Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

«Реалізація сервісу планування подорожей»

Текстова частина до курсової роботи
за спеціальністю «Комп'ютерні науки» 122

Керівник курсової роботи
доцент Ющенко Ю.О.

_____
(підпис)
"_____"_____2022 р.

Виконала студентка Кондратюк К.В.
"_____"_____2022 р.

Київ 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
доцент
_____Ющенко Ю. О.
. (підпис)
„____” _____2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентці Кондратюк Катерині

3-го курсу факультету інформатики

ТЕМА: Реалізація сервісу планування подорожей

Вихідні дані:
Зміст ТЧ до курсової роботи:

Вступ
Анотація
1. Аналіз предметної області
2. Вибір технологій та засобів для розробки
3. Реалізація веб застосунку

Висновки
Список використаної літератури
Додатки (за необхідністю)

Дата видачі „___”_____2022 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Календарний план виконання курсової роботи

Тема: Реалізація сервісу планування подорожей

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	листопад 2021 р.	
2.	Огляд літератури за темою роботи	листопад-грудень 2021 р.	
3.	Вибір програмного забезпечення для роботи	січень 2022 р.	
4.	Розробка веб-застосунку (практичної частини роботи)	лютий-березень 2022 р.	
5.	Написання текстової частини роботи	квітень 2022 р.	
6.	Підготовка слайдів для презентації та доповіді	квітень 2022 р.	
7.	Надання роботи науковому керівнику для перевірки	квітень 2022 р.	
8.	Корегування роботи за результатами попереднього захисту.	квітень 2022 р.	
9.	Остаточне оформлення пояснювальної роботи та слайдів.	квітень-червень 2022 р.	
10.	Захист курсової роботи	7-14 червня 2022 р.	

Студентка Кондратюк К.В

Керівник Ющенко Ю. О.

“_____” _____

Зміст

Вступ

Одна із базових потреб українця – дослідити власну територію. Україна є найбільшою країною Європи, тому нам дійсно є на що подивитись. Попри складні часи зараз (починаючи з лютого 2022 року), ми обов`язково повернемось до стабільного життя та вирушимо у мандри рідною землею. Роботу присвячено розробці зручного сервісу для планування подорожей з урахуванням вподобань користувача щодо відвідування/ознайомлення з найбільш цікавими йому історичними пам'ятками та іншими визначними місцями. Сервіс пропонує базовий набір послуг таких, як створення подорожі з обраними розвагами, можливість перегляду, додавання, редагування та видалення відгуків до вже існуючих подорожей або розваг. Звичайно ж передбачене сортування за полями, які найцікавіші для користувача (рейтинг, назва, опис тп). Для адміністратора також можливо вносити нові подорожі та розваги, коригувати та видаляти існуючі.

Анотація

Курсова робота присвячена створенню веб-застосунку для планування подорожей з використанням фреймворку ASP.NET Core та бази даних MSSQL від компанії Microsoft. Вищеназвані технології є надзвичайно сучасними та використовуються, як у стартапах, так і в компаніях-гігантах. Тематика цього додатку є актуальною, оскільки подорож – це перше, що приходить в голову більшості людей, коли їх спитали «Яке твоє хобі?» або «Що ти любиш робити у вільний час?». Всі ми любимо відкривати для себе щось нове.

Перевагою цього сервісу над вже існуючими є простота використання, а також те, що всі функції безкоштовні, тобто доступні для кожної людини. Загалом, цей сервіс націлений на подорожування Україною, тому він може стати корисним для кожного українця середнього та молодшого віку.

У роботі передбачено 3 головні розділи:

1) аналіз предметної області:
    - для дослідження актуальності тематики;
    - постановки головних цілей та задач;
    - розгляду подібних сервісів.
2) вибір технологій та засобів для розробки:
    - для опису особливостей процесу програмування.
3) реалізація веб-застосунку:
    - безпосередньо для демонстрації архітектурних рішень у кінцевій програмі.

1. Аналіз предметної області

1.1 Огляд подібних сервісів

Наразі існує досить багато подібних сервісів, таких як “Explore!”, ”inspirock”, “wanderlog” та багато інших. Всі ці сервіси пропонують створення власних подорожей за допомогою відміток на карті, а також написання нотаток для кожного місця, яке було відмічене (наприклад, власні враження від цього місця). Також є можливість поширення твоєї подорожі для друзів або для всієї спільноти сайту. Але, на жаль, без платної підписки, користувачу доступний досить обмежений функціонал (кількість власних подорожей обмежена, так само як і кількість відміток). Навігація на сайті не зовсім інтуїтивна, що може ускладнити користувачу подорож. Ці сервіси націлені на цілий світ, тому в них немає своєї спеціалізації, а відповідно їх рекомендації базуються на Google Maps або подібних сервісах. Задля подолання цих труднощів варто створити альтернативний сервіс для планування, який більше націлений на користувача, а не на прибутковість. Тому CityTraveler є актуальним рішенням для сучасного ринку.

Рисунок 1.1 – логотип CityTraveler

1.2 Постановка цілі

Головна ціль створення застосунку для планування подорожей – надати користувачам безкоштовний ресурс, який буде зручним у використанні. Такий сервіс також допоможе у підвищенні обізнаності про Україну та її історію.

1.3 Вимоги до кінцевого застосунку

Однією з найголовніших функцій сервісу є створення подорожей з вподобаних користувачем розваг.

Також задля того, щоб користувач міг порівнювати розваги, є можливість писати відгуки, коментарі та ставити рейтинг.

Звичайно ж, користувачу надана можливість фільтрування подорожей та розваг за корисними полями (такими як назва, рейтинг, опис і тп).

Ще до вимог створення додатку можна віднести створення інформаційного сервісу, який обраховує найпопулярніші розваги та подорожі.

Наявний розподіл користувачів на адміністраторів, контент менеджерів, гостів та звичайних користувачів. Власне внесенням розваг у базу даних займається адміністратор. Контент менеджер контролює якість контенту, що публікують користувачі. Інші ролі передбачені для повсякденного користування сайтом. Звичайно ж, є можливість реєстрації на сайті, логіну та логауту.

2. Вибір технологій та засобів для розробки

2.1 Вибір середовища розробки. Visual Studio 2022 + Visual Studio Code.

Для розробки бекенд частини було обрано Visual Studio 2022. Для клієнтської частини – Visual Studio Code.

Плюси застосування Visual Studio:

- можливість створення стартових проектів (для прикладу);
- багатофункціональність (є можливість працювати з C++, C#, Node.js, Python та ін.);

- контроль всього у одному застосунку (наприклад, можна подивитися на стан серверу, використання пам`яті, є можливість побудови діаграм);
- зручна робота з сервісами контролю версій (Git, GitHub, Azure);
- можливість персоналізації зовнішнього вигляду;
- просте налагодження коду.

Плюси застосування Visual Studio Code:

- легкість встановлення;
- без платних підписок;
- різноматна кількість розширень, які надзвичайно полегшують розробку. До прикладу було використано Angular Language Service;

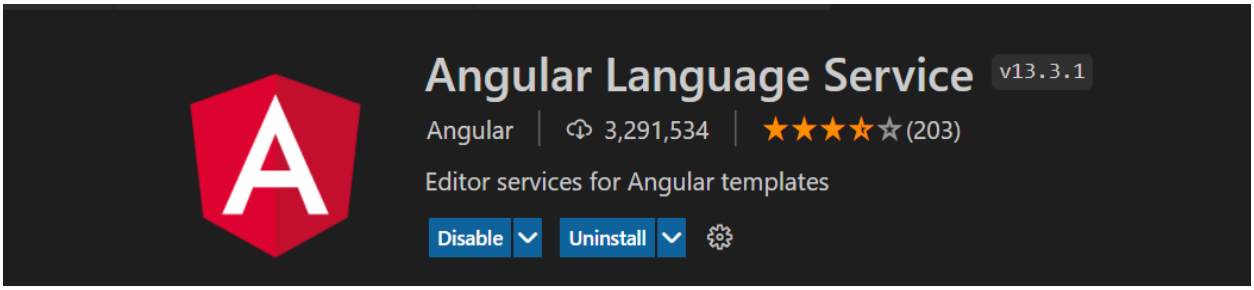

Рисунок 2.1 – плагін Angular Language Service

- зручна навігація;
- мінімальність інтерфейсу.

2.2 Клієнтська частина застосунку: Angular, препроцесор Saas.

Першою причиною для використання Angular стало те, що під час розробки можна використовувати Typescript. Оскільки це сильно типізована мова, то її

використання є набагато безпечнішим, ніж Javascript. Це також допомагає у швидкій обробці помилок, оскільки більшість із них виникнуть на етапі компіляції. Також не потрібно використовувати гетери та сетери, оскільки Typescript користується POJO (Plain Old JavaScript Object). І звичайно, Angular використовує вбудований MVC патерн, що є дуже корисно для нашого застосунку. Що не менш важливо, у Angular є досить велика спільнота, а також наявно багато документації.

Препроцесор Saas (Syntactically awesome style sheets) – це розширення css, яке надає багато переваг при розробці. Наприклад, використання змінних, «гніздування» (nesting) класів, можливість використання функцій (mixins), що може робити css динамічним. Також є досить багато зручних вбудованих операторів (наприклад, percentage(), floor(), round()).

2.3 Бекенд частина: ASP.NET Core Web API

У ASP.NET є досить багато переваг:

- постійна підтримки від розробників;
- інтеграція з сервісами Microsoft на високому рівні (що є критично, оскільки інші використані технології теж від Microsoft);
- мультиплатформенність;
- багато вбудованих сервісів (наприклад, ASP.NET Core Identety);

ASP.NET працює з шаблоном MVC . Це допомагає нам у легкій навігації проектом.

Основною метою цього шаблону є поділ програми на умовні 3 частини:

1. Controller. Мета – обробка даних, переданих користувачем.
2. View. Мета – взаємодія з користувачем.
3. Model. Мета – безпосередня реалізація методів для роботи з контролером та представленням.

