**Міністерство освіти і науки України**

**ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ІВАНА ФРАНКА**

Кафедра теорії

оптимальних процесів

**Курсова робота**

**“Проектування та розробка серверної частини web-ресурсу для організації заходів”**

**Виконав:**

студент 3 курсу групи ПМА-31,

спеціальності 124 – “системний аналіз”

Байцар Р. М.

**Керівник**:

к. т. н., доцент кафедри теорії оптимальних процесів, Мельничин А. В.

Національна шкала\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів:\_\_\_ Оцінка ECTS\_\_\_

**Члени комісії**

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

Львів – 2020

**Зміст**

[**Вступ** 3](#_Toc41230560)

[**Основні визначення та термінологія** 4](#_Toc41230561)

[**Розділ 1 Постановка задачі** 6](#_Toc41230562)

[***1.1.*** ***Вибір назви*** 6](#_Toc41230563)

[***1.2.*** ***Проблема клієнта*** 6](#_Toc41230564)

[***1.3.*** ***Вибір бази даних*** 6](#_Toc41230565)

[***1.4.*** ***Вибір архітектури*** 7](#_Toc41230566)

[**Розділ 2: Onion архітектура** 8](#_Toc41230567)

[***2.1*** ***Опис та принцип роботи*** 8](#_Toc41230568)

[***2.2*** ***Файлова структура*** 9](#_Toc41230569)

[**Розділ 3: Моделі та База даних** 11](#_Toc41230570)

[***3.1.*** ***Поняття моделі*** 11](#_Toc41230571)

[***3.2.*** ***Code first та міграції*** 11](#_Toc41230572)

[***3.3.*** ***Моделі аплікації*** 12](#_Toc41230573)

[***3.4.*** ***AutoMapper*** 13](#_Toc41230574)

[**Розділ 4: Користувачі та можливості** 15](#_Toc41230575)

[***4.1.*** ***Ролі*** 15](#_Toc41230576)

[***4.2.*** ***Пагінація та фільтрація*** 15](#_Toc41230577)

[**Розділ 5: Життєвий цикл запиту** 18](#_Toc41230578)

[***5.1.*** ***Конвеєр обробки запиту*** 18](#_Toc41230579)

[***5.2. IoC Контейнер*** 19](#_Toc41230580)

[**Використані технології** 21](#_Toc41230581)

[**Висновки** 23](#_Toc41230582)

[**Джерела та література** 24](#_Toc41230583)

# **Вступ**

У сучасному світі кожна людина має доступ до інтернету де є все і одразу. Сфера надання послуг отримує з цього особливий зиск. Сучасний бізнес зручно організовувати в інтернеті оскільки не потрібно винаймати дороге приміщення, а клієнту не потрібно бути безпосередньо присутнім щоб отримати послугу.

Кожна сучасна компанія бажає мати свій сайт не лише тому що це сучасно і круто, але й тому що це збільшує цільову аудиторію її продукту, допомагає об’єктивно оцінювати проблеми споживачів та ринку застосовуючи елементи Big Data.

Також сучасний інтернет дає нам можливість хмарних обчислень тому зникає потреба встановлювати сторонній софт на своєму комп’ютері, усе що потрібно – це веб переглядач (браузер) який буде транслювати результат нашого запиту.

У своїй курсовій роботі я хочу продемонструвати приклад програми, що запущена на віддаленому сервері та обробляє запити від різних користувачів, з використанням сучасних технологій та практик.

# **Основні визначення та термінологія**

У написанні будь якого софту використовується безліч термінів та понять які я буду використовувати і надалі, тому наведу перелік їх значень

Сервер – сервером може виступати будь який комп’ютер який виконує роботу, команда на виконання якої, прийшла ззовні. Такий комп’ютер має публічне ім’я в інтернеті (звичне нам ip, наприклад 127.0.0.1, або красивий адрес www.my-site.com, який насправді також переводиться у машинний номер 0.0.0.0) та встановлений на цей комп’ютер софт що власне і опрацьовує запит який прийшов ззовні.

*Клієнт* – це програма, що здатна надсилати та отримувати відповідь від сервера. Зазвичай володіє деякою бізнес логікою. Наприклад коли ми заходимо на сайт facebook – ми завантажуємо програму клієнт прямо у свій браузер, але кожен клік по самому сайту – це запита до сервера і як результат ми бачимо стрічку новин або чиюсь сторінку.

