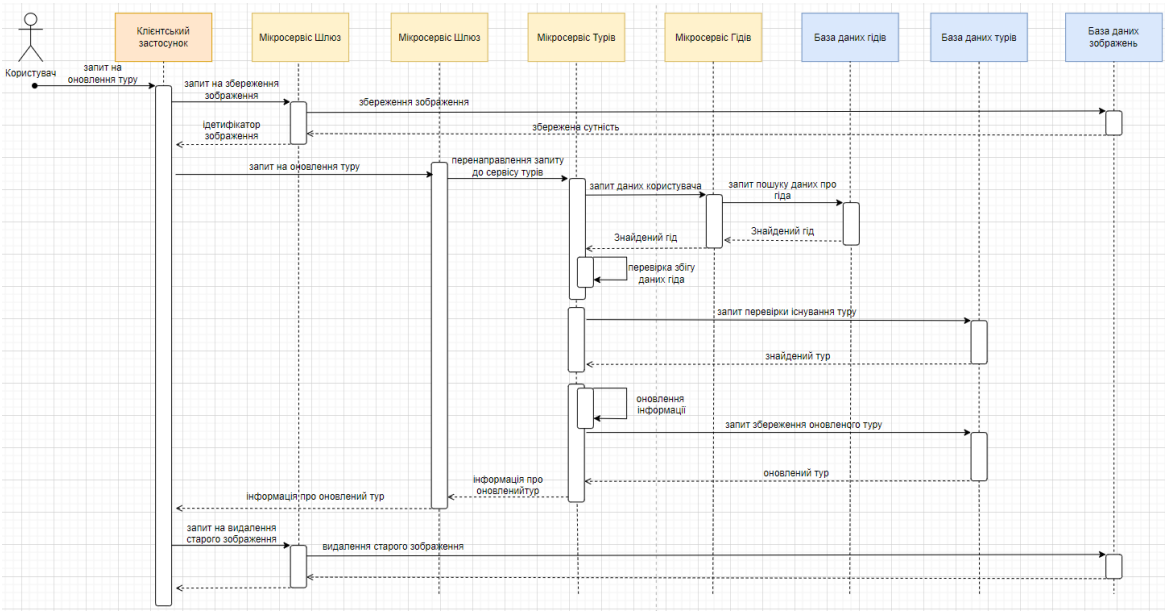


Рисунок 4.17 – Процес редагування пропозиції

Видалення інформації є важливим етапом у будь-якій системі бо при неправильному підчищенні даних можна згенерувати багато зайвих проблем, таких як: перенаповнення сховища або порушення цілісності даних, тому

було створено діаграму послідовностей для опису процесу видалення пропозицій (Рисунок 4.18).

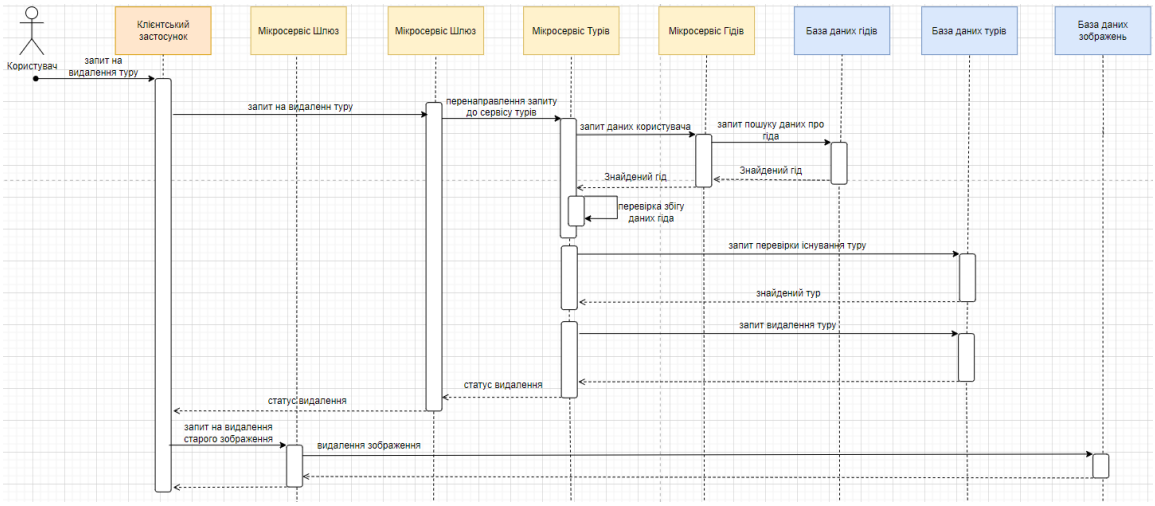


Рисунок 4.18 – Процес видалення пропозиції

4.4 Діаграми класів мікросервісів

Для розуміння взаємодії логічних елементів у коді, створюємо діаграми класів, які допомагають орієнтуватися у елементах мікросервісу, а також, маючи даний опис класів, можна легко відтворити систему, реалізувавши їх.

Діаграма класів для сервісу зображень, складається з таких елементів, як контролер, сервіс, репозиторії і модель зображення (Рисунок 4.19).

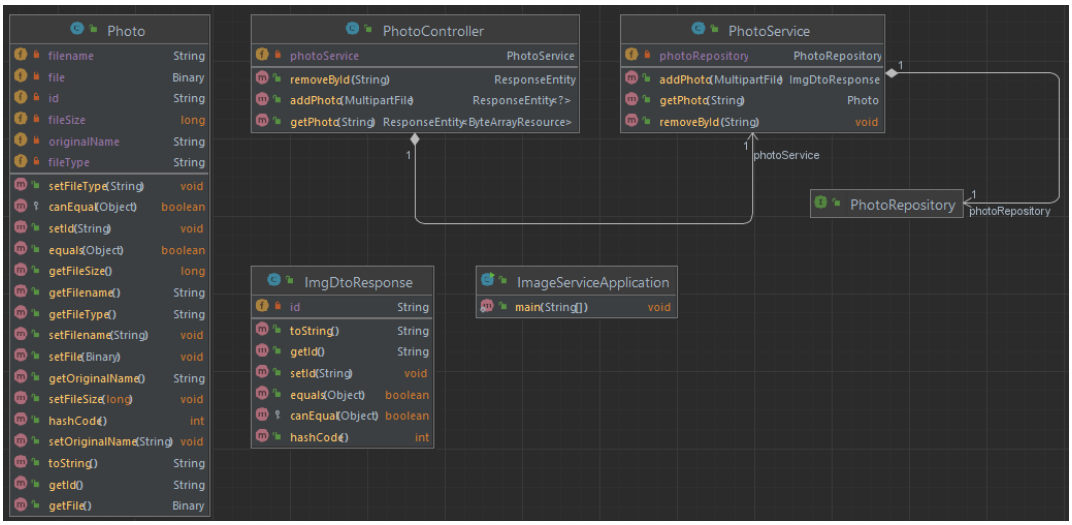


Рисунок 4.19 – Діаграма класів сервісу зображень

Існують також такі мікросервіси, які мають дуже мало класів, але багато функціональностей, які доступні безпосередньо «з коробки». Зазвичай ці сервіси конфігуруються, файлами налаштувань і анотаціями, цим самим, не потребують імплементації логіки. Таким чином, діаграми класів для сервісів конфігурації, шлюзу і реєстру є дуже простими (Рисунок 4.20).

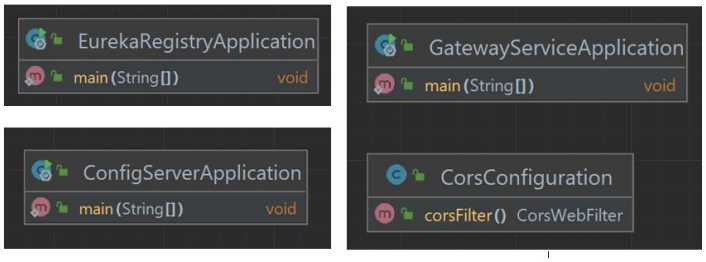


Рисунок 4.20 – Діаграма класів сервісів конфігурації, шлюзу і реєстру

Сервіс аутентифікації є складнішим за сервіс зображень бо містить в собі набагато більше логічних елементів (Рисунок 4.21)

Відповідальність даного сервісу полягає у обробленні запитів на вхід у систему і видачу спеціального токену доступу, за допомогою якого користувач здійсню запити до інший кінцевих точок системи.

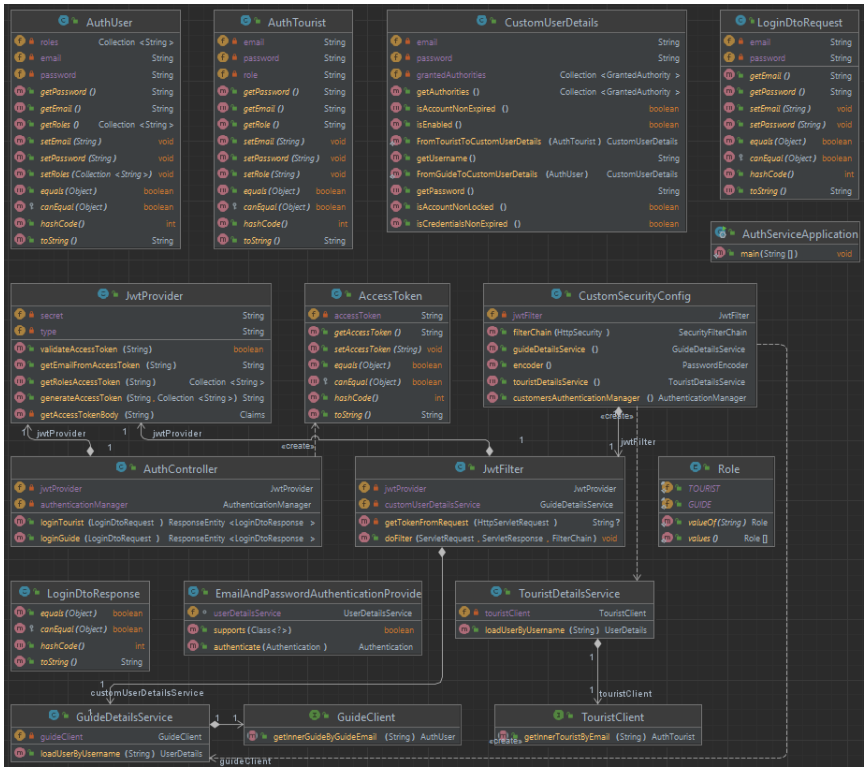


Рисунок 4.21 – Діаграма класів сервісу аутентифікації

Мікросервіс турів має велике значення для всієї системи бо з ним комунікують майже все інші сервіси, тому діаграма класів для нього є досить розгалуженою і насиченою (Рисунок 4.22).

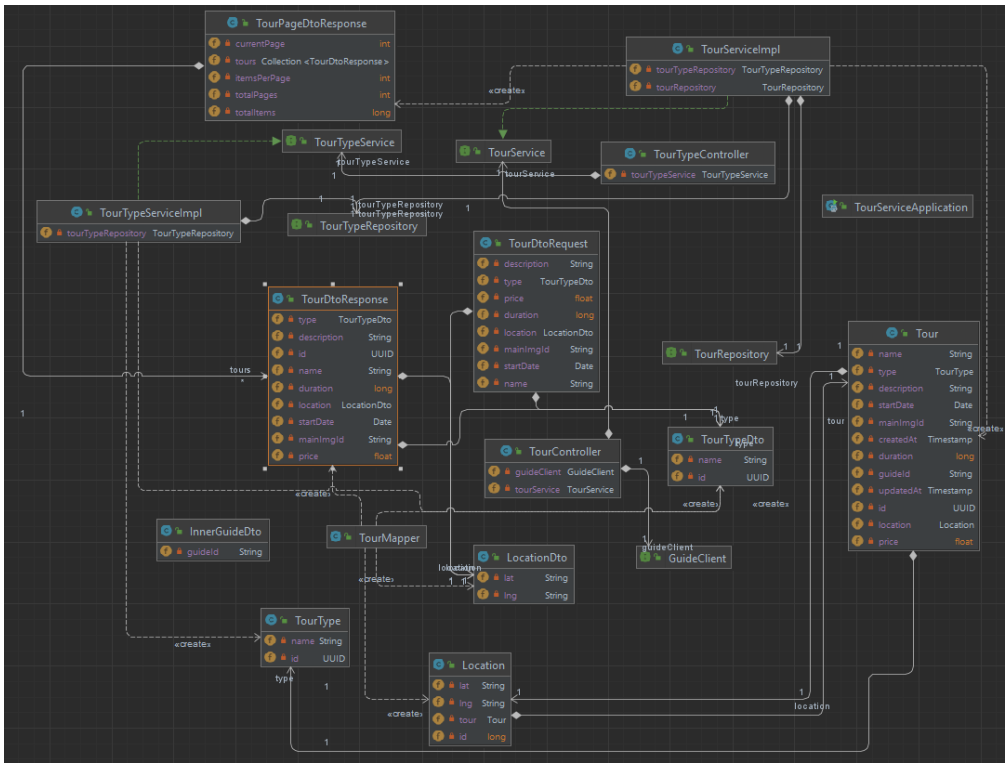


Рисунок 4.22 – Діаграма класів сервісу турів

4.5 Варіанти використання застосунку

Існує безліч варіантів, як можна розробити систему і які функції вона буде надавати, яким чином і у якій послідовності. Для визначення взаємодії користувачів з клієнтським інтерфейсом необхідно створити діаграму варіантів користування системою (Рисунок 4.23).