2.4 Тестування: Swagger, NUnit

Swagger – це фреймворк, який допомагає візуалізувати виконання методів контролерів, а також перевірити правильність виконання цих запитів. Такий інструмент значно спрощує налагодження програми, а також написання документації (оскільки значна частина генерується автоматично).

NUnit – це фреймворк для написання юніт тестів для програм на .Net. Тестування є невід`ємною частиною процесу розробки. Зазвичай процес розробки проходить у декілька етапів, старий код потребує певних змін. Саме тому варто забезпечити механізм, який буде гарантувати відсутність регресії при змінах. NUnit і є таким інструментом.

2.5 База даних: MSSQL

MSSQL – це система управління базами даних. Вона є надзвичайно інтуїтивною у використанні, а також пропонує досить багато засобів для аналізу даних та швидкої розробки. Великою перевагою є і те, що це продукт Microsoft, тобто інтеграція з ASP.NET пройде без зайвих зусиль. Також тести показують досить хорошу продуктивність за часом виконання запитів. Більш того, MSSQL вважається однією з найбезпечніших систем. Важливим пунктом є і відновлення даних. MSSQL пропонує шляхи для отримання втрачених даних, що є критичним для великих продуктів. У разі потреби можливе автоматичне налаштування індексації за будь-яким полем у таблиці, що може значно пришвидшити процес отримання даних (read).

2.6 Технології та фреймворки для полегшення розробки. Dapper, EF Core, AutoMapper, ASP.NET Core Identity.

Застосування Dapper для найбільш складних запитів є хорошою практикою, оскільки «чистий» SQL завжди виконується швидше. Коли ж Entity Framework використовує LINQ запити та потім конвертує їх у SQL.

AutoMapper використовується для того, щоб процес мапингу відбувався автоматично. Також це допомагає уникнути дублікації коду.

ASP.NET Core Identity використовується для:

- менеджменту профілей, паролей, ролей, токенів та електронної пошти;
- функціональності логіну.

Це значно полегшує розробку, оскільки більшість процесів відбуваються автоматично. Потребується тільки правильне налаштування конфігурації.

EF Core спрощує створення структури бази даних, оскільки моделі можуть бути автоматично конвертовані у відповідні таблиці. У файлі ApplicationContext.cs задаємо всі зв`язки між моделями (один до багатьох, багато до багатьох), а також onUpdate та onDelete поведінку.

```
protected override void OnModelCreating(ModelBuilder builder)
{
    builder.Entity<UserProfileModel>().HasOne(x => x.User).WithOne(x => x.Profile).HasForeignKey<UserProfileModel>(x => x.Id).OnDelete(DeleteBehavior.Cascade);
    builder.Entity<UserProfileModel>().HasOne(x => x.User).WithOne(x => x.Profile).HasForeignKey<ApplicationUserModel>(x => x.UserId).OnDelete(DeleteBehavior.Cascade);
    builder.Entity<ApplicationUserModel>().HasMany(x => x.Images).WithOne(x => x.User).HasForeignKey(x => x.UserId).OnDelete(DeleteBehavior.Cascade);
    builder.Entity<ApplicationUserModel>().HasMany(x => x.Reviews).WithOne(x => x.User).HasForeignKey(x => x.UserId).OnDelete(DeleteBehavior.Cascade);
    builder.Entity<ApplicationUserModel>().HasMany(x => x.Trips).WithMany(x => x.Users);
    builder.Entity<StreetModel>().HasMany(x => x.Addresses).WithOne(x => x.Street).HasForeignKey(x => x.StreetId).OnDelete(DeleteBehavior.Cascade);
    builder.Entity<CommentModel>().HasOne(x => x.Review).WithMany(x => x.Comments).HasForeignKey(x => x.ReviewId).OnDelete(DeleteBehavior.ClientSetNull);
    builder.Entity<ReviewModel>().HasMany(x => x.Images).WithOne(x => x.Review).HasForeignKey(x => x.ReviewId).OnDelete(DeleteBehavior.NoAction);

    builder.Entity<ReviewModel>().HasOne(x => x.Rating).WithOne(x => x.Review).HasForeignKey<RatingModel>(x => x.ReviewId).OnDelete(DeleteBehavior.ClientSetNull);
    builder.Entity<AddressModel>().HasOne(x => x.Coordinates).WithOne(x => x.Address).HasForeignKey<AddressModel>(x => x.CoordinatesId).OnDelete(DeleteBehavior.NoAction);

    builder.Entity<StreetModel>().HasMany(x => x.Coordinates).WithOne(x => x.Street).HasForeignKey(x => x.StreetId).OnDelete(DeleteBehavior.Cascade);
    builder.Entity<EntertaimentModel>().HasOne(x => x.Address).WithOne(x => x.Entertaiment).HasForeignKey<EntertaimentModel>(x => x.AddressId).OnDelete(DeleteBehavior.Cascade);
    builder.Entity<EntertaimentModel>().HasMany(x => x.Reviews).WithOne(x => x.Entertaiment).HasForeignKey(x => x.EntertainmentId).OnDelete(DeleteBehavior.Cascade);
    builder.Entity<TripModel>().HasMany(x => x.Reviews).WithOne(x => x.Trip).HasForeignKey(x => x.TripId).OnDelete(DeleteBehavior.Cascade);
    builder.Entity<EntertaimentModel>().HasMany(x => x.Images).WithOne(x => x.Entertaiment).HasForeignKey(x => x.EntertaimentId).OnDelete(DeleteBehavior.Cascade);
    builder.Entity<EntertaimentModel>().HasOne(x => x.AveragePrice).WithOne(x => x.Entertaiment).HasForeignKey<EntertaimentPriceModel>(x=>x.EntertaimentId).OnDelete(DeleteBehavior.Cascade);
    builder.Entity<TripModel>().HasMany(x => x.Images).WithOne(x => x.Trip).HasForeignKey(x => x.TripId).OnDelete(DeleteBehavior.Cascade);
    builder.Entity<TripModel>().HasOne(x => x.Price).WithOne(x => x.Trip).HasForeignKey<TripPriceModel>(x => x.TripId).OnDelete(DeleteBehavior.Cascade);
    builder.Entity<TripModel>().HasMany(x => x.Entertaiments).WithMany(x => x.Trips);

    base.OnModelCreating(builder);
}
```

Рисунок 2.2 – Метод створення зв`язків у базі даних

Також для подальшого використання запитів з LINQ нам варто створити відповідні колекції наших моделей.

```
5 references
public DbSet<UserProfileModel> UserProfiles { get; set; }
1 reference
public DbSet<AddressModel> Addresses { get; set; }
16 references | 2/2 passing
public DbSet<CommentModel> Comments { get; set; }
0 references
public DbSet<CoordinatesModel> Coordinates { get; set; }
15 references | 1/1 passing
public DbSet<ImageModel> Images { get; set; }
12 references
public DbSet<RatingModel> Ratings { get; set; }
40 references | 3/3 passing
public DbSet<EntertaimentModel> Entertaiments { get; set; }
33 references | 6/6 passing
public DbSet<ReviewModel> Reviews { get; set; }
17 references
public DbSet<StreetModel> Streets { get; set; }
47 references | 9/9 passing
public DbSet<TripModel> Trips { get; set; }
0 references
public DbSet<PriceModel> Prices { get; set; }
3 references
public DbSet<EmailTemplateModel> EmailTemplatesModels { get; set; }
```

Рисунок 2.3 – Колекції, де зберігається інформація з бази даних

3. Реалізація веб застосунку

3.1 Ієрархія класів

Загалом, у моделі винесені всі сутності, які є в базі даних. Вони наслідуються від базового класу Entity, який в свою чергу наслідується від інтерфесу IEntity. Для деяких моделей також є наслідування від IDescribable.

```
namespace CityTraveler.Domain.Entities
{
    public interface IEntity
    {
        Guid Id { get; set; }

        DateTime Created { get; set; }

        DateTime Modified { get; set; }
    }
}
```

```
namespace CityTraveler.Domain.Entities
{
    public interface IDescribable
    {
        48 references
        public string Title { get; set; }
        21 references
        public string Description { get; set; }
    }
}
```

Рисунок 3.1 – Інтерфейси IEntity та IDescribable

Також наявне «внутрішнє» наслідування. Наприклад, відгук може існувати як для подорожі, так і для розваги. Тому в нас є базова сутність Review, а від неї наслідується TripReview та EntertainmentReview. «Внутрішнє» наслідування наявне також для фотографій (Image), координат (Coordinates) да ціни (Price).

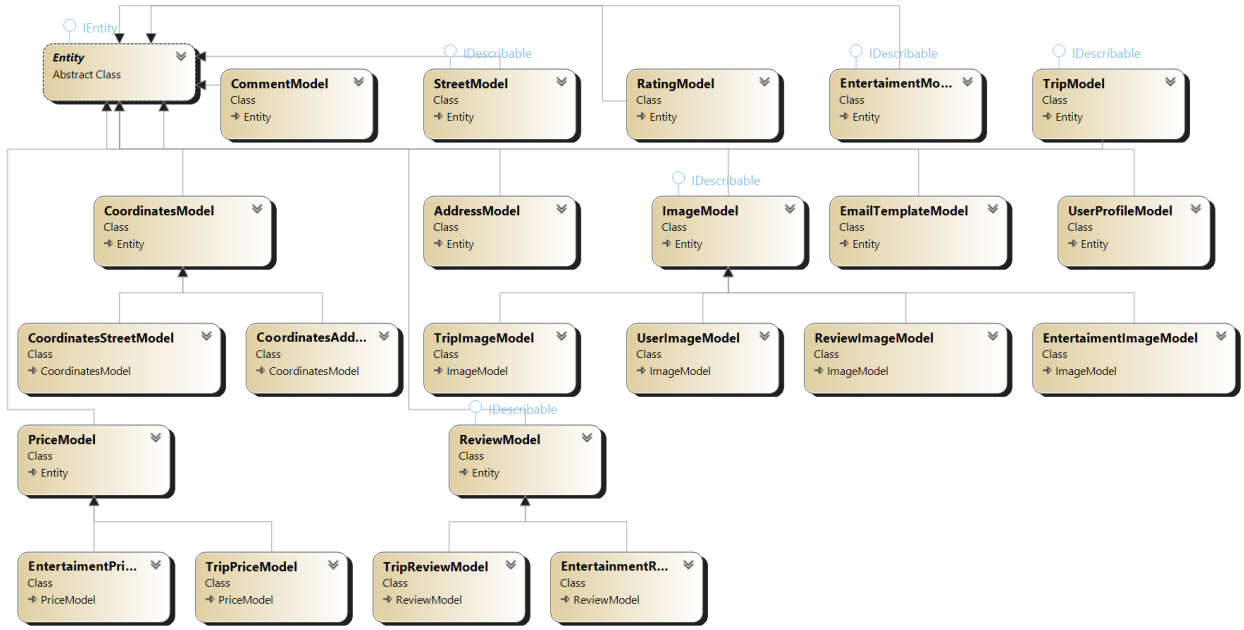

Рисунок 3.2 – Ієрархія класів

3.2 Використання Data Transfer Object (DTO)

Використано один із шаблонів проєктування Data Transfer Object (DTO) для того, щоб зробити комунікацію між фронтендом та бекендом легкою та швидкою. Це допомогло максимально мінімізувати кількість інформації, яка передається або приймається бекендом.