*Бізнес логіка* – формально це просто набір методів та функцій що розміщені в програмі.

*Single-page application* – програма-клієнт особливістю якої є зміна лише деяких елементів користувацького інтерфейсу, а не перемальовування всієї html сторінки. Це збільшує швидкість роботи програми та оптимізовує її структуру. Зазвичай для написання SPA використовуються фреймворки Angular, React або Vue.

*.NET Core* – це платформа, розроблена компанією Майкрософт, що дозволяє запускати, створений з її використанням, програмний продукт на операційній системі Windows, Linux або macOS.

*ASP.Net core* – програмне забезпечення (framework), що являє собою каркас веб застосунка. Розробляється компанією Майкрософт, але знаходиться у вільному доступі на github де кожен може долучитися до розробки.

*Web API* – принцип побудови веб аплікації, що повністю розбиває клієнтську на серверну частину на дві незалежні програми які спілкуються між собою через так звані endpoints (ендпоінти) за допомогою http протоколу.

*Endpoint* – кінцева точка зв’язку. Власне це і є той інтерфейс, що забезпечує зв’язок клієнта і сервера.

*Http* – протокол передачі даних між клієнтом і сервером. Оскільки дві програми є незалежними один від одного – запит що йде до сервера має бути певним чином оформлений. Http складається з двох частин Header і Body. У хедері розміщена інформація про запит та його тип (Get, Post, Put…), в тілі інформація яку ми відправляємо з клієнта (паролі, повідомлення…). Відповідь сервера характеризується ще наявністю статусного коду:

2xx – запит оброблено правильно.

3xx – переадресація (зустрічаються рідко).

4xx – помилка клієнта у формуванні запиту.

5xx – помилка сервера в обробці запиту (неопрацьований exception у самій програмі).

*Git* – система контролю версій що дозволяє фіксувати кожен крок написання програми та зберігати його на віддаленому репозиторію. Зручний інструмент без якого не можна уявити собі сучасний процес розробки програм.

*Scrum* – підхід у розробці, що дозволяє оптимізувати роботу розділяючи її на під завдання та виставляючи порядок їхнього виконання. Дозволяє вносити правки у план роботи безпосередньо під час розробки.

# **Розділ 1 Постановка задачі**

## ***Вибір назви***

Потрібно розробити веб аплікацію для організації фестивального музичного заходу. Для будь якого розробника вибір назви – це складний процес. Робоча назва цього проекту «Spring – Fest». Я використав довільну модель заходу, а висока масштабованість застосунку дозволить з мінімальними зусиллями змінити його тематику.

## ***Проблема клієнта***

Оскільки я пишу Web API мені не потрібно турбуватися про клієнта і я повністю незалежний від того що буде бачити користувач. Але оскільки у процесі розробки потрібно налагоджувати певні елементи програми – без клієнта не обійтися. Його замінить мені Swagger – програма що імітує клієнта і дозволяє відправляти власноруч написані запити.

## ***Вибір бази даних***

Також мені потрібно десь зберігати моделі що використовуються в програмі та відображаються для користувача. Для цього чудово підходить база даних SMSQL. Розроблена Майкрософт, вона чудово працює з платформою .NET.

## ***Вибір архітектури***

Щоб забезпечити високу масштабованість продукту потрібно використати такий підхід що дозволить безболісно та з мінімальними зусиллями заміняти деякі частини коду міняючи поведінку та призначення програми. Краще всього з цим справляється Onion архітектура.

# **Розділ 2: Onion архітектура**

## ***Опис та принцип роботи***

Onion архітектура дозволяє розбити програму на шари, кожен з яких має певний порядок та виконує якусь функцію. Схематично це виглядає так:

|  |
| --- |
| Користувач |

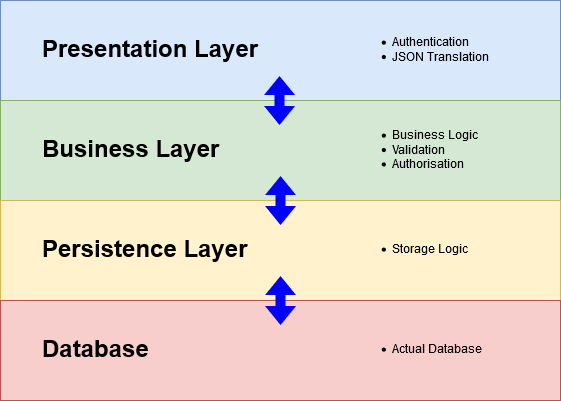


Рис. 2.1. Схематичне зображення onion архітектури

Кожен рівень може спілкуватися лише з сусідніми рівнями. Ця комунікація відбувається через внутрішні інтерфейси, що робить їх незалежними один від одного.