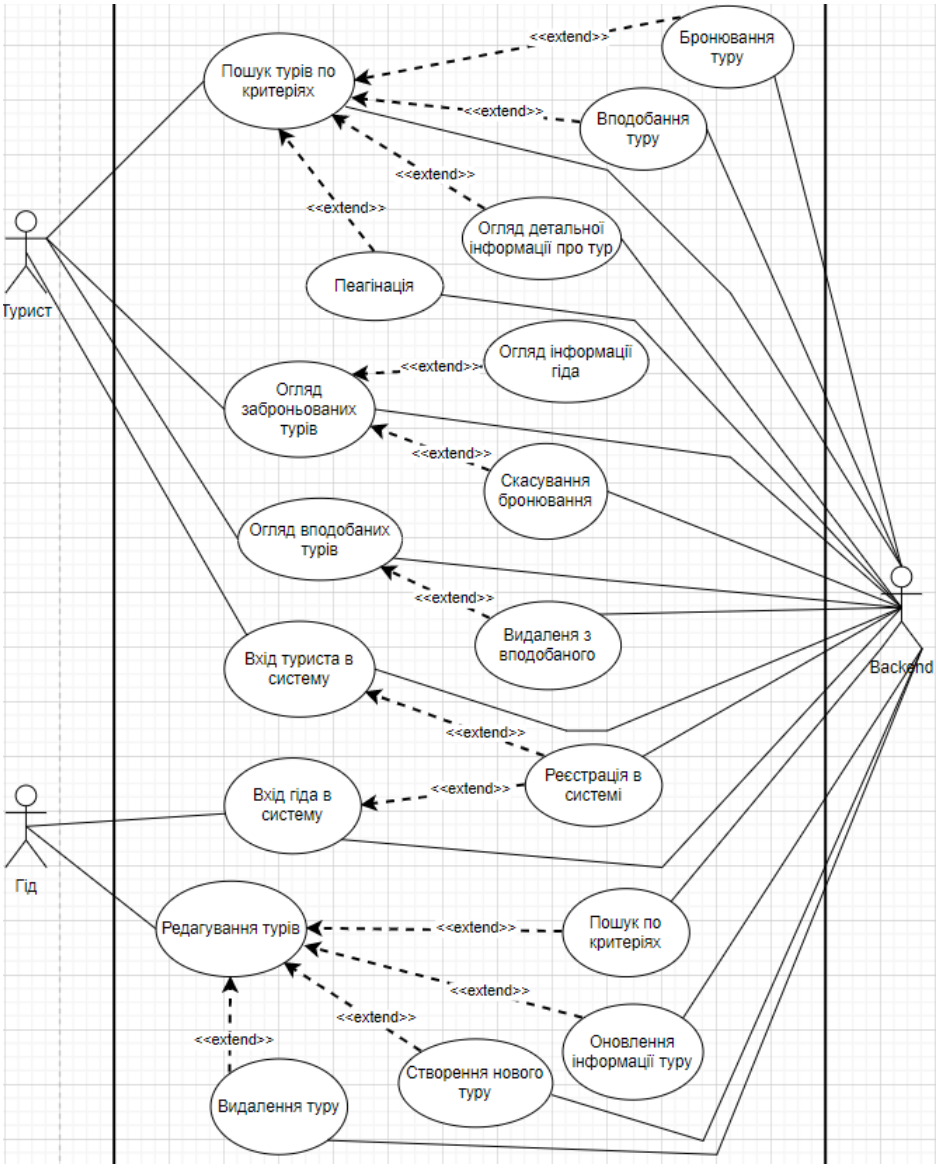
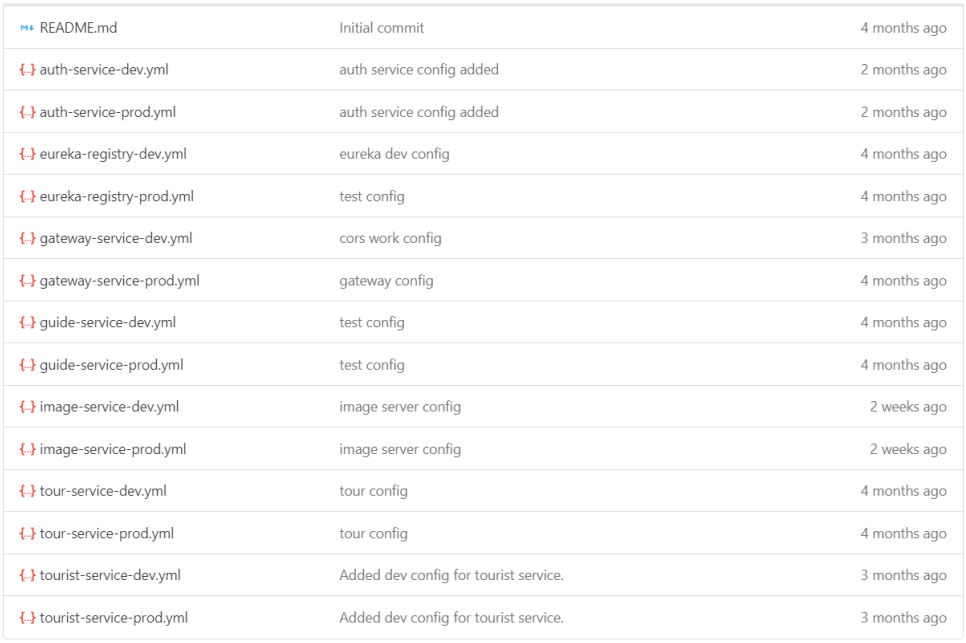


Рисунок 4.23 – Діаграма варіантів використання основних функцій

5 РЕАЛІЗАЦІЯ ПРОГРАМНИХ МОДУЛІВ

5.1 Налаштування мікросервісів

Кожен сервіс використовує спеціальні файли конфігурацій, для налаштування, при його запуску, особливо, коли існує декілька середовищ де програма може бути запущена. Для централізованого зберігання всіх файлів профілів запуску, різних мікросервісів, було створено спеціальний GitLab репозиторій де вони зберігаються (Рисунок 5.1).



README.md	Initial commit	4 months ago
auth-service-dev.yml	auth service config added	2 months ago
auth-service-prod.yml	auth service config added	2 months ago
eureka-registry-dev.yml	eureka dev config	4 months ago
eureka-registry-prod.yml	test config	4 months ago
gateway-service-dev.yml	cors work config	3 months ago
gateway-service-prod.yml	gateway config	4 months ago
guide-service-dev.yml	test config	4 months ago
guide-service-prod.yml	test config	4 months ago
image-service-dev.yml	image server config	2 weeks ago
image-service-prod.yml	image server config	2 weeks ago
tour-service-dev.yml	tour config	4 months ago
tour-service-prod.yml	tour config	4 months ago
tourist-service-dev.yml	Added dev config for tourist service.	3 months ago
tourist-service-prod.yml	Added dev config for tourist service.	3 months ago

Рисунок 5.1 – Репозиторій конфігурацій

Файли налаштувань містять різні конфігурації, відповідно до потреб і призначення того, чи іншого мікросервісу. Наприклад, yml файл профілю dev для Gateway містить налаштування порту запуску програми, доступу до реєстру сервісів, а також CORS (Рисунок 5.2).

gateway-service-dev.yml 914 bytes Open in

```yaml
server:
  port: 8011

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/

spring:
  cloud:
    gateway:
      default-filters:
        # Removes duplicates of headers.
        - DedupeResponseHeader=Access-Control-Allow-Origin Access-Control-Allow-Credentials, RETAIN_UNIQUE
      discovery:
        locator:
          enabled: true # enable eureka registry to provide services info
          lower-case-service-id: true # enable the lower case values in the URL.
      globalcors:
        add-to-simple-url-handler-mapping: true
        cors-configurations:
          '[/**]':
            allowedOrigins: "http://localhost:4200"
            allowedMethods:
              - OPTIONS
              - GET
              - PUT
              - POST
              - DELETE
```

Рисунок 5.2 – Файл налаштувань профілю dev для сервісу Gateway

Також, всі мікросервіси мають файл налаштувань за замовчанням, який використовується для встановлення критичних параметрів доступу до централізованого репозиторію (Рисунок 5.3). Після запуску програми, здійснюється підключення до сервісу налаштувань і мікросервіс отримує необхідний для певного профілю набір параметрів.

application.yml

```yaml
spring:
  application:
    name: gateway-service
  config:
    import: "optional:configserver:http://localhost:8888"
```

Рисунок 5.3 – Файл налаштувань за замовчанням для сервісу Gateway

Для запуску мікросервісу з певним профілем необхідно вказати його, як змінну при запуску програми (Рисунок 5.4).

Configuration Code Coverage Logs

Main class: com.cloud.guide_service.GuideServiceApplication

VM options: -Dspring.profiles.active=dev

Рисунок 5.4 – Налаштування змінної профіля перед запуском

5.2 Автоматична збірка модулів

Для збирання проектів було використано систему автоматичної збірки Gradle. Вона побудована на базі даних Apache Ant та Apache Maven, але створює DSL на базі Groovy та Kotlin замість XML-подібної форми представлення проекту [15].

