**A logo with black text

Description automatically generatedТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

Факултет Приложна математика и информатика

**ДИПЛОМНА РАБОТА**

Тема: Уеб базирана система за управление на манга колекции

|  |  |
| --- | --- |
| *Дипломант:* Кристиян Петров Кръчмаров |  |

Фак. №: 181220017

*Специалност:* Приложна Математика и Информатика

*Образователно- квалификационна степен*: **бакалавър**

Дипломен ръководител: доц. д-р Анна Розева

София 2024

**A logo with black text

Description automatically generatedТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

Факултет Приложна математика и информатика

**Утвърждавам:**

**Декан на ФПМИ:**

**(проф. д-р инж. Д. Иванова)**

Дата на задаване: 09.04.2024

***ЗАДАНИЕ***

за разработка на дипломна работа

|  |  |
| --- | --- |
| *Дипломант:* Кристиян Петров Кръчмаров | Фак. №: 181220017 |

*Специалност:* Приложна Математика и Информатика

*Образователно- квалификационна степен*: **бакалавър**

1. *Тема на дипломната работа:*

|  |
| --- |
| **Уеб базирана система за управление на манга колекции** |

1. *Заявител на темата:* Катедра Информатика - ФПМИ
2. *Обяснителна записка :*

|  |
| --- |
| Уеб базираната система за управление на манга колекции и поръчки ще бъде разработена с технологиите .NET, Blazor, HTTP, REST, JWT. Ще бъде създадена база с манга заглавия в PostgreSQL, от която потребителите да могат да избират и добавят към своите колекции, да осъществяват нейната поддръжка, както и да въвеждат информация за направени поръчки. Приложението ще генерира справки отностно колекцията и поръчките на потребителите. |

Дипломен ръководител: Ръководител катедра:

|  |  |
| --- | --- |
| доц. д-р Анна Розева | доц. д-р Златко Захариев |

Съдържание

[2 Списък на фигурите 2](#_Toc168150709)

[3 Списък на таблиците 2](#_Toc168150710)

[4 Съкращения 3](#_Toc168150711)

[5 Абстракт 4](#_Toc168150712)

[6 Използвани технологии 5](#_Toc168150713)

[6.1 PostgreSQL 5](#_Toc168150714)

[6.2 ASP.NET Web API 5](#_Toc168150715)

[6.2.1 Entity Framework Core 5](#_Toc168150716)

[6.2.2 AutoMapper 5](#_Toc168150717)

[6.2.3 AspNetCore Identity 5](#_Toc168150718)

[6.3 Redis 5](#_Toc168150719)

[6.3.1 AspNetCore.OutputCaching.StackExchangeRedis 6](#_Toc168150720)

[6.4 Json Web Token 6](#_Toc168150721)

[6.4.1 AspNetCore.Authentication.JwtBearer 7](#_Toc168150722)

[6.5 Swagger 7](#_Toc168150723)

[6.6 REST 7](#_Toc168150724)

[6.7 Blazor 7](#_Toc168150725)

[6.7.1 MudBlazor 8](#_Toc168150726)

[6.7.2 Java Script interoperability (JSinterop) 8](#_Toc168150727)

[6.8 Google Charts 8](#_Toc168150728)

[6.9 DBeaver 8](#_Toc168150729)

[6.10 Rider 8](#_Toc168150730)

[7 Архитектура и реализация на информационната система 9](#_Toc168150731)

[7.1 База данни 9](#_Toc168150732)

[7.1.1 LibraryMangas и Authors 11](#_Toc168150733)

[7.1.2 UserMangas 12](#_Toc168150734)

[7.1.3 Orders 13](#_Toc168150735)

[7.1.4 Таблици, свързани с потребителска информация 13](#_Toc168150736)

[7.1.5 Първоначални данни за LibraryMangas 13](#_Toc168150737)

[7.2 Сървърна част 15](#_Toc168150738)

[7.2.1 Repository 16](#_Toc168150739)