Коли користувач надсилає запит він потрапляє на presentation layer де проходить базову валідацію. У випадку якщо валідація не пройдена – користувач негайно отримає відповідь з кодом 400 і не прийдеться спускатися на наступні рівні.

У Business layer відбувається процес обробки запиту. Деякі запити не вимагають звернення до бази даних а опрацьовуються вже на цьому рівні (наприклад рендер зображення або множення матриць).

Persistence layer виконує роль бази даних. Зі сторони програми це виглядає ніби звернення до колекції в оперативній пам’яті, але насправді звернення може відправлятися навіть на інший комп’ютер де безпосередньо і розміщена база даних. Зазвичай на цьому рівні реалізовується так званий патерн «репозиторій».

Інформація яку користувач вводить або хоче побачити повинна десь зберігатися. База даних слугує постійним сховищем для цього.

Недоліком такої архітектури є те що ядро (нижній рівні) погано піддаються переписуванню. Мала зміна у ядрі може лавинним ефектом призвести до великих змін у вищих рівнях, але навпаки ні.

Також для підвищення масштабності використовується популярний сьогодні інструмент dependency injection. Він забезпечує слабку зв’язність компонентів програми змушуючи їх спілкуватися через інтерфейси. Якщо одну компоненту замінити то інші цього навіть не помітять і продовжать працювати. Таку заміну можна робити навіть у запущеній програмі не перекомпільовуючи код.

## ***Файлова структура***

У результаті побудови та структурування файлів програми вибудовується така файлова структура:

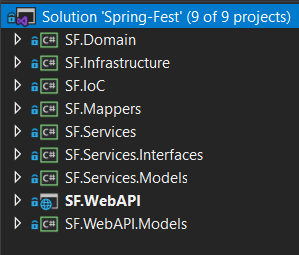


Рис. 2.2. Файлова структура проекту

Варто зауважити що кожен проект не є окремим архітектурним шаром. Часто цей шар складається з декількох проектів (наприклад SF.WebAPI та SF.WebAPI.Models). Попри це IoC Контейнер стоїть збоку від усієї архітектури оскільки через нього відбувається створення потрібних нових об’єктів зареєстрованих класів.

# **Розділ 3 Моделі та База даних**

## ***Поняття моделі***

Будь який веб сайт оперує поняттям моделі.

Модель – це відображення частини реального світу, спроба його описати в певному стані, зафіксувати цей стан. Усі ці модель повинні бути десь збережені. Для цього використовується реляційна база даних MSSQL.

Для написання цього проекту я вибрав MSSQL не спроста. Реляційна модель баз даних дозволяє будувати складні моделі з різними відношеннями (такі як 1-1, 1-N, N-N).

## ***Code first та міграції***

Для створення бази та написання таблиць я використав Entity Framework Core. Ця технологія Майкрософт дозволяє застосувати підхід code first коли ми спочатку пишемо модель (звичайний клас C#) у нашій програмі, створюємо файл міграції який Entity Framework генерує виходячи з того як ми описати нашу модель. Потім усі наші міграції застосовуються у порядку їх створення лише однією командою та будується база даних зі всіма потрібними таблицями і відношеннями.

Чим хороші міграції Entity Framework: коли ми передаємо наш застосунок для запуску його на продакшн сервері – там ще нема ніякої бази даних, тому при першому запуску вона буде створена і переведена зразу у потрібний для нас стан коли можна у неї зберігати та зчитувати дані.