Сервіси обробки даних мають схожі залежності, саме тому, містить дуже багато однакових залежностей (Рисунок 5.5).

```
dependencies {
    implementation 'org.springframework.cloud:spring-cloud-starter-config'
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'
    implementation 'org.springframework.cloud:spring-cloud-starter-openfeign'

    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
    implementation 'org.springframework.boot:spring-boot-starter-actuator'

    compileOnly "org.projectlombok:lombok"
    annotationProcessor "org.projectlombok:lombok"

    runtimeOnly 'mysql:mysql-connector-java'

    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

Рисунок 5.5 – Файл автоматичної збірки Gradle для сервісів даних

Кожна залежність є критично-необхідною і під своєю назвою містить велику частину вже готового функціоналу (Таблиця 5.1).

Таблиця 5.1 – Опис загальних бібліотек мікросервісів даних

Назва бібліотеки	Функціональність
spring-cloud-starter-config	Функціональність для клієнт-серверної комунікації Spring для зберігання та обслуговування розподілених конфігурацій у кількох програмах і середовищах.
spring-cloud-starter-netflix-eureka-client	Функціональність для включити клієнта репозиторія Eureka у проект.
spring-cloud-starter-	Функціональність для легкого включення між-

openfeign	мікросервісної комунікації.
spring-boot-starter-web	Функціональність для підключення обробки веб запитів, яка включає HTTP-сервер і контейнер Servlet, який має можливість обслуговувати статичний і динамічний вміст.
spring-boot-starter-data-jpa	Функціональність для підключення застосунку до реляційної бази даних.
spring-boot-starter-validation	Функціональність для перевірки коректності даних об’єкта, яка може бути легко налаштована за допомогою анотацій.
spring-boot-starter-actuator	Функціональність, яка використовує кінцеві точки HTTP для надання операційної інформації про будь-яку запущену програму. Основна перевага використання цієї бібліотеки полягає в тому, що ми отримуємо показники працездатності та моніторингу готових до виробництва програм.

У даному проекті, також були використані допоміжні бібліотеки і драйвери. Наприклад, бібліотека Lombok, для генерації коду на етапі компіляції. За рахунок анотацій, проводиться аналіз вихідного коду і створюються необхідні методи. Такий підхід зменшує кількість шаблонного коду, покращує читабельність і допомагаю сконцентрувати увагу на головних елементах системи. Найпопулярніші анотації, які були використані у наявній системі – це Data, NoArgConstructor AllArgConstrictor.

5.3 Моніторинг і статус

Після створення нового сервісу, його налаштувань і запуску він повинен буди відображений у реєстрі сервісів (Рисунок 5.6), це свідчить про

те, що сервіс був успішно запущений і зконфігурований. Статус програми може змінитися, якщо відбулись якісь проблеми і мікросервіс, наприклад, перестав відповідати.

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
AUTH-SERVICE	n/a (1)	(1)	UP (1) - localhost:auth-service:8055
CONFIG-SERVER	n/a (1)	(1)	UP (1) - localhost:config-server:8888
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - localhost:gateway-service:8011
GUIDE-SERVICE	n/a (1)	(1)	UP (1) - localhost:guide-service:8020
IMAGE-SERVICE	n/a (1)	(1)	UP (1) - localhost:image-service:8040
TOUR-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-SVIK3I8.fritz.box:tour-service:8021
TOURIST-SERVICE	n/a (1)	(1)	UP (1) - localhost:tourist-service:8023

Рисунок 5.6 – Реєстр мікросервісів

За потреби отримувати більш детальну інформацію, а також кастомні метрики для бізнесу, можна інтегрувати такі системи як Prometheus або InfluxDB, які дозволяють здійснювати гнучкий моніторинг внутрішніх процесів системи.

5.4 Призначення мікросервісів

Кожен компонент системи відповідальний за певні унікальні функції. Саме такий підхід, мається на увазі, при побудові мікросервісної архітектури. Кожен несе відповідальність за певну функціональність. Таким чином, зазвичай при розробці системи на мікросервісній архітектурі, створюється велика кількість незалежних програм, які можуть комунікувати за рахунок відкритих інтерфейсів.

Дана система містить різні сервіси з різними цілями і функціями (Таблиця 5.2), які повинні якісно виконувати тільки ті операції, які знаходяться у колі їх відповідальності і не перебирати чужі функції. Цим підходом уникається повторне використання коду. Дуплікація логіки є однією з найгірших практик у програмуванні.

Таблиця 5.2 – Функціональні відповідальності мікросервісів

Назва мікросервісу	Функціональна відповідальність
Config-server	Надає конфігурації іншим мікросервісам з репозиторію конфігурацій.
Eureka-registry	Реєструє і моніторить стан всіх інших сервісів. Також, має важливу роль у перенаправленні запитів, так як може надавати інформацію про розташування мікросервісу за його ім’ям.
Gateway-service	Приймає запити від клієнтських застосунків і передає її до певного мікросервісу базуючись на URL.
Guide-service	Оброблює всю інформацію про бізнес користувачів, а саме їх персональні дані – їх збереження, оновлення і видалення.
Tourist-service	Оброблює всю інформацію про туристів, а саме їх персональні дані – їх збереження, оновлення і видалення.
Tour-service	Оброблює всю інформацію про пропозиції від гідів, а саме ті дані, які завантажуються бізнес користувачем про активність, крім зображень.
Comment-service	Оброблює всю інформацію про коментарі до турів.
Image-service	Відповідальний за збереження і видалення зображень для турів.
Auth-service	Відповідальний за вхід в систему, а саме, за видачу токенів доступу.

5.5 Внутрішні структури

Як уже було описано раніше, у даній системі використовується підхід об’єктно-реляційного і об’єктно-документного відображення. Такий підхід дозволяє описати структуру бази даних, безпосередньо у коді програми і згенерувати всі необхідні таблиці в базі при завантаженні застосунку. Таким чином, наприклад, описується модель туру, як клас у систему, додаються спеціальні анотації для взаємодії з базою і при компіляції відбувається генерація таблиці і необхідних параметрів цієї ж таблиці турів(Рисунок 5.7).

```java
@Data
@Entity
@Table(name = "tours")
public class Tour implements Serializable {

    @Id
    @GeneratedValue(generator = "uuid2")
    @GenericGenerator(name = "uuid2", strategy = "uuid2")
    @Column(name = "tour_id", updatable = false, nullable = false, columnDefinition = "VARCHAR(36)")
    @Type(type = "uuid-char")
    private UUID id;

    @Column(name = "name")
    private String name;

    @Column(name = "description")
    private String description;

    @Column(name = "start_date")
    private Date startDate;

    @Column(name = "price")
    private float price;

    @Column(name = "duration")
    private long duration;

    @Column(name = "main_img_id")
    private String mainImgId;
```

Рисунок 5.7 – Клас відображення моделі туру без полів зв’язків

Існують різні зв'язки і обмеження у базі даних між різними таблицями (Таблиця 5.3), тому правильне створення цих відносин є дуже необхідним, в іншому випадку, дані можуть не зберігати цілісність при зберігання, оновленні чи видаленні або навіть пошуку. Для побудови зв’язків на рівні моделі об’єкту в коді програми використовуються спеціальні анотації, тфкі як OneToOne, OneToMany, ManyToMany і ManyToOne (Рисунок 5.8).

```
@Column(name = "guide_id", nullable = false, updatable = false, columnDefinition = "VARCHAR(36)")
private String guideId;

@OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy = "tour")
@JoinColumn(name = "loc_id", referencedColumnName = "loc_id")
private Location location;

@ManyToOne(cascade = CascadeType.MERGE, fetch = FetchType.LAZY)
@JoinColumn(name = "type_id", referencedColumnName = "type_id")
private TourType type;

@CreationTimestamp
@Column(name = "created_at", updatable = false, nullable = false)
private Timestamp createdAt;

@UpdateTimestamp
@Column(name = "updated_at")
private Timestamp updatedAt;
```

Рисунок 5.8 – Клас відображення моделі туру з полями зв’язків

Таблиця 5.3 – Опис анотацій об’єктно-реляційного відображення моделі

Анотація	Призначення
@Entity	Модель маєвикористовуватись, як сутність у базі дани.
@Table	Повиння бути створена таблиця зпевним ім’ям, якщо заданий параметр name. По замовчанню береться ім’я класу моделі.
@Id	Обов’язкова анотація при створенні об’єктно-реляційного/документного відображення. Позначає, що поле використовується, як ідентифікатор.
@GeneratedValue	Вказує на те, яким чином буде генеруватися значення.
@Column	Позначає, що поле це колонка, хоча всі поля автоматично стануть стовпчиками у таблиці. Параметр name відповідає за ім’я колонки у базі, unique за унікальність значення,

	nullable за можливість комірки приймати значення null.
@OneToOne,@OneToMany, @ManyToMany, @ManyToOne	Анотації, які описують відносини сутностей в сховищі даних.
@CreationTimestamp, @UpdateTimestamp	Спеціальні анотації для відстреження часу створення і оновлення об’єквів у сховищі відповідно.

Використання анотацій у застосунках на базі платформи Spring є гарною практикою для створення різних типів зв’язків і об’єктів програми. Так як використаний фреймворк реалізує патерн проектування – інверсії контроля, а саме впровадження залежностей то можна залишатися дуже гнучким у створенні об’єктів в глобальному контейнері, а потім використовувати їх у різним місцях програми. Якщо позначити клас анотацією @Service, @Repository, @RestConrtoller або @Component то Spring буде намагатися створити об’єкт такого класу і зареєструвати його в своєму контейнері DI (Рисунок 5.9).

```
@Service
public class TourServiceImpl implements TourService {
```

Рисунок 5.9 – Використання анотації @Service

Інколи необхідно створити об’єкт вручну бо система, просто, не знає звідки створити певний елемент, в такому випадку використовується анотація @Bean і зазвичай, реалізується певний метод, який повертає бажаний об’єкт (Рисунок 5.10).

```
@Bean
public GuideDetailsService guideDetailsService() { return new GuideDetailsService(); }

@Bean
public TouristDetailsService touristDetailsService() { return new TouristDetailsService(); }

@Bean
public AuthenticationManager customersAuthenticationManager() {
    return authentication -> {
        var roles : Collection<? extends GrantedAuthority> = authentication.getAuthorities();
        if (roles.stream().anyMatch(r -> r.toString().equals(Role.TOURIST.name()))) {
            var user : UserDetails = touristDetailsService().loadUserByUsername(authentication.ge
```

Рисунок 5.10 – Опис створення об’єктів з анотацією @Bean

Після створення необхідних об’єктів в контейнері впровадження залежностей, їх можна отримати трьома шляхами, а саме через поле, сеттер або конструктор у класі, ці елементи мають бути помічені анотацією @Autowired (Рисунок 5.11). Зазвичай використовують конструктор або метод сеттер для отримання об’єкту. Також гарною практикою є використання одного підходу у всій програмі за винятком деяких ситуацій.

```
@Service
public class TourServiceImpl implements TourService {

    private TourRepository tourRepository;
    private TourTypeRepository tourTypeRepository;

    @Autowired
    public TourServiceImpl(TourRepository tourRepositor
        this.tourRepository = tourRepository;
        this.tourTypeRepository = tourTypeRepository;
    }
```

Рисунок 5.11 – Впровадження об’єктів репозиторіїв у клас сервіс

Завдяки реалізації підходу ORM і ODM можна створити спеціальні інтерфейси, які матимуть доступ до комунікації зі сховищем даних (Рисунок 5.12), такий підхід значно спрощує взаємодію з базою і зменшує кількість непотрібного коду за рахунок того, що інтерфейс який наслідується надає вже багато існуючих стандартних запитів. За необхідності

в дочірньому інтерфейсі можна створити кастомні запити, слідуючи певним правилам в іменуванні або написати запит вручну використовуючи анотацію @Query.

```
@Repository
public interface TourTypeRepository extends JpaRepository<TourType, UUID> {
    Optional<TourType> findTourTypeByName(String name);
}
```

Рисунок 5.12 – Реалізація репозиторію типів турів

Спеціальні класи сервіси відповідають за бізнес логіку. В них відбувається обробка даних, отриманих за рахунок запитів до контролерів клієнтом. Наприклад, мікросервіс турів, при отриманні запиту на збереження нового туру, отримує інформацію через API, тобто контролер отримує запит, відбувається перевірка вхідних даних і потім інформація передається до класу TourServiceImpl. В якому, в свою чергу, реалізований метод зберігання нового туру (Рисунок 5.13), але перед збереженням, виконується етап перевірки на рівні сховища а також етап конвертації даних пересилання, так званих, DTO об’єктів, у модель даних туру. Якщо всі етапи проходять успішно то новий тур зберігається у сховищі.

```
@Override
@Transactional
public TourDtoResponse create(TourDtoRequest tourDtoRequest, String guideId) {
    TourType type = checkTourTypeExistence(tourDtoRequest.getType());

    Tour tour = new Tour();
    tour.setType(type);
    tour.setGuideId(guideId);

    tour = tourRepository.save(TourMapper.tourDtoRequestToTour(tourDtoRequest, tour));
    return TourMapper.tourToTourDtoResponse(tour);
}
```

Рисунок 5.13 – Реалізація методу зберігання туру

Data transfer objects – це патерн який використовується у даній системі для обмеження доступу і зменшення навантаження. Наприклад, при запиті інформації про користувача, приватна інформація, така, як пароль не надсилається, що дозволяє уникнути розкриття персональної інформації.

Майже для кожного типу запиту існує свій DTO клас, який описує структуру об’єкту (Рисунок 5.14).

```
@Data
public class GuideDtoResponse {
    private UUID id;
    private String username;
    private String email;
    private String phone;
    private String firstName;
    private String lastName;
    private Collection<Role> roles;
}
```

Рисунок 5.14 – Клас для опису відповіді при запиті інформації про гіда

Класи контролери є саме тими елементами, які дозволяють налаштувати інтерфейс системи і оброблювати запити, що надходять від програм клієнтів. Більша частина конфігурацій API базується на налаштуваннях за допомогою анотацій (Таблиця 5.4).

Таблиця 5.4 – Опис анотацій класів контролерів

Анотація	Опис
@RestController	Позначає, що клас є REST контролером.
@RequestMapping("/шлях")	Позначає, що запити, які мають таку складову URL повинні направлятися до зазначеного класу.
@GetMapping, @PostMapping, @PutMapping, @DeleteMapping	Позначає, який вид HTTP запитів може оброблювати метод класу, а також вказується шлях до нього.
@RequestBody	Позначає тіло запиту, яке надходить від клієнтського застосунка.

@Valid	Ця анотація, дозволяє перевірити тіло запику за критеріями, якщо вони налаштовані.

5.6 Зовнішні інтерфейси

Важливу роль у системі відіграють класи контролери, за допомогою яких створюється веб інтерфейс доступу до застосунку. В програмі реалізовані різні контролери для різних моделей. Кожен клас має в собі спеціальні методи для створення, оновлення, отримання і видалення інформації. Відповідно, кожен метод контролеру це end point – URL по якому доступна певна функціональність. Функція отримує конфігурації за допомогою анотацій, в яких налаштовується шлях до кінцевої точки і http метод (Рисунок 5.15).