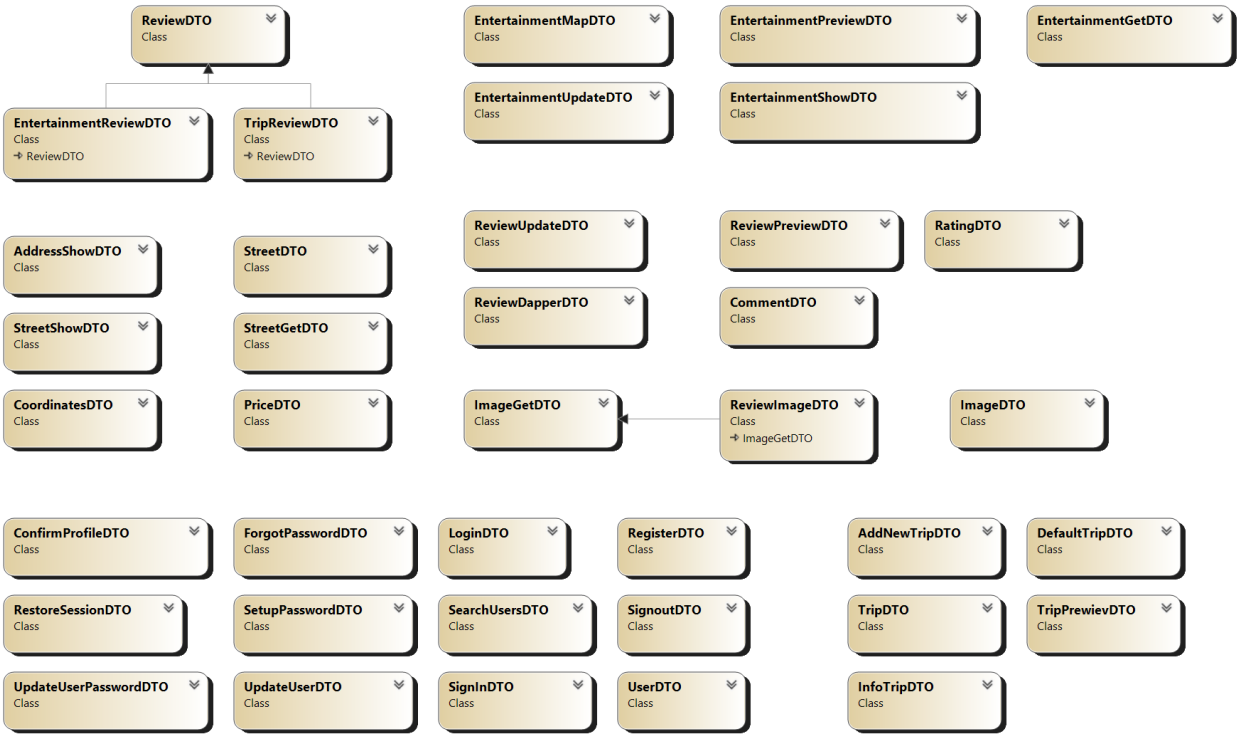

Рисунок 3.3 – Всі DTO, що використовуються в застосунку

Для деяких моделей можемо бачити декілька DTO об`єктів, оскільки при застосуванні у різних сервісах ми можемо потребувати різні поля однієї моделі.

3.3Використання enum

Для задання ролі користувача, стану подорожі та типу розваги використаний enum. Спочатку планувалося зберігати ці значення у відповідних таблицях у базі даних, як ключ-значення. Проте це виявилось невигідно по часу. Оскільки кожного разу доводилось виконувати sql процедуру (наприклад, select * from entertainmentTypes where key=1). Тому вирішено використати enum.

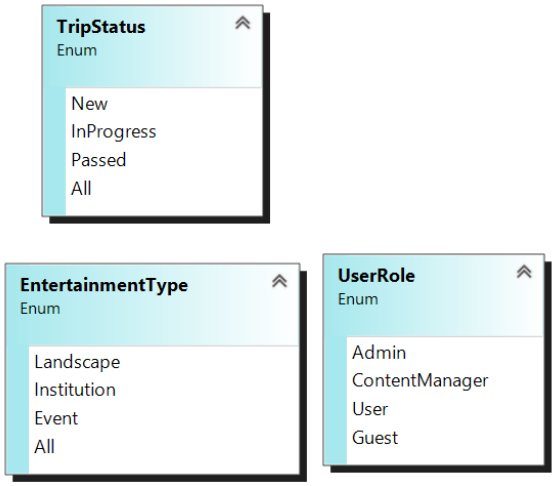

Рисунок 3.4 – Enum у застосутку

3.4 Обробка помилок

Одним із найголовніших аспектів програми є обробка нестандартних ситуації, а саме комунікація з користувачем під час виключень. Саме тому підхід до цього питання був ґрунтовним та потребував досить велику увагу. Створено багато персоналізованих виключень, а також помилки задані текстом.

Рисунок 3.5 – Персоналізовані виключення, які доступні в застосунку

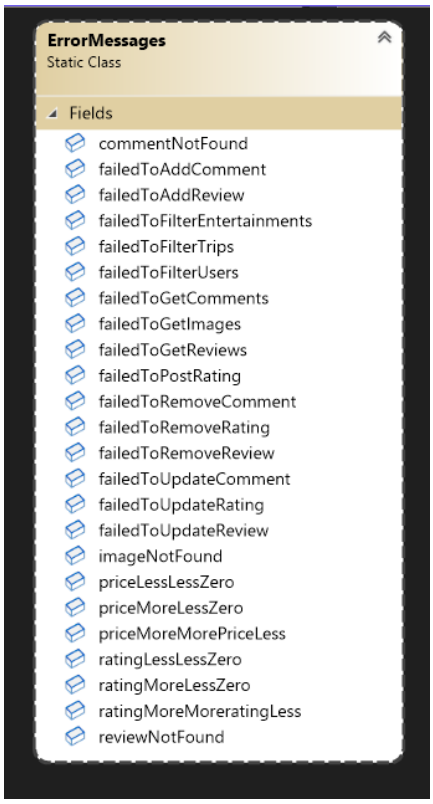

Рисунок 3.6 – Помилки, задані текстом

Кожен метод, у якому є звернення до бази даних обробляється за допомогою блока try catch, оскільки зв`язок може бути втрачений з різних причин. Також передбачено логування помилок, щоб легше зрозуміти на якому етапі сталась помилка.

```
    }
    2 references
    public async Task<IEnumerable<ReviewImageDTO>> GetImagesByReviewId(Guid reviewId)
    {
        try
        {
            if (!_dbContext.Reviews.Any(x => x.Id == reviewId))
            {
                _logger.LogError("Review not found");
                return null;
            }
            var imageModels = _dbContext.Reviews.FirstOrDefault(x => x.Id == reviewId).Images;
            return await Task.Run(() => imageModels.Select(x => _mapper.Map<ReviewImageModel, ReviewImageDTO>(x)));
        }
        catch (Exception e)
        {
            _logger.LogError($"Failed to get comments by id {e.Message}");
            return null;
        }
    }
```

Рисунок 3.7 – Приклад методу, який звертається до бази даних

3.5Сутності у базі даних

Загалом, кінцева схема бази даних вийшла досить о`ємною. Дуже багато сутностей пов`язані з використанням ASP.NET Core Identity (AspNetUsers, AspNetRoleClaims, AspNetUserTokens та інше)