Так виглядає файл міграції:

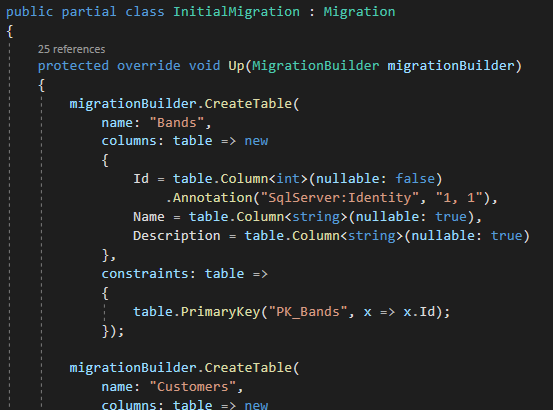


Рис. 3.1. Приклад файлу міграції

Кожному такому файлу властиві два методи: Up Та Down. Коли міграція застосовується командою Update-Database Ми рухамося в гору до останньої міграці застосовуючи метод Up. У випадку якщо нам потрібно повернутися у попередній стан бази даних – метод Down містить зміни протилежні методу Up.

## ***Моделі аплікації***

Моделі які використовуються в програмі:

Band – музичний гурт. Має ім’я, опис, фотографію, та жанри в яких виступає.

Genre – музичний жанр. Містить назву жанру. Утворює відношення N-N з Band’ом оскільки один гурт може виступати в багатьох жанрах і до одного жанру може належати багато гуртів.

Stage – сцена на якій гурт виступає. Містить назву, є елементом виступу.

Performance – виступ. Складається з гурта, сцени, фестивалю в межах якого все відбувається, має час початку та тривалість.

Festival – фестиваль. Містить час проведення та місце розміщення (локацію) а також партнерів.

Partner – партнер, спонсор. Має поле ім’я та опис. Утворює з фестивалем N-N зв’язок оскільки кожен фестиваль може співпрацювати з багатьма спонсорами, а спонсор з року в рік може підтримувати один фестиваль.

Ticket – квиток. Містить ціну, дату початку фестивалю, тривалість (один, два або всі дні) та тип (парковка, наметове містечко, або власне фестиваль).

Customer – людина що купляє квиток повинна ввести своє ім’я та пошту. Це допомагає збирати деяку інформацію про фестиваль та його аудиторію, а також необхідна для здійснення покупки.

Purchase – модель покупки. Містить інформацію про того хто робить цю покупку, квиток який був куплений, його унікальний код (бар код) та доступність цього квитка (два рази використати один квиток не вдасться).

Admin – спеціальна модель що зберігає інформацію адміністраторів: логін, хеш пароля, та хеш - сіль (потрібна щоб відновити хеш введеного паролю).

Ось так виглядає модель відображення таблиці гурту та файл конфігурації цієї таблиці:

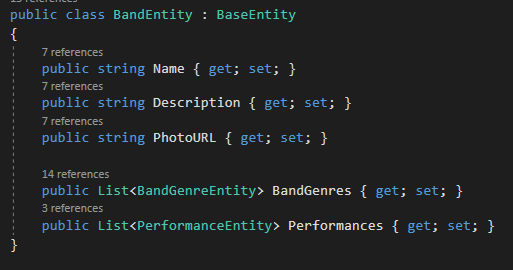
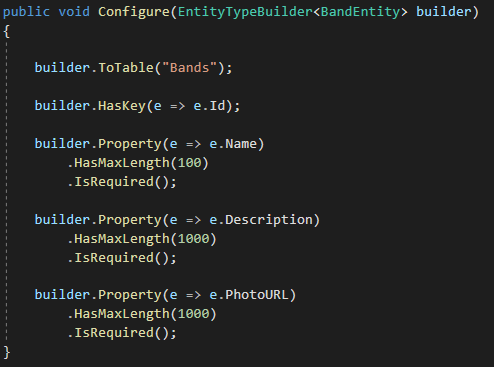
 

Рис. 3.2. Модель відображення гурту. Рис. 3.3. Файл конфігурації

## ***AutoMapper***

Усі ці моделі дублюються на кожному шарі та мають різні форми. Наприклад коли ми створюємо нову групу то в неї ще немає id у базі даних тому модель CreateBand буде відрізнятися від моделі яка повертається з бази.

Переводити моделі з одного рівня на інших допомагає AutoMapper. Його конфігурація займає мінімум зусиль, це економить безліч часу при розробці.