```
@PutMapping("/{tourId}")
public ResponseEntity<TourDtoResponse> updateTourById(@Validated @RequestBo
    return new ResponseEntity<>(tourService.update(tourDtoRequest, tourId),
}

@DeleteMapping("/{tourId}")
public ResponseEntity deleteTourById(@PathVariable UUID tourId) {
    tourService.delete(tourId);
    return new ResponseEntity(HttpStatus.OK);
}

/**
 *
 * @param guideId - ID of a guide.
 * @param pageNo - Page number.
 * @param itemsNumber - Amount of items per page.
 * @return List of tours created by a Guide.
 */
@GetMapping("/guide/{guideId}")
public ResponseEntity<Collection<TourDtoResponse>> getToursByGuideId(
        @PathVariable String guideId,
        @RequestParam(defaultValue = "0") int pageNo,
        @RequestParam(defaultValue = "10") int itemsNumber) {
```

Рисунок 5.15 – Деякі методи класу контролера бізнес користувачів

Створивши і налаштувавши контролери програма отримує певний API. Знаючи інформацію налаштувань інтерфейсу було створено спеціальну колекцію за допомогою програми Postman, яка дозволяє надсилати http

запити (Рисунок 5.16). Даний застосунок дуже зручно використовувати для ручного тестування при розробці продукту.

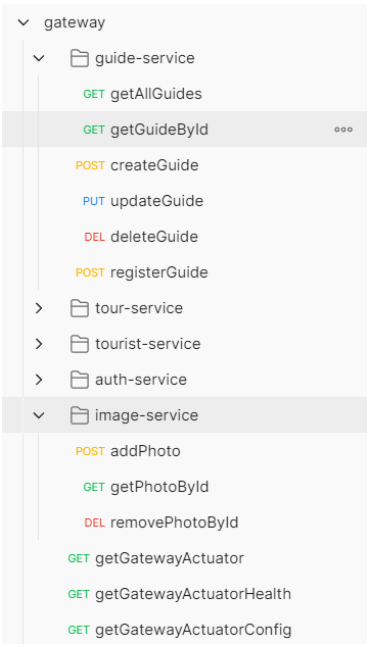


Рисунок 5.16 – Інтерфейси мікросервісів у програмі Postman

5.7 Аутентифікація і авторизація

Майже всі сучасні системи потребують тієї чи іншої системи захисту і контролю доступу до ресурсів. Даний застосунок не є виключенням. У даній системі було реалізовано два різних типи користувачів, а саме туристи і гіди – бізнес користувачі. Головною відмінність цих двох ролей є доступ до пропозицій – їх створення, редагування і видалення. Ці два типи акаунтів повністю незалежні, що означає, якщо турист захоче запропонувати екскурсію то йому, буде необхідно увійти або створити новий бізнес акаунт.

На даний момент, у системі використовується підхід авторизації з JWT. Такий варіант доступу до ресурсів, дозволяє користуватися одним токеном і отримувати доступ до різних частин різних мікросервісів, що значно спрошує розробку застосунку. Для налаштування аутентифікації бу використаний спеціальний фреймворк Spring Security. Завдяки чому конфігурування доступу зводиться до реалізації декількох методів не беручи до уваги авторизацію за допомогою JWT (Рисунок 5.17).

```
@Autowired
public CustomSecurityConfig(JwtFilter jwtFilter) { this.jwtFilter = jwtFilter; }

@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.cors() CorsConfigurer<HttpSecurity>
            .and().httpBasic().disable() HttpSecurity
            .csrf().disable()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and() HttpSecurity
            .authorizeRequests() ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
            .antMatchers( ...antPatterns: "/auth/admin").hasAuthority("ADMIN")
            .antMatchers( ...antPatterns: "/auth/**").permitAll()
            .anyRequest().authenticated()
            .and() HttpSecurity
            .authenticationManager(customersAuthenticationManager())
            .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);
    return http.build();
}
```

Рисунок 5.17 – Налаштування безпеки доступу

Для доступу до ресурсів, була розроблена спеціальна структура JWT яка містить інформацію про власника даного токену (Рисунок 5.18). Токен містить інформацію про поштову скриньку і роль користувача, термін придатності токену. Дана інформаціє зберігається середині токена і є доступною для всіх хто його має. Велику роль має «сіль» - підпис, який зберігається у застосунку і за рахунок якого токен дуже складно підробити. Таким чином, базуючись ні підписі генерується певна послідовність символів, що і захищає токен від редагувань і зовнішнього впливу.

```
@Component
public class JwtProvider {

    @Value("${jwt.secret}")
    private String secret;

    @Value("${jwt.type}")
    private String type;