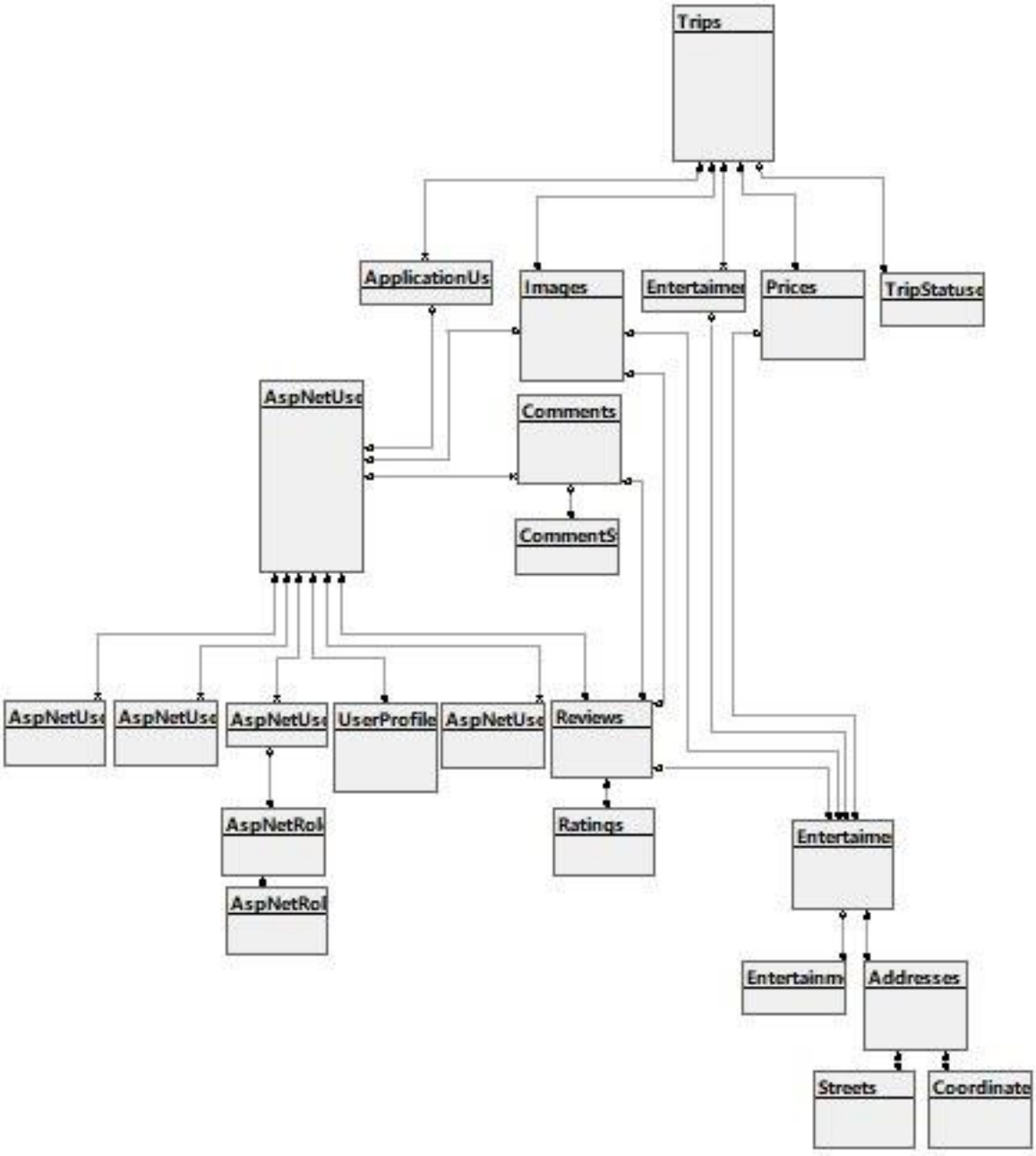

Рисунок 3.8 – Структура бази даних без назв колонок

3.6 Сервіси

Для роботи застосунку використовується 14 сервісів, кожен з яких відповідає за певну частину нашої програми.

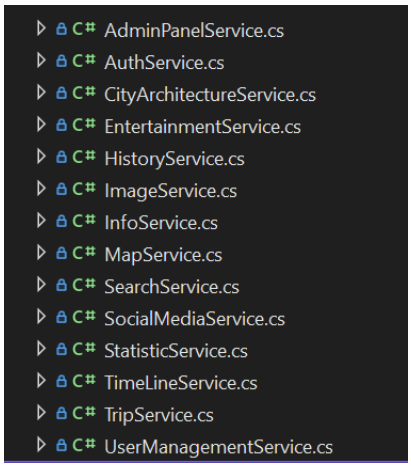

Рисунок 3.11 – Сервіси у програмі

Якщо подивитись на поля всіх сервісів, то вони скрізь подібні. У кожного з них є своя мета:

- логер ILogger<T> для логування помилок
- IDbRequestManager<T> для роботи з дапером
- IMapper для зручної конвертації об`єктів DTO->Model або навпаки
- ApplicationContext для роботи з Entity Framework.

Всі ці поля ініціалізуються за допомогою вставки залежностей (dependency injection).

```
1 reference
public static ILogger<EntertainmentService> LoggerEntertainment { set; get; }
0 references
public static IConfiguration configEntertainment { set; get; }
private readonly IDbRequestManager<ReviewDapperDTO> _requestManager;
private readonly ILogger<SocialMediaService> _logger;
private readonly ApplicationContext _dbContext;
private readonly IMapper _mapper;

1 reference
public SocialMediaService(ApplicationContext context, IMapper mapper, ILogger<SocialMediaService> logger, IConfiguration configuration)
{
    _dbContext = context;
    _mapper = mapper;
    _logger = logger;
    _requestManager = new DbRequestManager<ReviewDapperDTO>(configuration.GetConnectionString("DefaultConnection"));
}
```

Рисунок 3.12 – Вставка залежностей (dependency injection).

Також важливо відмітити, що кожен сервіс наслідується від свого інтерфейсу. Наявність інтерфейсу для сервісів робить розробку гнучкішою, а також це дає можливість використовувати вищезгадану вставку залежності.

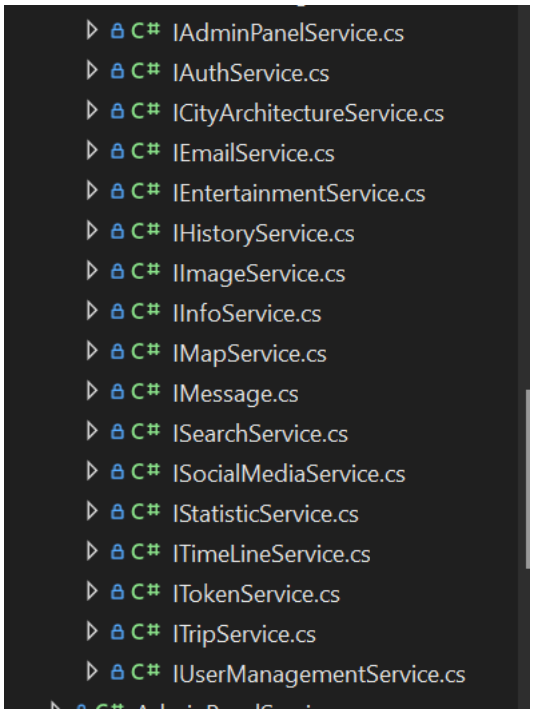

Рисунок 3.13 – Список інтерфейсів сервісів

3.7 Права доступу

Загалом, у програмі передбачено 3 різні моди доступу: адміністратор, гість, контент менеджер та користувач.

Гості можуть:

- перегладяти подорожі/ розваги/ коментарі/ відгуки/ рейтинг

- користуватись сервісом пошуку

- користуватися інформаційним сервісом (дивитися найпопулярніші подорожі, розваги)

- зареєструватися на сайті

Користувачі можуть (всі можливості гостей):

- переглядати свій профіль та змінювати інформацію в ньому

- додавати нові подорожі/ коментарі/ відгуки/ рейтинг

- редагувати свої подорожі/ коментарі/ відгуки/ рейтинг

-видаляти свої коментарі/відгуки/рейтинг

Адміністратор може (всі можливості користувачів):

- додавати розваги/ вулиці

- видаляти та редагувати подорожі/ відгуки/ коментарі/ розваги / вулиці

- дивитися зведену статистику по кількісті створених подорожей, розваг, відгуків

Контент менеджер:

- керування всім наявним контентом (фотографії, описи, відгуки і тп);

- можливість видаляти неприйнятний контент.

3.8 Тестування: атоматичне та мануальне

Автоматичне тестування було проведене за допомогою NUnit. Всього було написано 64 тести. Майже всі тести націлені на перевірку регресії у продукті.

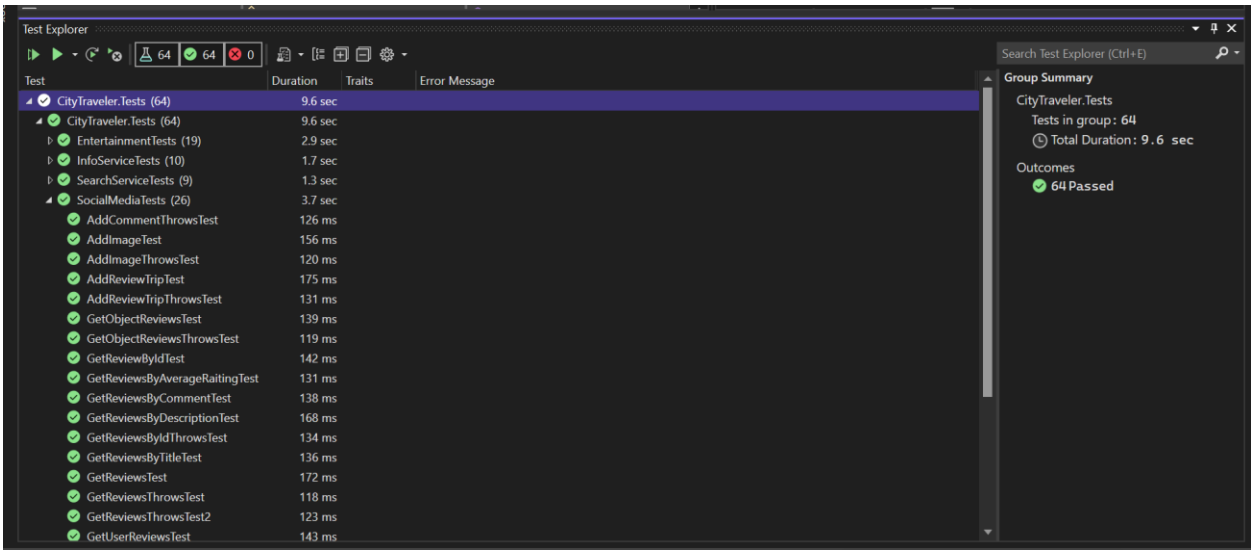

Рисунок 3.14 – unit-тестування

Мануально виконання операцій було перевірено за допомогою Swagger. До прикладу показано виконання методу отримання ДТО відгуку за його id. Загалом, Swagger виявився досить легким та зручним у використанні.