Приклад конфігурації мапінгу моделей рівня сервісів на ядра: 

Рис. 3.4. Файл конфігурації AutoMapper

# **Розділ 4: Користувачі та можливості**

## ***Ролі***

Сайт забезпечує можливість анонімного перегляду – будь хто може зайти та глянути список гуртів що виступають, зробити покупку.

Також для адміністрації присутня роль адміна. Ввівши логін та пароль ми отримуємо повний доступ до усіх можливостей сервера. У базі даних завжди присутній принаймні один адміністратор (створюється за замовченням з логіном MasterOfPuppets і паролем Admin12345).

Коли адміністратор авторизується йому буде відправлено спеціальний JWT токен який є дійсним протягом 7 днів (можна змінювати у файлі налаштування). З цим токеном можна здійснювати звернення до захищених endpoints, що перевіряють наявність дійсного токена, та отримувати розширений функціонал.

Які саме можливості є у адміністратора? Можна створювати інших адміністраторів, редагувати їх логіни та паролі, видаляти. Проте система не дозволить видалити останнього адміністратора. Також в обов’язок адмінів входить створення гуртів, виступів, фестивалів, та квитків, є можливість усі ці сутності редагувати та видаляти. Адміністратор має можливість переглядати статистику за фестиваль: кількість проданих квитків певного типу, кількість покупців, загальний прибуток від цих квитків).

## ***Пагінація та фільтрація***

У програмі реалізована пагінація та фільтрація. Немає сенсу витягувати усі наявні в базі даних гурти якщо на сторінку їх влізає не більше десяти. Користувач сам вибирає потрібну йому сторінку та її розмір і відповідно до запиту відбудеться певний select з бази даних. Це значно зменшує час виконання запиту та зменшує навантаження на систему. Фільтрація дозволяє виводити об’єкти що відповідають певному критерію. Наприклад виводити гурти що виступають у певному жанрі або на певній сцені.

Реалізація цієї простої, на перший погляд, ідеї вимагає певних зусиль. Як виглядає SQL скрипт з певним критерієм:

SELECT x FROM table WHERE критерій1

якщо нам потрібно буде добавити ще один критерій ми до існуючого скрипта додаємо WHERE критерій2. Результат запиту буде відповідати одночасно і критерію1 і критерію2. Відбувається логічне множення критеріїв (операція and).

На випадок коли потрібно отримати результат запиту що відповідає хоча б одному критерію: критерій1 або критерій2 (операція or) приходить на допомогу Predicate Builder. Це бібліотека, що дозволяє будувати складні предикати та виконувати селекти з різноманітною фільтрацією.

Практика використання Predicate Builder особливо популярна у .NET у зв’язці з LINQ to entity. LINQ – це ще один потужний інструмент, що дозволяє будувати різноманітні запити до бази даних безпосередньо у коді C# і виглядає це як робота зі звичайною колекцією хоча насправді написаний нами код транслюється у команду SQL.

Так виглядає сервіс що відфільтровує та пагінує результат запиту згідно того як цей запит налаштовано:

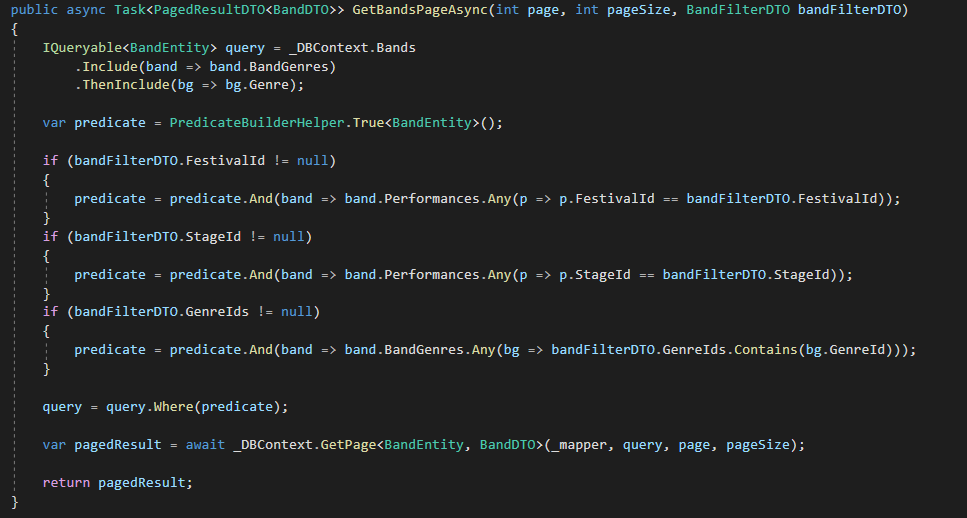


Рис 4.1. Метод обробки запиту на представлення сторінки гуртів відфільтрованих за певними ознаками