    public String generateAccessToken(String email, Collection<String> roles) {
        return Jwts.builder()
                .setHeaderParam( name: "typ", type)
                .claim( name: "roles", roles.toArray())
                .setSubject(email)
                .setExpiration(
                        Date.from(LocalDateTime.now()
                                .plus(Duration.of( amount: 30000, ChronoUnit.MINUTES))
                                .atZone(ZoneId.systemDefault()).toInstant())
                )
                .signWith(SignatureAlgorithm.HS512, secret)
                .compact();
    }
```

Рисунок 5.18 – Реалізація логіки створення JWT

При виконанні запиту на вхід і успішному його обробленні, користувач отримує токен доступу (Рисунок 5.19). Який, у подальшому буде використовувати для того, щоб отримати дані з системи.

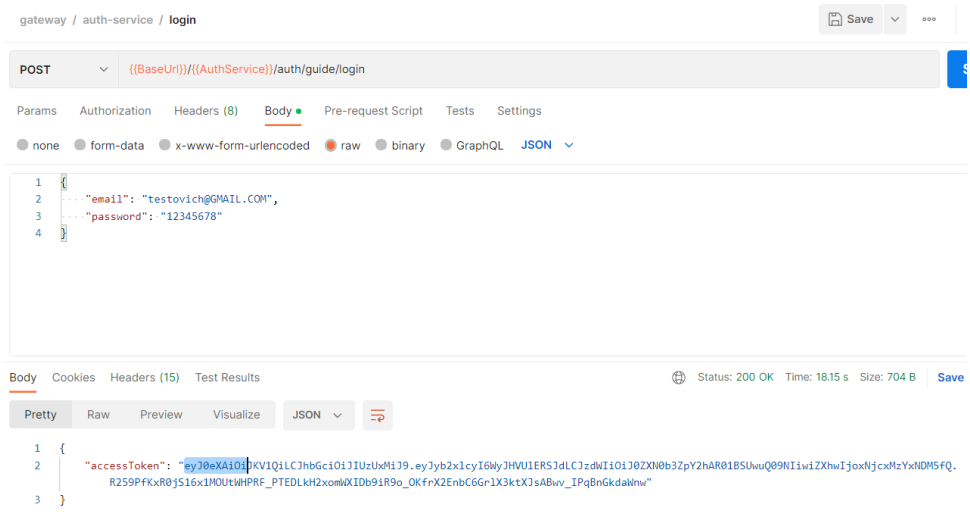


Рисунок 5.19 – Тестування запиту на вхід, через програму Postman

Не всі дані мають буди захищені. Наприклад коли з'являється новий користувач застосунку – турист, то він не мусить мати акаунт для того, щоб переглянути наявні пропозиції, але якщо він захоче забронювати тур то йому буде необхідно створити акаунт.

Створення акаунту, ще один важливий етап, користувач має обов’язково вказати свою пошту і придумати пароль, а також надати ім’я і прізвище (Рисунок 5.20), ці дані є критично важливими для подальшого користування системою.

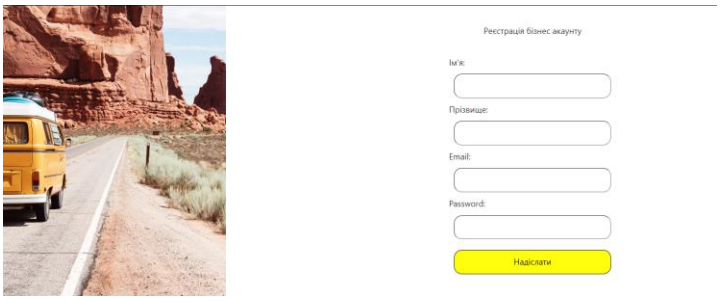


Рисунок 5.20 – форма створення бізнес акаунту

Взаємодія і реалізація всіх мікросервісів є доволі складною і містить багато різних елементі конфігурацій, класів і інтерфейсів, які в свою чергу

реалізують певні бізнес процеси і описують поведінку різних елементів (ДОДАТОК А. ЧАСТИНА ПРОГРАМНОГО КОДУ МІКРОСЕРВІСІВ).

5.8 Клієнтський застосунок

Клієнт був побудований, як односторінковий застосунок з використанням платформи Angular. Даний підхід дозволяє використовувати готові модулі і компоненти багаторазово, що пришвидшує розробку сайтів такого типу.

Застосунок складається з двох основних частин, а саме інтерфейсу туристів і бізнес користувачів. При першому потраплянні на сайт, клієнт бачить головну сторінку на якій можна шукати тури за різними параметрами (Рисунок 5.21).

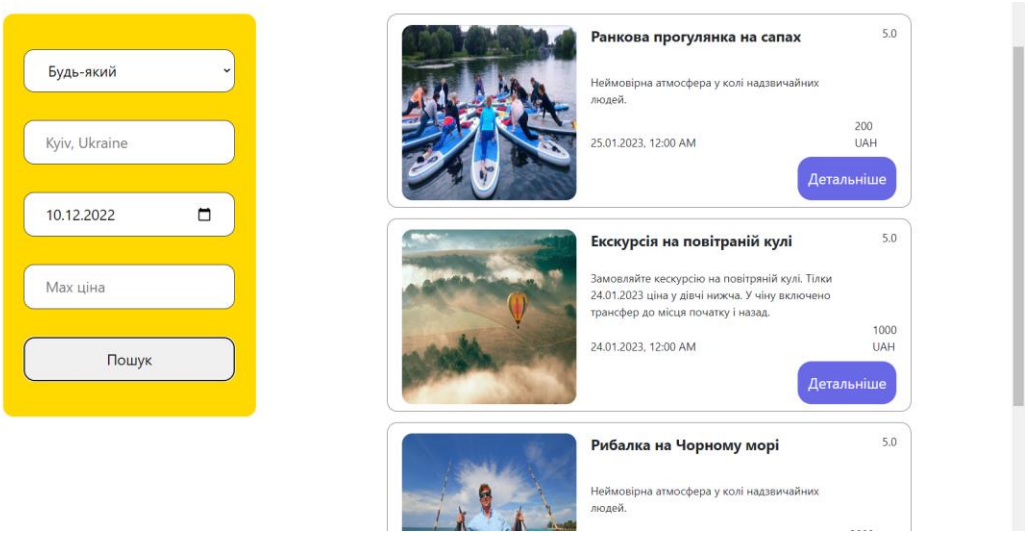


Рисунок 5.21 – Головна сторінка з формою пошуку турів

Якщо користувач зацікавлений у створенні власних пропозицій, то йому необхідно, пройти аутентифікацію у бізнес акант (Рисунок 5.22).

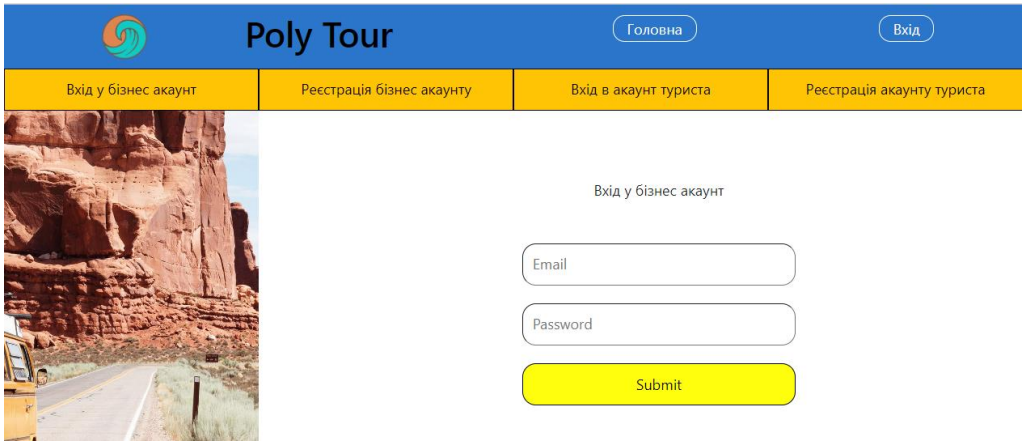


Рисунок 5.22 – Сторінка входу в бізнес акаунт

Після успішного входу в систему, користувачу буде наданий доступ до інтерфейсу гідів, де бізнес клієнт зможе створювати нові пропозиції і редагувати існуючі (Рисунок 5.23).

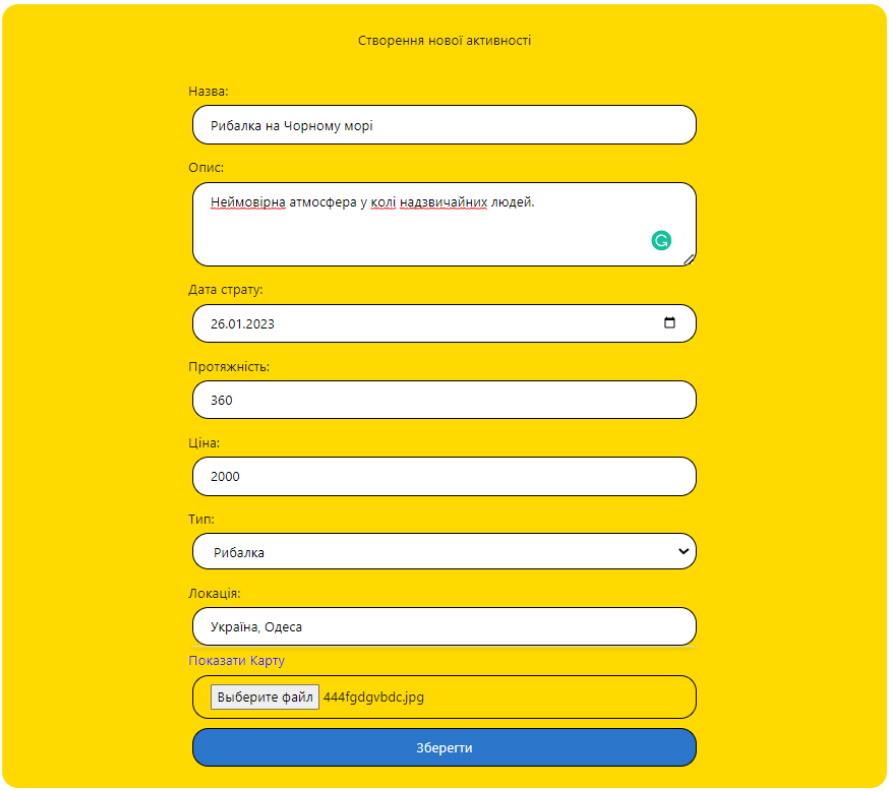


Рисунок 5.23 – Форма створення і редагування активності

Для створення нової активності гіду необхідно заповнити форму і зберегти її. Форма створення пропозиції містить такі поля:

- Назва
- Опис

- Дата старту
- Протяжність у хвилинах
- Ціна у гривнях
- Тип
- Локація
- Зображення для туру

В подальшому, дана форма може бути розширена і оптимізована відповідно до потреб клієнтів.

Веб-застосунок складається з модулів, які реалізують різні функціональності сайту (Рисунок 5.24). Розділення логіки за модулями дозволяє краще орієнтуватися в існуючому коді і завантажувати тільки необхідні функції певного модулю.

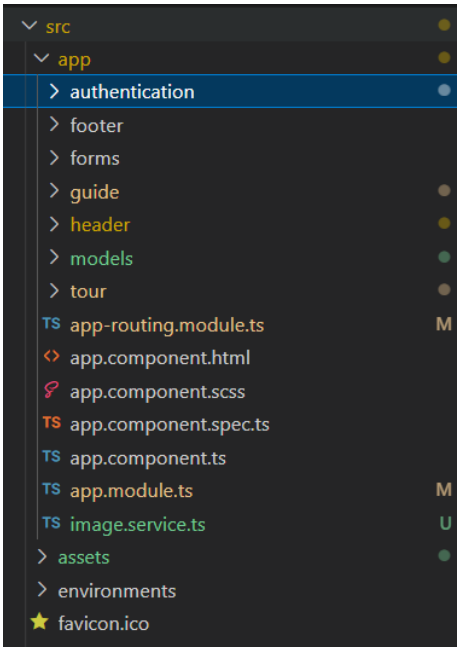


Рисунок 5.24 – Файлова структура веб-застосунку

Кожен модуль містить в собі унікальну функціональність і включає необхідні бібліотеки для подальшого використання в компонентах і сервісах, які в свою чергу реалізують всю логіку взаємодії з користувачем і поведінку застосунку в цілому, маючи певну реалізацію бізнес-логіки (ДОДАТОК Б. ЧАСТИНА ПРОГРАМНОГО КОДУ ВЕБ-КЛІЄНТА).

ВИСНОВОК

Результатом роботи є інформаційна технологія, побудована на базі мікросервісної архітекткри з використанням фреймворків сімейства Spring, допоміжних технологій і бібліотек. На базі розробленої технології спроєктовано і програмно реалізовано застосунок направлений на популяризацію і підтримку активного відпочинку, а також розвиток малого і середнього бізнесу.

У системі реалізовано різні типи користувачів, а саме туристи і організатори. Налаштовані ролі клієнтів і обмежині їх дії, відносно їхного рівня доступу.

Створена можливість пропонування активностей, бізнес користувачами і пошук пропозицій для туристів.

Розроблено дев’ять різних мікросервісів для розподілення навантаження і гнучкості системи, а також подальшого розвитку.

Реалізовані мікросервіси:

- Config-server
- Eureka-registry
- Gateway-service
- Guide-service
- Tourist-service
- Tour-service
- Comment-service
- Image-service
- Auth-service

Впроваджено функціональністі захисту приватної інформації і запобігання стороннього втручання у систему за допомогою фреймворку Spring Sequrity і підходу JWT. Імплементовано підхід об’єктно реляційного і

документного відображення, що дозволило ефективно впровадити роботу з такими базами даних, як MySQL і MongoDB.

Створено клієнтський інтерфейс за технологією SPA на базі платформи Angular, що дозволило оптимізувати склідні процеси розробки багатозадачних веб-застосунків. Також, використані технології ідеально підходить для системи даного типу.

При проектуванні системи були розроблені спеціальні діаграми, що представляють базову документацію реалізованої системи. Розроблені діаграми:

- Діаграма загальної мікросервісної архітектури системи.
- Діаграма варіантів використання.
- Діаграми класів.
- Діаграми сутностей і зв’язків.
- Діаграми послідовностей.

Для розробленої складної системи використовувались сучасні технологі, такі як:

- Spring Boot
- Spring Cloud
- Spring Security
- Spring Data JPA
- Hibernate
- Project Lombok
- MySQL
- MongoDB
- Angular
- Bootstrap

В подальшому, даний проект може бути легко розщирений іншими мікросервісами і функціональностями, які будуть задовольняти виникаючі потреби користувачів.

СПИСОК ЛІТЕРАТУРИ

1. Richardson, C. (2019). Microservices patterns: With examples in Java. Manning Publications.
2. Carnell, J., & Sánchez Illary. (2021). Spring microservices in action. Manning Publications Co.
3. Binildas, C. A. (2019). Practical microservices architectural patterns: Event-based Java microservices with Spring Boot and Spring Cloud. Apress.
4. Musib, S., & Long, J. (2022). Spring Boot in practice. Manning Publications.
5. Coury, F., Lerner, A., Murray, N., & Taborda, C. (2016). Ng-book 2: The complete guide to angular 2. Fullstack.io.
6. Wilken, J. (2018). Angular in action. Manning Publication.
7. Fain, Y., & Moiseev, A. (2019). Angular development with typescript. Manning.
8. Мікросервісна архітектура для початківців. Частина I [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one
9. Blokdyk G. Object-relational mapping Complete Self-Assessment Guide. - 5STARCooks,2022. - 309 p.
10. JSON Web Token [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://uk.wikipedia.org/wiki/JSON_Web_Token
11. Spring Cloud [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://spring.io/projects/spring-cloud
12. Gradle vs Maven Comparison [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://gradle.org/maven-vs-gradle/

13. React vs Angular: Which JS Framework to Pick for Front-end Development?. [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://radixweb.com/blog/react-vs-angular
14. Idiomatic persistence for Java and relational databases. [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://hibernate.org/orm/
15. Gradle [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https:// uk.wikipedia.org/wiki/Gradle

ДОДАТОК А. ЧАСТИНА ПРОГРАМНОГО КОДУ МІКРОСЕРВІСІВ

Guide-Service

```
@RestController
@RequestMapping("/guide")
public class GuideController {

    private GuideService guideService;

    @Autowired
    public GuideController(GuideService guideService) {
        this.guideService = guideService;
    }

    @GetMapping("/all")
    public ResponseEntity<Collection<GuideDtoResponse>> getAll() {
        return new ResponseEntity<>(guideService.getAll(), HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<GuideDtoResponse> getByGuideId(@PathVariable UUID
id) {
        return new ResponseEntity<>(guideService.getById(id), HttpStatus.OK);
    }

    @GetMapping("email/{email}")
    public ResponseEntity<GuideDtoResponse> getByEmail(@PathVariable String
email) {
        return new ResponseEntity<>(guideService.getGuideByEmail(email),
HttpStatus.OK);
    }

    @PostMapping
    public ResponseEntity<GuideDtoResponse> create(@RequestBody
GuideDtoRequest guideDtoRequest) {
        return new ResponseEntity<>(guideService.create(guideDtoRequest),
HttpStatus.CREATED);
    }

    @PutMapping("/{id}")
    public ResponseEntity<GuideDtoResponse> update(@RequestBody
GuideDtoRequest guideDtoRequest, @PathVariable UUID id) {
        return new ResponseEntity<>(guideService.update(guideDtoRequest, id),
HttpStatus.CREATED);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity delete(@PathVariable UUID id) {
        guideService.delete(id);
        return new ResponseEntity(HttpStatus.OK);
    }

    @PostMapping("/reg")
    public ResponseEntity<GuideRegDtoResponse> registerGuide(@RequestBody
GuideRegDtoRequest registerGuide) {
        return new
ResponseEntity<>(guideService.registerGuide(registerGuide),
```

```
HttpStatus.CREATED);
    }
}

@Repository
public interface GuideRepository extends JpaRepository<Guide, UUID> {
    Optional<Guide> findByEmail(String email);
}

public interface GuideService {
    GuideDtoResponse create(GuideDtoRequest guideDtoRequest);

    GuideDtoResponse update(GuideDtoRequest guideDtoRequest, UUID guideId);

    void delete(UUID guideId);

    Collection<GuideDtoResponse> getAll();

    GuideDtoResponse getById(UUID guideId);

    InnerLoginGuideDtoResponse getGuideByEmailInner(String guideEmail);
    GuideDtoResponse getGuideByEmail(String email);

    GuideRegDtoResponse registerGuide(GuideRegDtoRequest registerGuide);
}

@Service
public class GuideServiceImpl implements GuideService {

    private GuideRepository guideRepository;
    private RoleRepository roleRepository;
    private PasswordEncoder encoder;

    @Autowired
    public GuideServiceImpl(GuideRepository guideRepository, RoleRepository
roleRepository, PasswordEncoder encoder) {
        this.guideRepository = guideRepository;
        this.roleRepository = roleRepository;
        this.encoder = encoder;
    }

    @Override
    public GuideDtoResponse create(GuideDtoRequest guideDtoRequest) {
        Guide guide =
guideRepository.save(GuideMapper.guideDtoRequestToGuide(guideDtoRequest, new
Guide()));
        return GuideMapper.guideToGuideDtoResponse(guide);
    }

    @Override
    public GuideDtoResponse update(GuideDtoRequest guideDtoRequest, UUID
guideId) {
        Optional<Guide> optionalGuide = guideRepository.findById(guideId);
        if (optionalGuide.isEmpty()) {
            throw new RuntimeException("Guide not found.");
        }
        Guide guide = GuideMapper.guideDtoRequestToGuide(guideDtoRequest,
optionalGuide.get());
        return
GuideMapper.guideToGuideDtoResponse(guideRepository.save(guide));
    }
```

```
    @Override
    public void delete(UUID guideId) {
        Guide guide = guideRepository.getById(guideId);
        if (guide == null) {
            throw new RuntimeException("Guide not found.");
        }
        guideRepository.deleteById(guideId);
    }

    @Override
    public Collection<GuideDtoResponse> getAll() {
        return
GuideMapper.guidesToGuideDtosResponse(guideRepository.findAll());
    }

    @Override
    public GuideDtoResponse getById(UUID guideId) {
        return
GuideMapper.guideToGuideDtoResponse(guideRepository.getReferenceById(guideId)
);
    }