[7.2.2 Service 16](#_Toc168150740)

[7.2.3 Endpoint 16](#_Toc168150741)

[7.3 Мениджмънт на потребители 19](#_Toc168150742)

[7.3.1 Жетон за достъп 20](#_Toc168150743)

[7.3.2 Опреснителен жетон 22](#_Toc168150744)

[7.4 Потребителски интерфейс 23](#_Toc168150745)

[7.5 CORS 23](#_Toc168150746)

[8 Използвана литература 24](#_Toc168150747)

# Списък на фигурите

[Фигура 1 Подписване на JWT 6](#_Toc168150694)

[Фигура 2 Примерен JWT 6](#_Toc168150695)

[Фигура 3 Структура на DbContext 9](#_Toc168150696)

[Фигура 4 Схема на базата данни 10](#_Toc168150697)

[Фигура 5: Прочитане на CSV файл 13](#_Toc168150698)

[Фигура 6: Прочитане на нужните данни 13](#_Toc168150699)

[Фигура 7 Проверка за съществуващи автори 14](#_Toc168150700)

[Фигура 8 Архитектура на сървърната част 15](#_Toc168150701)

[Фигура 9 Пример за Minimal API 15](#_Toc168150702)

[Фигура 10 Списък с Claim обекти 19](#_Toc168150703)

[Фигура 11: Генериране на JWT 20](#_Toc168150704)

[Фигура 12: Декодиране на JWT 21](#_Toc168150705)

[Фигура 13: Конфигурация за проверка на JWT 21](#_Toc168150706)

[Фигура 14: Генериране на опреснителен жетон 22](#_Toc168150707)

[Фигура 15: Настройване на CORS политика 23](#_Toc168150708)

# Списък на таблиците

[Таблица 1: Операции за модела LibraryManga 16](#_Toc168150689)

[Таблица 2: Операции за модела UserManga 17](#_Toc168150690)

[Таблица 3: Операции за модела Order 17](#_Toc168150691)

[Таблица 4:Операции за статистиките 17](#_Toc168150692)

[Таблица 5: Операции за оторизация на потребител 18](#_Toc168150693)

# Съкращения

REST

JWT

API

ACID

HTTP

ORM

DOM

HTML

SVG

CSV

IDE

UI

CORS

# Абстракт

С нарастващата популярност на Японската култура в световен мащаб и мангата като един от най-емблематичните ѝ представители, все повече хора започват да четат, събират и колекционират различни заглавия. Мангата е форма на комикс, който обхваща широк спектър от жанрове и тематики, привличайки много хора със своя стил и културна значимост. Въпреки това, управлението на колекции от манга може да бъде предизвикателно и да отнема много време.

Необходимостта от информационна система, която да улеснява управлението на колекции и поръчки е нараснала значително, с технологическия напредък и дигитализацията на повечето аспекти от нашия живот. Основни проблеми пред хората са: загуба на данни, дублиране на купени заглавия и трудности с проследяването на поръчките. Съществуват методи за управление, като ръчни записи и електронни таблици, но не предоставят достатъчна ефективност и удобство.

Целите на приложението са да предостави на потребителите централизирана система за управление на своите колекции и поръчки, както и да предложи визуализация за колекцията и за похарчените средства.

Основните функционалности ще бъдат:

* Набор от готови заглавия: Потребителите ще имат достъп до готов списък от заглавия, както и да виждат информация за всяко произведение.
* Създаване на дигитална колекция: Потребителите ще могат да избират от предоставените заглавия и да добавят към своята колекция, като въвеждат информация, свързана с тяхното отношение към творбата.
* Регистрация на поръчки: Потребителите ще могат да добавят и редактират информация за поръчките си.
* Визуализация на данни: Приложението ще генерира графики и статистки, които ще помагат на потребителя за анализира своята колекция и своите поръчки, както и да има по-добро разбиране за своите разходи.