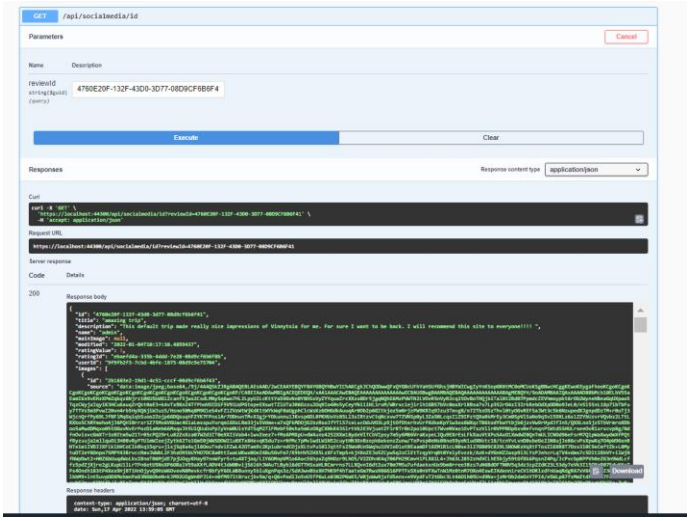

Рисунок 3.15 – Використання Swagger

Також до мануального тестування можна віднести тестування клієтської частини. Воно було проведене згідно з сервісами, які ми маємо. Наприклад, тестуючи сервіс соціальних мереж, ми заходили на кожну сторінку, де використовуються відгуки, коментарі або рейтинг та перевіряли функціональність CRUD (create, read, update, delete).

Можливість покращення полягає у тому, щоб додати інтеграційні тести (specflow), автоматичні тести клієнтської частини (такі як cypress або ж selenium).

Тобто, якщо зараз дивитися на піраміду важливості різного виду тестування, у нашому продукті покриті тільки найважливіші (юніт) та найменш важливі тести (мануальні)