    @Override
    public InnerLoginGuideDtoResponse getGuideByEmailInner(String guideEmail)
{
        var guide =  getByEmail(guideEmail);
        var innerGuide = new InnerLoginGuideDtoResponse(
                guide.getEmail(),
                guide.getPassword(),
                guide.getRoles().stream().map(r ->
r.getRole().name()).collect(Collectors.toList())
        );
        return innerGuide;
    }

    @Override
    public GuideDtoResponse getGuideByEmail(String email) {
        return GuideMapper.guideToGuideDtoResponse(getByEmail(email));
    }

    private Guide getByEmail(String email) {
        var guideOptional = guideRepository.findByEmail(email);
        if (guideOptional.isEmpty()) {
            throw new RuntimeException("Guide not found.");
        }
        return guideOptional.get();
    }

    @Override
    public GuideRegDtoResponse registerGuide(GuideRegDtoRequest
guideRegDtoRequest) {
        var guideOptional =
guideRepository.findByEmail(guideRegDtoRequest.getEmail());
        if (guideOptional.isPresent()) {
            return new GuideRegDtoResponse(RegStatus.FAILED, "Email already
exists.");
        }

        var newGuide = new Guide();

newGuide.setPassword(encoder.encode(guideRegDtoRequest.getPassword()));
        newGuide.setEmail(guideRegDtoRequest.getEmail());
```

```
        newGuide.setFirstName(guideRegDtoRequest.getFirstName());
        newGuide.setLastName(guideRegDtoRequest.getLastName());
        newGuide.setAccountState(AccountState.NEW);

        var roleOptional = roleRepository.findRoleByRole(UserRole.GUIDE);
        Role role = roleOptional.isEmpty() ? roleRepository.save(new
Role(UserRole.GUIDE)) : roleOptional.get();
        newGuide.setRoles(Arrays.asList(role));

        guideRepository.save(newGuide);
        return new GuideRegDtoResponse(RegStatus.NEW, "Success.");
    }
}

@Data
@Entity
@Table(name = "guides")
public class Guide {

    @Id
    @GeneratedValue(generator = "uuid2")
    @GenericGenerator(name = "uuid2", strategy = "uuid2")
    @Column(name = "guide_id", updatable = false, nullable = false,
columnDefinition = "VARCHAR(36)")
    @Type(type = "uuid-char")
    private UUID id;

    @Column(name = "username", unique = true/*, nullable = false*/)
    private String username;

    @Column(name = "email", unique = true, nullable = false)
    private String email;

    @Column(name = "phone", unique = true)
    private String phone;

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

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

    @CreationTimestamp
    @Column(name = "created_at", updatable = false, nullable = false)
    private Timestamp createdAt;

    @UpdateTimestamp
    @Column(name = "updated_at")
    private Timestamp updatedAt;

    @Column(name = "password", nullable = false)
    private String password;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
            name = "user_role",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id")
    )
    private Collection<Role> roles;

    @Enumerated(value = EnumType.STRING)
```

```
    @Column(name = "account_state", nullable = false)
    private AccountState accountState;
}


@Data
public class GuideDtoRequest {

    @NotBlank
    @Size(min = 6, max = 40)
    private String username;

    @NotBlank
    @Email
    private String email;

    @NotBlank
    private String password;

    @NotBlank
    @Size(min = 7, max = 20)
    private String phone;

    @NotBlank
    @Size(min = 1, max = 40)
    private String firstName;

    @NotBlank
    @Size(min = 1, max = 40)
    private String lastName;
}

@Data
public class GuideDtoResponse {
    private UUID id;
    private String username;
    private String email;
    private String phone;
    private String firstName;
    private String lastName;
    private Collection<Role> roles;
}

@Data
public class GuideRegDtoRequest {

    @Email
    private String email;

    @NotEmpty
    @Min(8)
    private String password;

    @NotEmpty
    private String firstName;

    @NotEmpty
    private String lastName;
}

public class GuideMapper {
```

```
    public static GuideDtoResponse guideToGuideDtoResponse(Guide guide) {
        GuideDtoResponse guideDtoResponse = new GuideDtoResponse();
        guideDtoResponse.setId(guide.getId());
        guideDtoResponse.setEmail(guide.getEmail());
        guideDtoResponse.setFirstName(guide.getFirstName());
        guideDtoResponse.setLastName(guide.getLastName());
        guideDtoResponse.setPhone(guide.getPhone());
        guideDtoResponse.setUsername(guide.getUsername());
        guideDtoResponse.setRoles(guide.getRoles());
        return guideDtoResponse;
    }

    public static Collection<GuideDtoResponse>
guidesToGuideDtosResponse(Collection<Guide> guides) {
        return guides.stream().map(guide ->
guideToGuideDtoResponse(guide)).collect(Collectors.toList());
    }

    public static Guide guideDtoRequestToGuide(GuideDtoRequest
guideDtoRequest, Guide guide) {
        guide.setEmail(guideDtoRequest.getEmail());
        guide.setPhone(guideDtoRequest.getPhone());
        guide.setFirstName(guideDtoRequest.getFirstName());
        guide.setLastName(guideDtoRequest.getLastName());
        guide.setUsername(guideDtoRequest.getUsername());
        guide.setPassword(guideDtoRequest.getPassword());
        guide.setRoles(new HashSet<Role>(Arrays.asList(new
Role(UserRole.GUIDE))));
        return guide;
    }
}

@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
public class GuideServiceApplication {

   public static void main(String[] args) {
      SpringApplication.run(GuideServiceApplication.class, args);
   }
}
```

Image-Service

```
@RestController
@RequestMapping("/photo")
public class PhotoController {

    private PhotoService photoService;

    public PhotoController(PhotoService photoService) {
        this.photoService = photoService;
    }

    @PostMapping
    public ResponseEntity<?> addPhoto(@RequestParam("image") MultipartFile
image) throws IOException {
        return new ResponseEntity<>(photoService.addPhoto(image),
HttpStatus.OK);
    }
```

```
    @DeleteMapping(value = "/{photoId}")
    public ResponseEntity removeById(@PathVariable String photoId) {
        photoService.removeById(photoId);
        return new ResponseEntity(HttpStatus.OK);
    }

    @GetMapping(value = "/{photoId}", produces = {MediaType.IMAGE_JPEG_VALUE,
MediaType.IMAGE_PNG_VALUE})
    @ResponseBody
    public ResponseEntity<ByteArrayResource> getPhoto(@PathVariable String
photoId) {
        Photo loadFile = photoService.getPhoto(photoId);
        loadFile.getFile().getData();

        return ResponseEntity.ok()
                .contentType(MediaType.parseMediaType(loadFile.getFileType()
))
                .header(HttpHeaders.CONTENT_DISPOSITION, "attachment;
filename=\"" + loadFile.getFilename() + "\"")
                .body(new ByteArrayResource(loadFile.getFile().getData()));
    }
}

@Data
@Document(collection = "photos")
public class Photo {
    @Id
    private String id;

    private String filename;
    private String fileType;
    private long fileSize;
    private String originalName;
    private Binary file;
}

@Data
@NoArgsConstructor
@AllArgsConstructor
public class ImgDtoResponse {
    private String id;
}

@Repository
public interface PhotoRepository extends MongoRepository<Photo, String> { }

@Service
public class PhotoService {

    private PhotoRepository photoRepository;

    @Autowired
    public PhotoService(PhotoRepository photoRepository) {
        this.photoRepository = photoRepository;
    }

    public ImgDtoResponse addPhoto(MultipartFile file) throws IOException {
        Photo photo = new Photo();

        photo.setFilename(file.getName());
        photo.setOriginalName(file.getOriginalFilename());
```

```
        photo.setFileSize(file.getSize());
        photo.setFileType(file.getContentType());
        photo.setFile(new Binary(BsonBinarySubType.BINARY, file.getBytes()));

        photo = photoRepository.insert(photo);

        return new ImgDtoResponse(photo.getId());
    }

    public Photo getPhoto(String id) {
        var photoOptional = photoRepository.findById(id);
        if (photoOptional.isEmpty()) {
            throw new RuntimeException("Photo not found.");
        }
        return photoRepository.findById(id).get();
    }

    public void removeById(String photoId) {
        photoRepository.deleteById(photoId);
    }
}

@SpringBootApplication
@EnableEurekaClient
public class ImageServiceApplication {

   public static void main(String[] args) {
      SpringApplication.run(ImageServiceApplication.class, args);
   }

}
```

Auth-Service

```
@Configuration
@EnableWebSecurity
public class CustomSecurityConfig {

    private JwtFilter jwtFilter;

    @Autowired
    public CustomSecurityConfig(JwtFilter jwtFilter) {
        this.jwtFilter = jwtFilter;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
        http.cors()
                .and().httpBasic().disable()
                .csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests()
                .antMatchers("/auth/admin").hasAuthority("ADMIN")
                .antMatchers("/auth/**").permitAll()
                .anyRequest().authenticated()
                .and()
                .authenticationManager(customersAuthenticationManager())
                .addFilterBefore(jwtFilter,
```

```
UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }

    @Bean
    public GuideDetailsService guideDetailsService() {
        return new GuideDetailsService();
    }

    @Bean
    public TouristDetailsService touristDetailsService() {
        return new TouristDetailsService();
    }

    @Bean
    public AuthenticationManager customersAuthenticationManager() {
        return authentication -> {
            var roles = authentication.getAuthorities();
            if (roles.stream().anyMatch(r ->
r.toString().equals(Role.TOURIST.name()))) {
                var user =
touristDetailsService().loadUserByUsername(authentication.getName());
                if (user != null && encoder().matches((CharSequence)
authentication.getCredentials(), user.getPassword())) {
                    return new
UsernamePasswordAuthenticationToken(authentication.getName(),
authentication.getAuthorities());
                }
            } else if (roles.stream().anyMatch(r ->
r.toString().equals(Role.GUIDE.name()))) {
                var user =
guideDetailsService().loadUserByUsername(authentication.getName());
                if (user != null && encoder().matches((CharSequence)
authentication.getCredentials(), user.getPassword())) {
                    return new
UsernamePasswordAuthenticationToken(authentication.getName(),
authentication.getAuthorities());
                }
            }
            throw new UsernameNotFoundException(authentication.getName());
        };
    }

    @Bean
    public PasswordEncoder encoder() {
        return new BCryptPasswordEncoder();
    }
}

public class CustomUserDetails implements UserDetails {

    private String email;
    private String password;
    private Collection<? extends GrantedAuthority> grantedAuthorities;

    @Override
    public String getUsername() {
        return email;
    }

    @Override
    public String getPassword() {
```

```
        return password;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return grantedAuthorities;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }


    public static CustomUserDetails FromGuideToCustomUserDetails(AuthUser
authUser) {
        var user = new CustomUserDetails();
        user.email = authUser.getEmail();
        user.password = authUser.getPassword();
        if (authUser.getRoles() == null || authUser.getRoles().isEmpty()) {
            throw new RuntimeException("Auth user does not have any role!");
        }
        user.grantedAuthorities = authUser.getRoles()
                .stream()
                .map(r -> new SimpleGrantedAuthority(r))
                .collect(Collectors.toList());
        return user;
    }

    public static CustomUserDetails
FromTouristToCustomUserDetails(AuthTourist authUser) {
        var user = new CustomUserDetails();
        user.email = authUser.getEmail();
        user.password = authUser.getPassword();
        if (authUser.getRole() == null || authUser.getRole().isEmpty()) {
            throw new RuntimeException("Auth user does not have any role!");
        }
        user.grantedAuthorities = Arrays.asList(new
SimpleGrantedAuthority(authUser.getRole()));
        return user;
    }
}

public class GuideDetailsService implements UserDetailsService {

    @Autowired
    private GuideClient guideClient;
```

```
    @Override
    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
        var authUser = guideClient.getInnerGuideByGuideEmail(email);
        return CustomUserDetails.FromGuideToCustomUserDetails(authUser);
    }
}

public class TouristDetailsService  implements UserDetailsService {
    @Autowired
    private TouristClient touristClient;

    @Override
    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
        var authUser = touristClient.getInnerTouristByEmail(email);
        return CustomUserDetails.FromTouristToCustomUserDetails(authUser);
    }
}


      @Component
public class JwtFilter extends GenericFilterBean {

    private JwtProvider jwtProvider;
    private GuideDetailsService customUserDetailsService;

    @Autowired
    public JwtFilter(@Lazy JwtProvider jwtProvider, @Lazy GuideDetailsService
customUserDetailsService) {
        this.customUserDetailsService = customUserDetailsService;
        this.jwtProvider = jwtProvider;
    }


    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException {

        var httpServletRequest = (HttpServletRequest) request;
        var httpServletResponse = (HttpServletResponse) response;

        if (httpServletRequest.getRequestURI().contains("login")) {
            chain.doFilter(request, response);
            return;
        }