# **Розділ 5 Життєвий цикл запиту**

## ***5.1.*** ***Конвеєр обробки запиту***

На прикладі одного запиту я спробую описати весь принцип роботи веб аплікації. Хорошим та найбільш повним буде приклад запиту на створення нового гурту.

Як буде виглядати такий запит:

curl -X POST "https://localhost:44319/api/band"

-H "accept: \*/\*" -H "Authorization: bearer some.jwt.token"

-H "Content-Type: application/json"

-d "{ \"name\":\"BandName\", \"description\":\"BandDescription\", \"photoURL\":\"www.somePhoto.com\",\"genreIds\":[1,2]}"

Перш за все це POST запит що відправляється за шляхом https://localhost:44319/api/band. Кожен запит потрапивши на сервер проходить по конвеєру обробки запитів. Першим його зустрічає Routing module який визначає який саме endpoint буде обробляти цей запит. На цьому етапі користувач вже може отримати відповідь. Якщо Routing module не знайде відповідну точку входу то користувачу повернеться відповідь з відповідним повідовленням.

Наступним йде Authentication module який з’ясовує яка у користувача роль у системі і чи має запит право йти далі. Власне тут і перевіряється надісланий користувачем JWT токен. Тут користувач може отримати повідомлення про відсутність прав.

Далі потрібно змапити, тобто перетворити json об’єкт, що прийшов у тілі запиту, у внутрішню модель рівня Presentation layer – CreateBand. На цьому етапі проводиться базова валідація моделі. Група не може існувати без імені, але це ім’я не може містити більше 100 символі бо не влізе у базу даних. Також на цьому етапі ми не можемо перевірити чи присутня група з таким іменем у базі даних бо Presentation layer не має доступу до неї.

У разі успішної валідації модель мапиться у Data Transfer Object та потрапляє на наступний рівень. Тут ми вже маємо доступ до таблиць бази даних та можемо перевірити наявність жанрів з id 1 та 2 та чи була вже створена група з таким іменем. Якщо всі перевірки пройшли успішно то в таблицю Bands робиться новий запис, а стан бази даних фіксується. Ця операція має транзитивний характер. Далі ця модель дістається з бази даних та повертається на вищі рівні, попутно перетворюючись у DTO та ViewModel, і тоді користувач отримає створений ним гурт та код 200.

## ***5.2. IoC Контейнер***

Варто зауважити, що об’єкти класів які знаходяться далі в конвеєрі обробки запитів створюються лише тоді коли в них з’являється необхідність. Забезпечує це Inversion of Control контейнер (контейнер інверсії управління) що створює потрібні класи в момент їх необхідності. Відбувається це через механізм dependency injection (впровадження залежностей).

Налаштовуючи Ioc контейнер можна вибирати принцип створення об’єктів. Наприклад клас глобальних налаштувань навряд чи буде змінюватися в процесі роботи програми, тому його можна створити один раз і впроваджувати в інші класи по мірі необхідності, але клас для створення групи навряд чи потрібен постійно, тому створювати його необхідно лише коли прийшов запит на створення групи.

Реєстрація власних сервісів відбувається за допомогою механізму extensions C#:

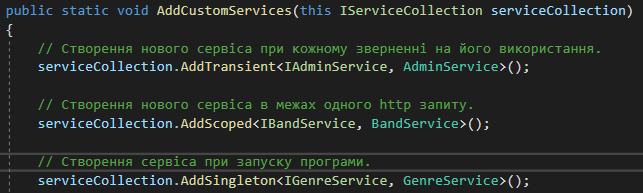


Рис. 5.1. Приклад реєстрації власних сервісів в IoC контейнері

# **Використані технології**

*Git* – найважливіша складова написання будь якого програмного забезпечення це система контролю версій. Вона дозволяє зберігати зроблений прогрес на віддаленому сервері без страху втратити написаний код. Також git дозволяє швидко відкотити зроблені зміни якщо вони виявилися не робочими або містять іншу помилку.