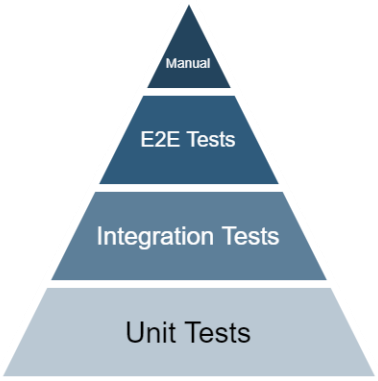

Рисунок 3.16 – Піраміда важливості тестування

3.9 Інтерфейс користувача

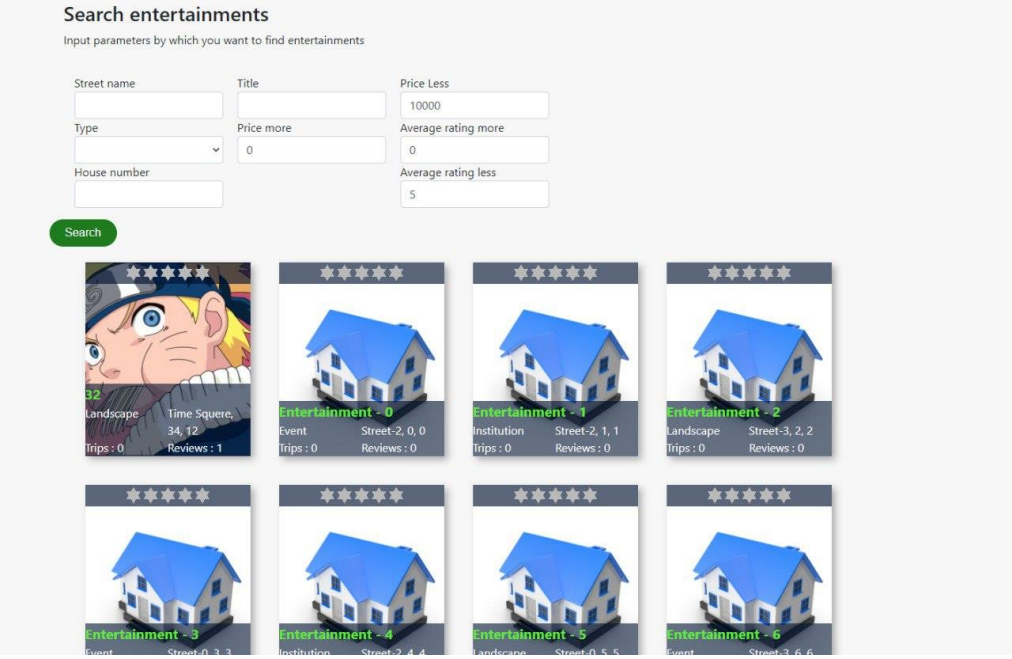

Рисунок 3.17 – Сервіс пошуку розваг

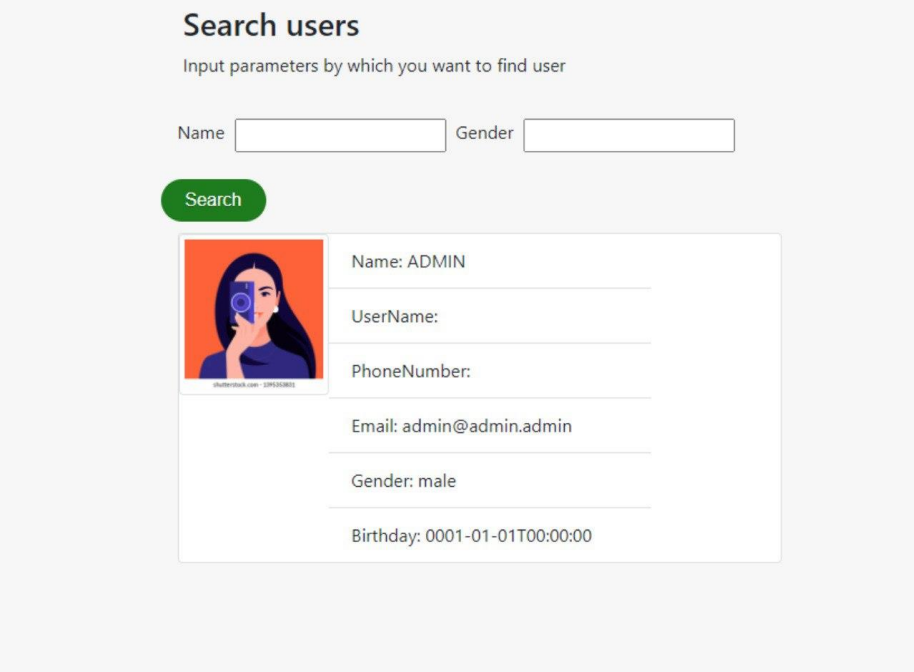

Рисунок 3.18 – Сервіс пошуку користувачі

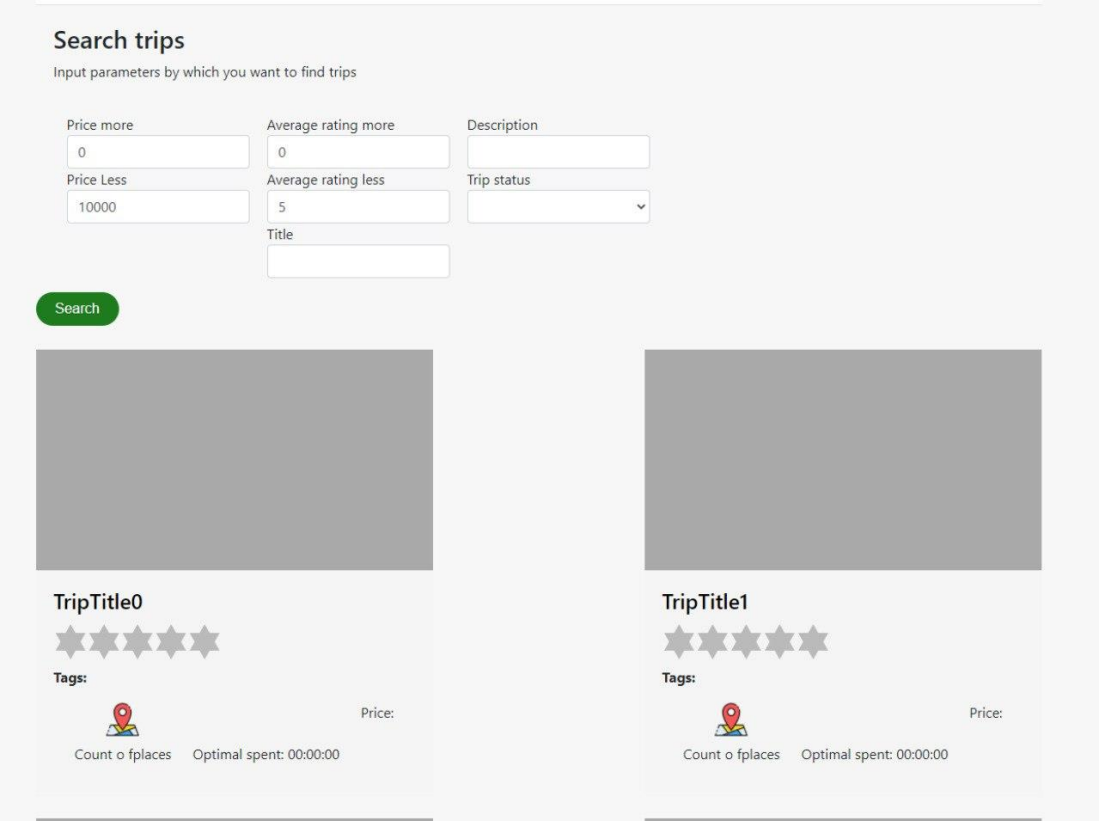

Рисунок 3.19 – Сервіс пошуку подорожей

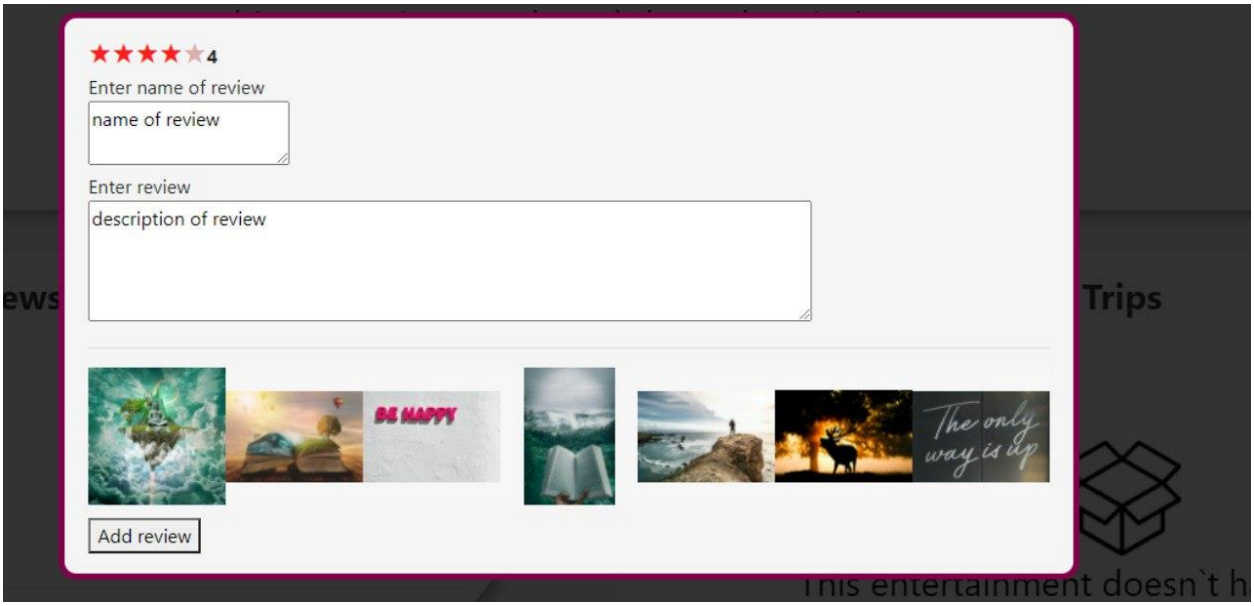

Рисунок 3.20 – Модальне вікно створення відгука

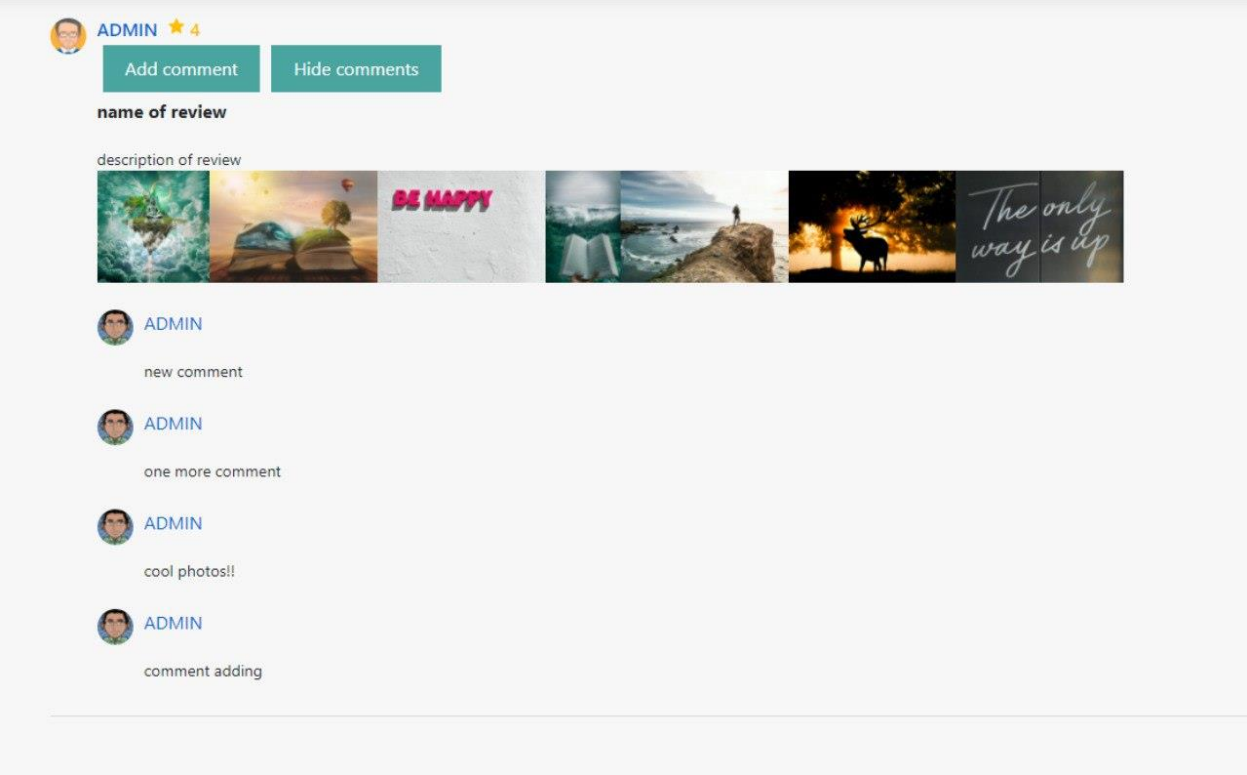

Рисунок 3.21 – Перегляд відгуків для розваги

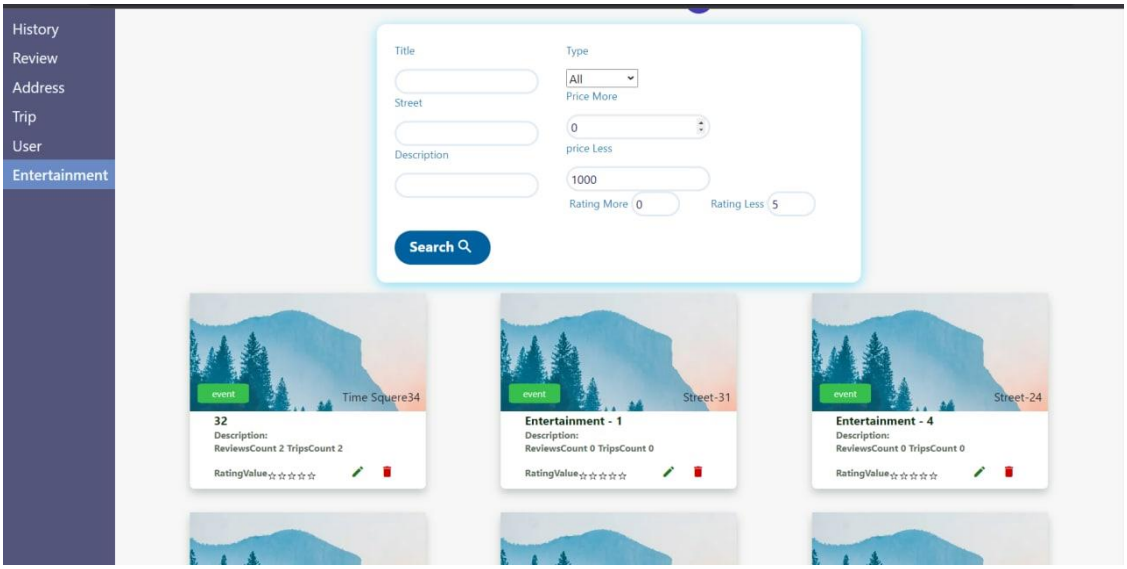

Рисунок 3.22 – Сервіс адміністратора (радагування та видалення розваг)

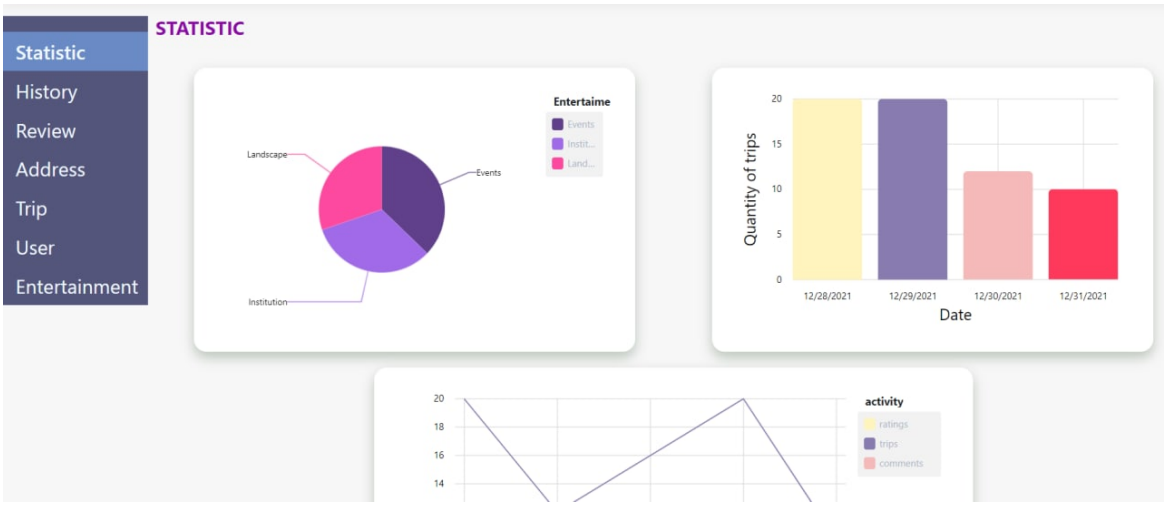

Рисунок 3.23 – Сервіс адміністратора (статистика)