        String token = getTokenFromRequest(httpServletRequest);

      if (token != null && jwtProvider.validateAccessToken(token)) {
    var userEmail = jwtProvider.getEmailFromAccessToken(token);
    var customUserDetails =
customUserDetailsService.loadUserByUsername(userEmail);
    if (customUserDetails != null) {
        var auth = new UsernamePasswordAuthenticationToken(customUserDetails,
null, customUserDetails.getAuthorities());
        SecurityContextHolder.getContext().setAuthentication(auth);
    }
    chain.doFilter(request, response);
} else {
```

```
        httpServletResponse.setStatus(HttpStatus.UNAUTHORIZED.value());
    }

        }

        private String getTokenFromRequest(HttpServletRequest request) {
            String bearer = request.getHeader("Authorization");
            if (hasText(bearer) && bearer.startsWith("Bearer ")) {
                return bearer.substring(7);
            }
            return null;
        }
    }

    @Component
    public class JwtProvider {

        @Value("${jwt.secret}")
        private String secret;

        @Value("${jwt.type}")
        private String type;

        public String generateAccessToken(String email, Collection<String> roles)
    {
            return Jwts.builder()
                    .setHeaderParam("typ", type)
                    .claim("roles", roles.toArray())
                    .setSubject(email)
                    .setExpiration(
                            Date.from(LocalDateTime.now()
                                    .plus(Duration.of(30000, ChronoUnit.MINUTES))
                                    .atZone(ZoneId.systemDefault()).toInstant())
                    )
                    .signWith(SignatureAlgorithm.HS512, secret)
                    .compact();
        }

        public boolean validateAccessToken(String token) {
            try {
                Jwts.parser().setSigningKey(secret).parseClaimsJws(token);
                return true;
            } catch (Exception e) {
                return false;
            }
        }

        public String getEmailFromAccessToken(String token) {
            return getAccessTokenBody(token).getSubject();
        }

        public Collection<String> getRolesAccessToken(String token) {
            return getAccessTokenBody(token).get("roles", Collection.class);
        }

        private Claims getAccessTokenBody(String token) {
            return
    Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();
        }
    }
```

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class AccessToken {
    private String accessToken;
}
@RestController
@RequestMapping("/auth")
public class AuthController {

    private AuthenticationManager authenticationManager;
    private JwtProvider jwtProvider;

    public AuthController(AuthenticationManager authenticationManager,
JwtProvider jwtProvider) {
        this.authenticationManager = authenticationManager;
        this.jwtProvider = jwtProvider;
    }

    @PostMapping("/guide/login")
    public ResponseEntity<LoginDtoResponse> loginGuide(@RequestBody
LoginDtoRequest loginDtoRequest) {
        try {
            authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(
                            loginDtoRequest.getEmail(),
                            loginDtoRequest.getPassword(),
                            Arrays.asList(new
SimpleGrantedAuthority(Role.GUIDE.name()))
                    )
            );
        } catch (Exception e) {
            return new ResponseEntity("Authentication failed!",
HttpStatus.UNAUTHORIZED);
        }

        var accessToken = new AccessToken(
                jwtProvider.generateAccessToken(loginDtoRequest.getEmail(),
                        Arrays.asList(Role.GUIDE.name()))
        );

        return new ResponseEntity(accessToken, HttpStatus.OK);
    }

    @PostMapping("/tourist/login")
    public ResponseEntity<LoginDtoResponse> loginTourist(@RequestBody
LoginDtoRequest loginDtoRequest) {
        try {
            authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(
                            loginDtoRequest.getEmail(),
                            loginDtoRequest.getPassword(),
                            Arrays.asList(new
SimpleGrantedAuthority(Role.TOURIST.name()))
                    )
            );
        } catch (Exception e) {
            return new ResponseEntity("Authentication failed!",
HttpStatus.UNAUTHORIZED);
        }

        var accessToken = new AccessToken(
```

```
                jwtProvider.generateAccessToken(loginDtoRequest.getEmail(),
                        Arrays.asList(Role.TOURIST.name()))
        );

        return new ResponseEntity(accessToken, HttpStatus.OK);
    }
}
```

Gateway-Service

```
@SpringBootApplication
@EnableEurekaClient
public class GatewayServiceApplication {
   public static void main(String[] args) {
      SpringApplication.run(GatewayServiceApplication.class, args);
   }
}

@Configuration
public class CorsConfiguration {

    @Bean
    public CorsWebFilter corsFilter() {
        org.springframework.web.cors.CorsConfiguration corsConfiguration =
new org.springframework.web.cors.CorsConfiguration();
        corsConfiguration.setAllowCredentials(true);

corsConfiguration.setAllowedOrigins(Arrays.asList("http://localhost:4200"));
        corsConfiguration.setAllowedMethods(Arrays.asList("GET", "POST",
"PUT", "DELETE", "OPTIONS", "HEAD"));
        corsConfiguration.addAllowedHeader(ALL);
        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", corsConfiguration);
        return new CorsWebFilter(source);
    }

}
```

Tour-Service

```
@RestController
@RequestMapping("/tour")
public class TourController {

    private TourService tourService;
    private GuideClient guideClient;

    @Autowired
    public TourController(TourService tourService, GuideClient guideClient) {
        this.tourService = tourService;
        this.guideClient = guideClient;
    }

    @GetMapping("/{tourId}")
    public ResponseEntity<TourDtoResponse> getTourById(@PathVariable UUID
tourId) {
        return new ResponseEntity<>(tourService.getById(tourId),
HttpStatus.OK);
    }
```

```
    @PostMapping("/{guideId}")
    public ResponseEntity<TourDtoResponse> createTourByGuideId(@Validated
@RequestBody TourDtoRequest tourDtoRequest,
@PathVariable UUID guideId) {
        InnerGuideDto innerGuideDto =
guideClient.getInnerGuideByGuideId(guideId);
        return new ResponseEntity<>(tourService.create(tourDtoRequest,
innerGuideDto.getGuideId()), HttpStatus.CREATED);
    }

    @PutMapping("/{tourId}")
    public ResponseEntity<TourDtoResponse> updateTourById(@Validated
@RequestBody TourDtoRequest tourDtoRequest, @PathVariable UUID tourId) {
        return new ResponseEntity<>(tourService.update(tourDtoRequest,
tourId), HttpStatus.OK);
    }

    @DeleteMapping("/{tourId}")
    public ResponseEntity deleteTourById(@PathVariable UUID tourId) {
        tourService.delete(tourId);
        return new ResponseEntity(HttpStatus.OK);
    }

    @GetMapping("/guide/{guideId}")
    public ResponseEntity<Collection<TourDtoResponse>> getToursByGuideId(
            @PathVariable String guideId,
            @RequestParam(defaultValue = "0") int pageNo,
            @RequestParam(defaultValue = "10") int itemsNumber) {

        return new ResponseEntity<>(
                tourService.getToursByGuideIdAndPagination(guideId, pageNo,
itemsNumber), HttpStatus.OK);
    }

    @GetMapping("/page")
    public ResponseEntity<TourDtoResponse> getPageWithFiltersAndPagination(
            @RequestParam(required = false, defaultValue = "0") int pageNo,
            @RequestParam(required = false, defaultValue = "10") int
itemsNumber,
            @RequestParam(required = false) Date startDate,
            @RequestParam(required = false) UUID typeId,
            @RequestParam(required = false) Float price) {

        return new ResponseEntity(
                tourService.getTourPageWithFiltersAndPagination(
                        typeId,
                        startDate,
                        price,
                        pageNo,
                        itemsNumber), HttpStatus.OK);
    }
}

@Service
public class TourServiceImpl implements TourService {

    private TourRepository tourRepository;
    private TourTypeRepository tourTypeRepository;

    @Autowired
    public TourServiceImpl(TourRepository tourRepository, TourTypeRepository
```

```
tourTypeRepository) {
        this.tourRepository = tourRepository;
        this.tourTypeRepository = tourTypeRepository;
    }

    @Override
    @Transactional
    public TourDtoResponse create(TourDtoRequest tourDtoRequest, String
guideId) {
        TourType type = checkTourTypeExistence(tourDtoRequest.getType());

        Tour tour = new Tour();
        tour.setType(type);
        tour.setGuideId(guideId);

        tour =
tourRepository.save(TourMapper.tourDtoRequestToTour(tourDtoRequest, tour));
        return TourMapper.tourToTourDtoResponse(tour);
    }

    @Override
    public TourDtoResponse update(TourDtoRequest tourDtoRequest, UUID tourId)
{
        TourType type = checkTourTypeExistence(tourDtoRequest.getType());

        var tourOptional = tourRepository.findById(tourId);
        if (tourOptional.isEmpty()) {
            throw new RuntimeException("No tour found to update");
        }

        var tour = tourOptional.get();
        tour.setType(type);
        tour =
tourRepository.save(TourMapper.tourDtoRequestToTour(tourDtoRequest, tour));
        return TourMapper.tourToTourDtoResponse(tour);
    }

    @Override
    public void delete(UUID tourId) {
        var tourOptional = tourRepository.findById(tourId);
        if (tourOptional.isEmpty()) {
            throw new RuntimeException("No tour to delete.");
        }
        tourRepository.deleteById(tourId);
    }
    @Override
    public TourDtoResponse getById(UUID tourId) {
        var tourOpt = tourRepository.findById(tourId);
        if (tourOpt.isEmpty()) {
            throw new RuntimeException("Tour not found.");
        }
        return TourMapper.tourToTourDtoResponse(tourOpt.get());
    }

    @Override
    public Collection<TourDtoResponse> getToursByGuideIdAndPagination(String
guideId, int pageNo, int itemsNumber) {
        return TourMapper.toursToTourDtosResponse(tourRepository
                .findToursByGuideId(guideId, PageRequest.of(pageNo,
itemsNumber)));
    }
```

```
    private TourType checkTourTypeExistence(TourTypeDto typeDto) {
        var typeOptional = tourTypeRepository.findById(typeDto.getId());
        if (typeOptional.isEmpty()) {
            throw new RuntimeException("Tour type not found");
        }
        return typeOptional.get();
    }

    @Override
    public TourPageDtoResponse getTourPageWithFiltersAndPagination(
            UUID typeId, Date startDate, Float price, int pageNo, int
itemsNumber) {

        Page<Tour> tourPage;
        startDate = startDate == null ? new
Date(Calendar.getInstance().getTimeInMillis()) : startDate;
        var priceVal = price == null ? Float.MAX_VALUE : price;

        if (typeId == null) {
            tourPage =
tourRepository.getTourPagesByStartDateGreaterThanEqualAndPriceLessThanEqual(
                    startDate,
                    priceVal,
                    PageRequest.of(pageNo, itemsNumber)
            );
        } else {
            tourPage =
tourRepository.getTourPagesByTypeIdAndStartDateGreaterThanEqualAndPriceLessTh
anEqual(
                    typeId,
                    startDate,
                    priceVal,
                    PageRequest.of(pageNo, itemsNumber)
            );
        }

        var page = new TourPageDtoResponse(

TourMapper.toursToTourDtosResponse(tourPage.get().collect(Collectors.toList()
)),
                tourPage.getNumber(),
                tourPage.getTotalPages(),
                tourPage.getTotalElements(),
                tourPage.getNumberOfElements()
        );
        return page;
    }

@Data
@Entity
@Table(name = "tours")
public class Tour implements Serializable {

    @Id
    @GeneratedValue(generator = "uuid2")
    @GenericGenerator(name = "uuid2", strategy = "uuid2")
    @Column(name = "tour_id", updatable = false, nullable = false,
columnDefinition = "VARCHAR(36)")
    @Type(type = "uuid-char")
    private UUID id;

    @Column(name = "name")
```

```
    private String name;

    @Column(name = "description")
    private String description;

    @Column(name = "start_date")
    private Date startDate;
    @Column(name = "price")
    private float price;

    @Column(name = "duration")
    private long duration;

    @Column(name = "main_img_id")
    private String mainImgId;

    @Column(name = "guide_id", nullable = false, updatable = false,
columnDefinition = "VARCHAR(36)")
    private String guideId;

    @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy =
"tour")
    @JoinColumn(name = "loc_id", referencedColumnName = "loc_id")
    private Location location;

    @ManyToOne(cascade = CascadeType.MERGE, fetch = FetchType.LAZY)
    @JoinColumn(name = "type_id", referencedColumnName = "type_id")
    private TourType type;

    @CreationTimestamp
    @Column(name = "created_at", updatable = false, nullable = false)
    private Timestamp createdAt;

    @UpdateTimestamp
    @Column(name = "updated_at")
    private Timestamp updatedAt;
}
```