*Trello* – організація проекту такого масштабу є не простим завданням. На допомогу приходить Trello де можна розбити процес розробки на окремі завдання та виставляючи порядок їх виконання. Не потрібно тримати все в голові. Це допомагає на стадії проектування. Моя дошка складається з колонок to do, in progress, done а також окрема колонка для багів які були виявлені в процесі тестування.

*Microsoft Visual Studio 2019* – потужне IDE для розробки програмних продуктів на базі .NET. Версія 2019 є необхідною для використання .NET Core 3.0.

*ASP.NET Core 3.1* – каркас, фундамент, фреймворк на якому була виконана робота. Потужний та легкий у використанні інструмент для створення веб додатків.

*Entity Framework Core* – ORM технологія розроблена Microsoft. Дозволяє робити відображення бази даних з її таблицями і користуватися ними наче звичайними колекціями C#. У деякій мірі це є реалізацією патерну репозиторій який розміщений на Data access layer.

*LINQ to entity* – проміжна мова запитів дозволяє будувати схожі до SQL скриптів команди які відправившись до сервера бази даних перетворюються у SQL команду, якщо її неможливо виконати то повернуться усі необхідні дані для того щоб виконати її вже на стороні програми засобами C#.

*Swagger* – це програмне забезпечення що допомагає в процесі розробки імітуючи програму – клієнта. На основі написаних моделей та точок входу swagger допомагає будувати http запити та тестувати поведінку програми.

*NLog* – ця бібліотека дозволяє здійснювати логування процесу роботи програми. Особливістю логера є те що він запускається швидше ніж сама програма, дозволяючи записати помилку запуску, якщо така сталася.

*AutoMapper* – бібліотека, що дозволяє перетворювати об’єкти одного типу в об’єкти іншого типу тим самим створюючи мости передачі даних з одного рівня на інший.

*Migrations* – є компонентом Entity Framework дозволяючи будувати базу даних автоматично при першому запуску програми, спрощуючи роботу програміста і нівелюючи ймовірність зробити помилку в процесі створення бази даних.

*Docker* – коли вся робота була зроблена було вирішено загорнути усю програму в docker – контейнер. Це дозволить передати готовий програмний продукт замовнику без остраху що його середовище не підготовлене для запуску програми. Лише однією командою docker-compose up буде створена уся необхідна інфраструктура на якій і запуститься аплікація готова для виконання роботи.

# **Висновки**

У даній курсовій роботі розглянуто на застосовано на практиці сучасні принципи створення Web API застосунків з використанням стеку технологій .NET.

Грамотне планування та використання напрацювань досвідчених програмістів дозволили написати структуровану, відкриту для розширення програму з великим обсягом функціоналу.

Посилання на відкритий код можна знайти за цим посиланням: https://github.com/Rostik18/Spring-Fest

Посилання на контейнеризовану версію розміщене за цим посиланням:   
<https://hub.docker.com/repository/docker/rostyslavbaitsar/spring-fest>

# **Джерела та література**

1. Pro C# 7 - With .NET and .NET Core, Andrew Troelsen, Eighth Edition.
2. CLR via C#, Jeffrey Richter, fourth edition.
3. Розробка Web API [Електронний ресурс] – режим доступу URL: <https://habr.com/ru/post/181988/>
4. Гід ASP.NET Core [Електронний ресурс] – режим доступу URL: <https://habr.com/ru/post/312226/>
5. IoC, DI, IoC-контейнер [Електронний ресурс] – режим доступу URL: <https://habr.com/ru/post/131993/>
6. Onion архітектура [Електронний ресурс] – режим доступу URL: <https://habr.com/ru/post/233747/>
7. Entity Framework Core [Електронний ресурс] – режим доступу URL: <https://habr.com/ru/company/otus/blog/500012/>
8. Офіційна документація Майкрософт [Електронний ресурс] – режим доступу URL : <https://docs.microsoft.com/ru-ru/aspnet/core/web-api/?view=aspnetcore-3.1>