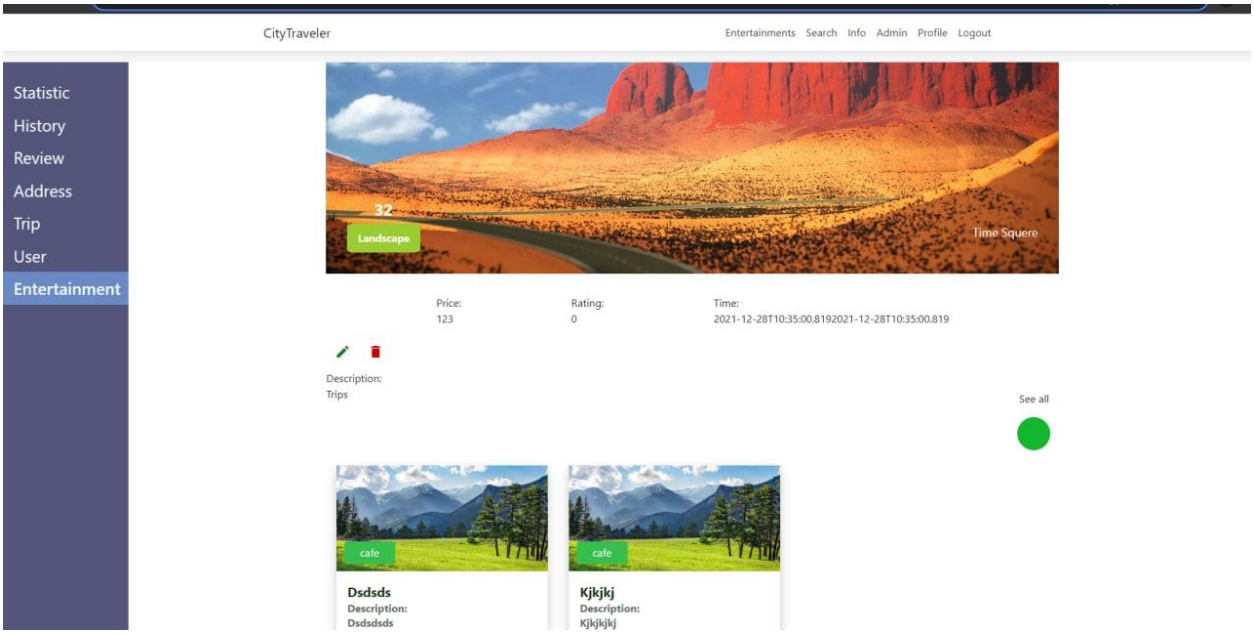

Рисунок 3.24 – Сервіс адміністратора

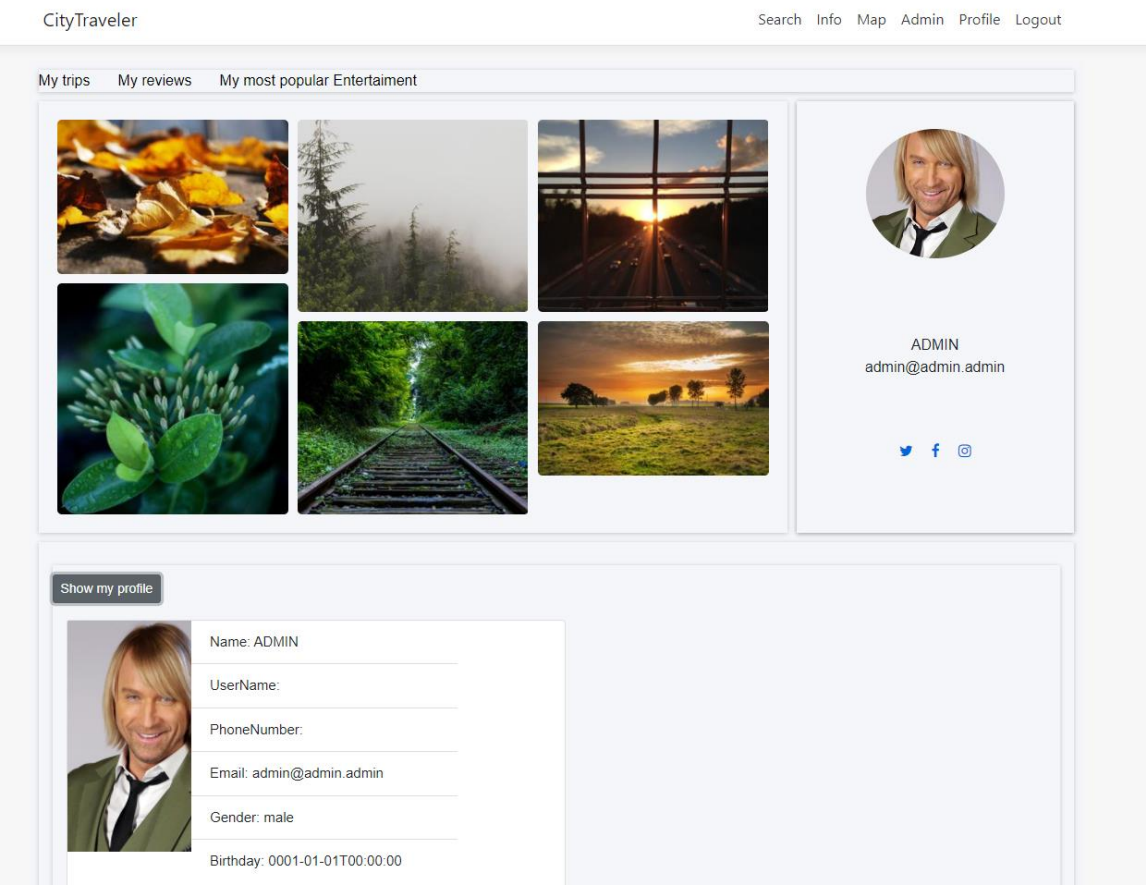

Рисунок 3.25 – Перегляд профілю користувача

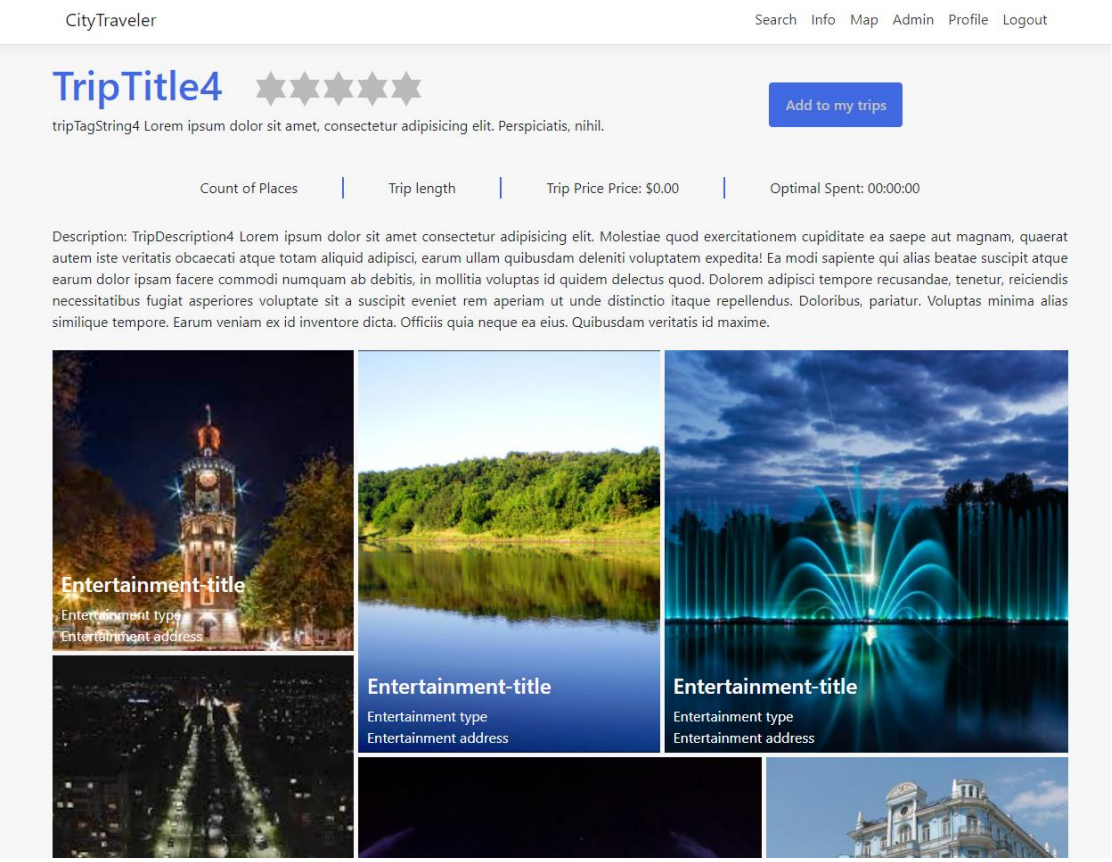

Рисунок 3.26 – Сторінка перегляду подорожі

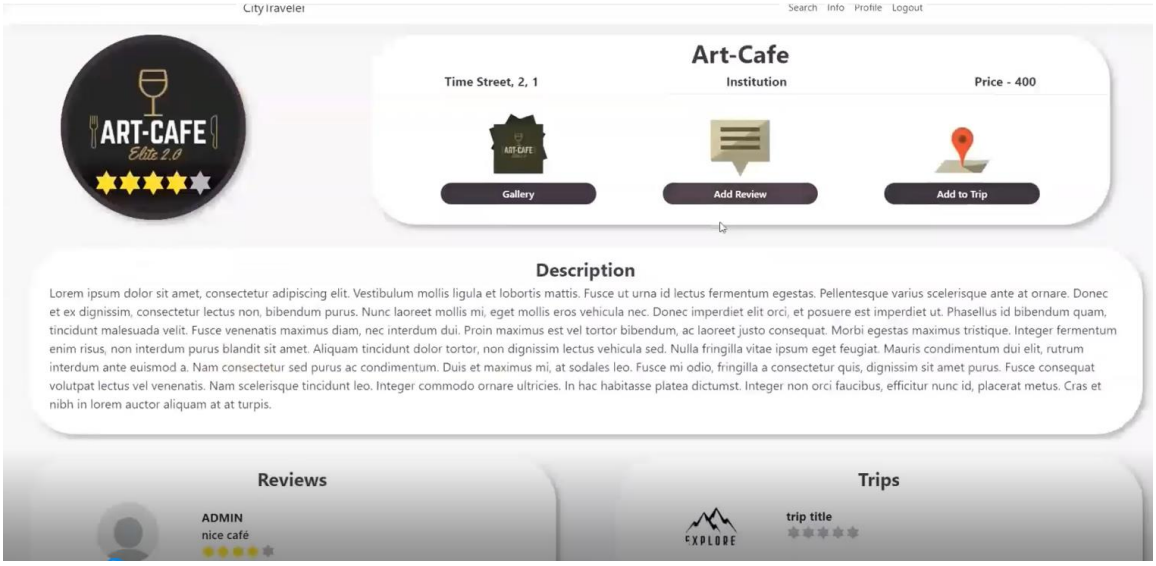

Рисунок 3.27 – Сторінка перегляду розваги (Опис, назва, рейтинг)