Config-Service

```
@SpringBootApplication
@EnableConfigServer
@EnableEurekaClient
public class ConfigServerApplication {
   public static void main(String[] args) {
      SpringApplication.run(ConfigServerApplication.class, args);
   }
}
```

Registry-Service

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaRegistryApplication {
   public static void main(String[] args) {
      SpringApplication.run(EurekaRegistryApplication.class, args);
   }
}
```

Tourist-Service

```
@RestController
@RequestMapping("/tourist")
public class TouristController {

    private TouristService touristService;

    @Autowired
    public TouristController(TouristService touristService) {
        this.touristService = touristService;
    }

    @GetMapping("/all")
    public ResponseEntity<List<TouristDtoResponse>> getAll() {
        return new ResponseEntity<>(touristService.getAll(), HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<TouristDtoResponse> getById(@PathVariable UUID id)
{
        return new ResponseEntity<>(touristService.getById(id),
HttpStatus.OK);
    }

    @PostMapping
    public ResponseEntity<TouristDtoResponse> create(@RequestBody
TouristDtoRequest touristDtoRequest) {
        return new ResponseEntity<>(touristService.create(touristDtoRequest),
HttpStatus.CREATED);
    }

    @PutMapping("/{id}")
    public ResponseEntity<TouristDtoResponse> update(@RequestBody
TouristDtoRequest touristDtoRequest, @PathVariable UUID id) {
        return new ResponseEntity<>(touristService.update(touristDtoRequest,
id), HttpStatus.OK);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity deleteById(@PathVariable UUID id) {
        touristService.delete(id);
        return new ResponseEntity(HttpStatus.OK);
    }

    @PostMapping("/reg")
    public ResponseEntity<TouristDtoResponse> registerTourist(@RequestBody
TouristDtoRegRequest touristDtoRegRequest) {
        return new
ResponseEntity(touristService.registerTourist(touristDtoRegRequest),
HttpStatus.CREATED);
    }
}

@Data
@Entity
@Table(name = "tourists")
public class Tourist implements Serializable {

    @Id
    @GeneratedValue(generator = "uuid2")
    @GenericGenerator(name = "uuid2", strategy = "uuid2")
```

```
    @Column(name = "tourist_id", updatable = false, nullable = false,
columnDefinition = "VARCHAR(36)")
    @Type(type = "uuid-char")
    private UUID id;

    @Column(name = "username", unique = true)
    private String username;

    @Enumerated(value = EnumType.STRING)
    @Column(name = "role", nullable = false)
    private Role role;

    @Column(name = "email", unique = true, nullable = false)
    private String email;

    @Column(name = "phone", unique = true)
    private String phone;

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

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

    @CreationTimestamp
    @Column(name = "created_at", updatable = false, nullable = false)
    private Timestamp createdAt;

    @UpdateTimestamp
    @Column(name = "updated_at")
    private Timestamp updatedAt;

    @Column(name = "password", nullable = false)
    private String password;

    @Enumerated(value = EnumType.STRING)
    @Column(name = "account_status", nullable = false)
    private AccountState accountState;
}

@Service
public class TouristServiceImpl implements TouristService {

    private TouristRepository touristRepository;
    private PasswordEncoder encoder;

    @Autowired
    public TouristServiceImpl(TouristRepository touristRepository,
PasswordEncoder encoder) {
        this.touristRepository = touristRepository;
        this.encoder = encoder;
    }

    @Override
    public TouristDtoResponse create(TouristDtoRequest touristDtoRequest) {
        Tourist tourist =
touristRepository.save(TouristMapper.touristDtoRequestToTourist(touristDtoReq
uest, new Tourist()));
        return TouristMapper.TouristToTouristDtoResponse(tourist);
    }

    @Override
```

```
    public TouristDtoResponse update(TouristDtoRequest touristDtoRequest,
UUID touristId) {
        var optionalTourist = touristRepository.findById(touristId);
        Tourist tourist =
touristRepository.save(TouristMapper.touristDtoRequestToTourist(touristDtoReq
uest, optionalTourist.get()));
        return TouristMapper.TouristToTouristDtoResponse(tourist);
    }

    @Override
    public void delete(UUID touristId) {
        var touristOptional = touristRepository.findById(touristId);
        touristRepository.deleteById(touristId);
    }

    @Override
    public List<TouristDtoResponse> getAll() {
        return
TouristMapper.TouristsToTouristDtosResponse(touristRepository.findAll());
    }

    @Override
    public TouristDtoResponse getById(UUID touristId) {
        var optionalTourist = touristRepository.findById(touristId);
        return
TouristMapper.TouristToTouristDtoResponse(optionalTourist.get());
    }

    @Override
    public TouristDtoRegResponse registerTourist(TouristDtoRegRequest
touristDtoRegRequest) {
        var touristOptional =
touristRepository.findByEmail(touristDtoRegRequest.getEmail());
        if (touristOptional.isPresent()) {
            return new TouristDtoRegResponse(RegStatus.FAILED, "Registration
failed. Email already exists.");
        }

        Tourist tourist = new Tourist();
        tourist.setEmail(touristDtoRegRequest.getEmail());
        tourist.setFirstName(touristDtoRegRequest.getFirstName());
        tourist.setLastName(touristDtoRegRequest.getLastName());

tourist.setPassword(encoder.encode(touristDtoRegRequest.getPassword()));
        tourist.setAccountState(AccountState.NEW);
        tourist.setRole(Role.TOURIST);
        touristRepository.save(tourist);

        return new TouristDtoRegResponse(RegStatus.NEW, "success");
    }

    public Tourist getByEmail(String email) throws SQLException {

        var touristOptional =  touristRepository.findByEmail(email);
        return touristOptional.get();
    }
}
```

ДОДАТОК Б. ЧАСТИНА ПРОГРАМНОГО КОДУ ВЕБ-КЛІЄНТА

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule, HttpRequest } from '@angular/common/http';
import { JwtHelperService, JwtModule } from "@auth0/angular-jwt";

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { CommonModule, DatePipe } from '@angular/common';
import { HeaderComponent } from './header/header.component';
import { FooterComponent } from './footer/footer.component';
import { TourModule } from './tour/tour.module';
import { TourFormComponent } from './guide/tour-form/tour-form.component';
import { GoogleMapsModule } from '@angular/google-maps';
import { GuideModule } from './guide/guide.module';
import { AuthenticationModule } from './authentication/authentication.module';
import { NgbModule } from '@ng-bootstrap/ng-bootstrap';

export function tokenGetter(request?: HttpRequest<any>) {
  if (request) {
    console.log(request.url)
    if (request.url.includes("tourist")) {
      console.log("tourist_access_token")
      return localStorage.getItem("tourist_access_token")
    } else if (request.url.includes("guide")) {
      console.log("guide_access_token")
      localStorage.getItem("guide_access_token");
    }
  }
  return null;
}

@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    FooterComponent,
    TourFormComponent
  ],
  imports: [
    CommonModule,
    BrowserModule,
    HttpClientModule,
    AppRoutingModule,
```

```
    FormsModule,
    ReactiveFormsModule,
    GoogleMapsModule,
    TourModule,
    GuideModule,
    AuthenticationModule,
    JwtModule.forRoot({
      config: {
        tokenGetter,
        allowedDomains: ["http://localhost:8011"],
        throwNoTokenError: true,
      },
    }),
    NgbModule,
    DatePipe,
  ],
  providers: [
    JwtHelperService
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

```
const routes: Routes = [
  { path: '',   redirectTo: '/tours', pathMatch: 'full' },
  {
    title: "tours",
    path: "tours",
    component: TourSearchListComponent
  },
  {
    title: "guid_control",
    path: "guid_control",
    component: TourFormComponent
  },
  {
    title: "guid_control_tour_list",
    path: "guid_control_tour_list",
    component: GuideTourListComponent
  },
  {
    title: "tour-details",
    path: "tour-details/:tourId",
    component: TourDetailsComponent
  },
  {
    title: "auth",
```

```
    path: "auth",
    component: AuthenticationComponent,
    children: [
      {
        title: "guide-login",
        path: "guide-login",
        component: GuideLoginComponent
      },
      {
        title: "guide-reg",
        path: "guide-reg",
        component: GuideRegComponent
      },
      {
        title: "tourist-login",
        path: "tourist-login",
        component: TouristLoginComponent
      },
      {
        title: "tourist-reg",
        path: "tourist-reg",
        component: TouristRegComponent
      },
    ]
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }


@Injectable({
  providedIn: 'root'
})
export class TourService {

  private baseURL = "http://localhost:8011/tour-service/tour/%endpoint%"
  private readonly getPage = "page"
  private readonly createEndpoint = "guide/"

  constructor(private http: HttpClient, public datepipe: DatePipe) { }

  public createTourWithGuideId(tour: Tour, guideId: string): Observable<Tour> {
    return this.http.post<Tour>(this.getFinalUrl(guideId), tour);
  }
```

```
  public getToursByGuideId(guideId: string, pageNo = 0, itamsNumber = 20):
Observable<Tour[]> {
    const params = new HttpParams().set('pageNo', pageNo).set('itamsNumber',
itamsNumber);
    return this.http.get<Tour[]>(this.getFinalUrl(this.createEndpoint + guideId),
{ params });
  }

  public getToursById(tourId: string): Observable<Tour> {
    return this.http.get<Tour>(this.getFinalUrl(tourId));
  }

  private getFinalUrl(tail: string): string {
    return this.baseURL.replace("%endpoint%", tail);
  }

  public getTourPageWithFiltersAndPagination(searchParams: TourSearchParams = {
    pageNo: 0,
    itamsNumber: 10,
    typeId: null,
    startDate: this.datepipe.transform(Date.now(), 'yyyy-MM-dd'),
    price: Number.MAX_SAFE_INTEGER
  }): Observable<TourPage> {

    let params = new HttpParams()
      .set('pageNo', searchParams.pageNo ? searchParams.pageNo : 0)
      .set('itamsNumber', searchParams.itamsNumber ? searchParams.itamsNumber :
10)
      .set('price', searchParams.price ? searchParams.price :
Number.MAX_SAFE_INTEGER);


    if (searchParams.typeId) {
      params = params.set('typeId', searchParams.typeId);
    }

    if (searchParams.startDate) {
      params = params.set('startDate', searchParams.startDate);
    }


    return this.http.get<TourPage>(this.getFinalUrl(this.getPage), { params });
  }
}

@NgModule({
  declarations: [
    TourComponent,
    TourSearchListComponent,
```

```
    TourSearchFormComponent,
    TourDetailsComponent
  ],
  imports: [
    CommonModule,
    FormsModule,
    ReactiveFormsModule,
    NgbModule,
  ],
  providers: [TourService, DatePipe],
  exports: [
    TourComponent,
    TourSearchListComponent,
    TourDetailsComponent
  ]
})
export class TourModule { }

<form id="searchForm" [formGroup]="searchForm" (ngSubmit)="searchByForm()">
    <select id="type" name="type" formControlName="typeId">
        <option value="" selected>Будь-який</option>
        <ng-container *ngFor="let type of tourTypes$ | async">
            <option [value]="type.id">{{type.name}}</option>
        </ng-container>
    </select>
    <input id="location" type="text" formControlName="location"
placeholder="Kyiv, Ukraine">
    <input id="startDate" type="date" formControlName="startDate">
    <input id="price" type="number" formControlName="price" placeholder="Max
ціна">
    <button type="submit">Пошук</button>
</form>

#searchForm {
    display: flex;
    flex-direction: column;
    background-color: #ffd900;
    border-radius: 15px;
    padding: 30px 30px;

    input, select, button {
        margin: 20px 0px;
        font-size: 21px;
        padding: 15px 30px;
        border-radius: 15px;
        border: 1px solid #343434;
    }

    button {
```

```
        cursor: pointer;
    }
}


@Component({
  selector: 'app-tour-search-form',
  templateUrl: './tour-search-form.component.html',
  styleUrls: ['./tour-search-form.component.scss']
})
export class TourSearchFormComponent implements OnInit {

  @Output() searchEvent = new EventEmitter<TourSearchParams>();

  protected searchForm: FormGroup = this.formBuilder.group({
    typeId: [""],
    location: [""],
    startDate: [""],
    price: [""],
  });

  tourTypes$: Observable<TourType[]>;

  constructor(private formBuilder: FormBuilder, private tourTypeService:
TourTypeService) {
    this.tourTypes$ = this.tourTypeService.getAllTourTypes();
   }

  ngOnInit(): void {
  }

  searchByForm() {
    this.searchEvent.emit(this.searchForm.value as TourSearchParams);
    console.log(this.searchForm.value);
  }

}
```