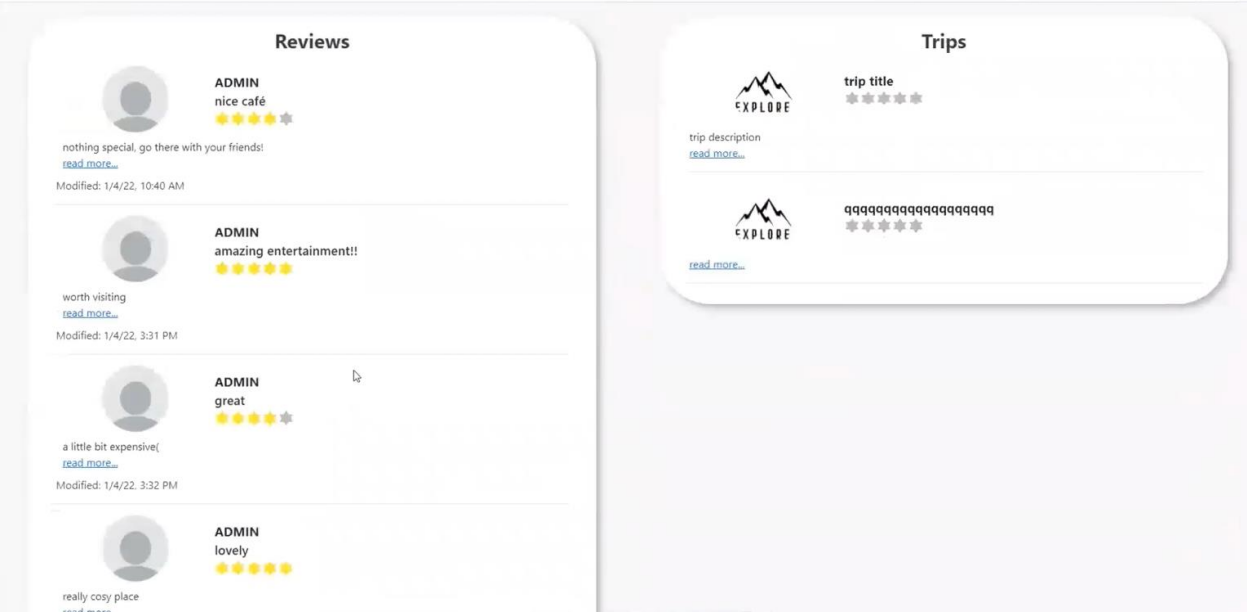

Рисунок 3.28 – Сторінка перегляду розваги (Відгуки, подорожі)

Висновки

Отже, під час виконання цієї роботи я познайомилась з технологіями Angular, препроцесором Saas. Також я покращила свої знання у використання ASP.NET Core Web API, Swagger, NUnit, MSSQL, Dapper, EF Core, AutoMapper, ASP.NET Core Identity.

Була розглянута актуальність створення онлайн сервісів пов`язаних з подорожуванням. Майже всі цілі, які були поставлені спочатку, були досягнуті.

Головним результатом роботи є робочий сервіс планування подорожей, який може бути застосований до будь-якого регіону України. Люди полюбляють подорожувати, тому такий додаток точно не залишиться без активності. Перевагою цього застосунку є простота використання, а також відсутність обмежень пов`язаних з платною підпискою.

Список використаної літератури:

1. 8 Proven Reasons You Need Angular for Your Next Development Project [Електронний ресурс] https://www.grazitti.com/blog/8-proven-reasons-you-need-angular-for-your-next-development-project/
2. Unit testing vs integration testing [Електронний ресурс] https://circleci.com/blog/unit-testing-vs-integration-testing/#:~:text=While%20unit%20tests%20always%20take,works%20in%20an%20integrated%20way.
3. MSSQL [Електронний ресурс] https://www.atlantic.net/vps-hosting/what-is-mssql/.
4. Pro ASP.NET Core MVC 2, А. Фрімен, 2017.
5. ASP.NET Core in Action, Е. Лок, 2018.

Додатки

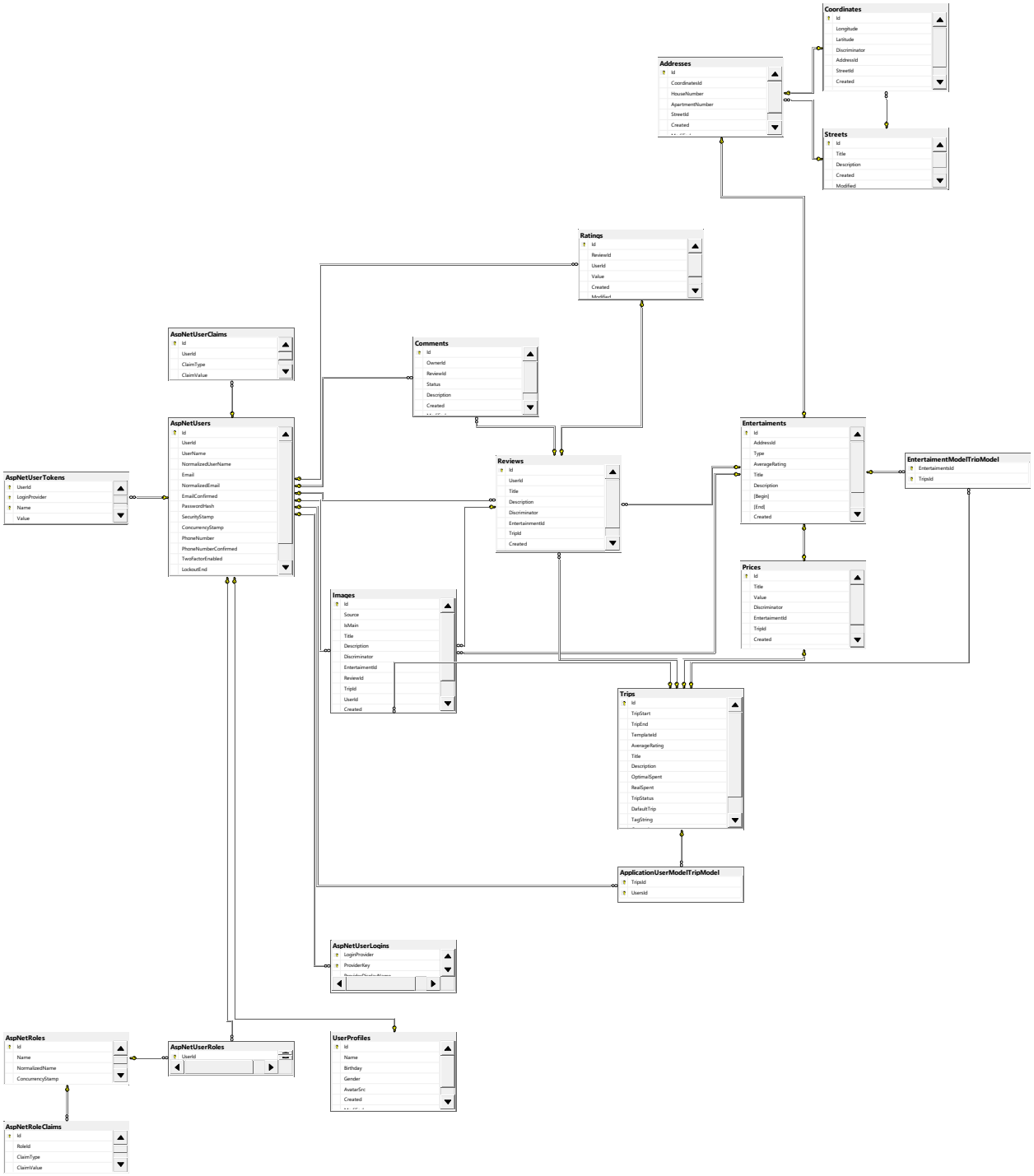

Рисунок 3.9 – Структура бази даних з назвами колонок

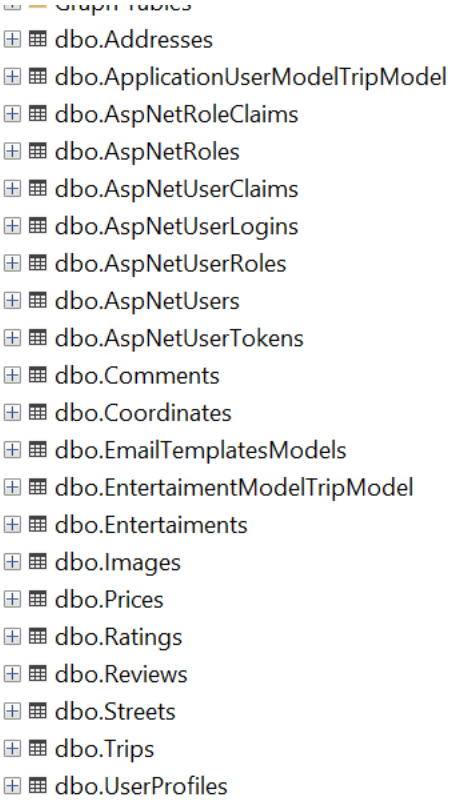

Рисунок 3.10 – Сутності у базі