# Използвани технологии

## PostgreSQL

PostgreSQL [1], или по честно наричана Postgres е релационна база данни с отворен код. Postgres позволява на потребителите да създават собствени типове данни, под формата на обекти, върху които могат да бъдат прилагани функции като наследяване и полиморфизъм. Също така се поддържат транзакции с ACID свойства и поддържка на други езици, освен SQL.

## ASP.NET Web API

ASP.NET Web API [2] е framework който позволява лесно създаване на уеб приложения. Той е част от ASP.NET Core платформата и предоставя голям брой библиотеки за разработването на REST базирани уеб услуги.

### Entity Framework Core

Entity Framework Code (EF Core) [3] е библиотека, която позволява връзката и интеракцията между базата данни и проекта. Библиотеката работи на ORM принципа, при който записи от таблица в базата биват превърнати в обект, който може да се използва от обектно ориентиран език.

### AutoMapper

AutoMapper [4] е библиотека която позволява лесното преобразуване от един обект в друг.

### AspNetCore Identity

AspNetCore Identity [5] позволява мениджмънт и съхраняване на потребителски акаунти, възможните роли и възможните права, които може един потребител да притежава.

## Redis

Redis [6] e нерелационна база данни, която работи на принципа „ключ-стойност“. Най честото ѝ предназначение е за кеширане. Дизайнът на Redis позволява ниско латентни операции, защото информацията се съхранява на паметта, а не на диска.

### AspNetCore.OutputCaching.StackExchangeRedis

AspNetCore.OutputCaching.StackExchangeRedis [7] e библиотека, която имплементира кеширане, чрез Redis. Output Caching е междинен слой в приложението, който позволява кеширането на заявката, която ще бъде подадена към сървъра, както и отговора, който трябва да се върне, във двойка „ключ-стойност“.

## Json Web Token

Json Web Token (JWT) [8] е стандарт за размяна на информация между две страни в JSON формат. Самият жетон (token) има три части, който са кодирани в Base 64 формат, разделени с точка. Неговите части са Заглавен ред (Header), Тяло (Payload) и Подпис (Signature).

В заглавния ред се съдържа информация за типа токън и алгоритъма, с който е подписан. Най често се използва алгоритъма HS256. Тялото съдържа информацията, която ще бъде изпратена, както и някои стандартни полета, като идентификатор, издател, кога е издаден и други.

Подписа се генерира като неподписания токън, който представлява комбинация от кодирания заглавен ред и кодираното тяло разделени с точка, се криптира с таен ключ и избрания алгоритъм.

A screenshot of a computer code

Description automatically generated

Фигура 1 Подписване на JWT

Крайният резултат е трите кодирани части, разделени с точка.

A close up of a computer code

Description automatically generated

Фигура 2 Примерен JWT

За проверка на валидността е необходимо да се провери само подписа. Чрез единичната проверка на подписа позволява на JWT да бъде самостоятелна структура, която съдържа в себе си информация и метода за нейното криптиране.

### AspNetCore.Authentication.JwtBearer

AspNetCore.Authentication.JwtBearer [9] е библиотека, в която има имплементация за валидиране и създаване на JWT както и позволява да се използва оторизация, базирана на JWT, посредством междинен слой.

## Swagger

Swagger [10] е комплект от инструменти, които позволяват за лесно документиране, чрез OpenAPI [11] стандарта. Swagger предоставя визуален интерфейс, чрез който може да се види различните операции, които едно приложение поддържа, необходимите данни за заявка, необходимост от оторизация и формат на крайния резултат.

## REST

REST [12] е архитектурен стил за реализиране на уеб услуги между клиент и сървър, които си комуникират с HTTP заявки. Заявките носят в себе си следните компоненти: HTTP метод, заглавни редове (Headers) с информация свързана за оторизиране, кеширане и други мета данни и тяло (Body) в което се съдържа информацията. Отговорите имат във себе си статус код, който показва какво се е случило, Headers, в които има информация за сървъра изпълняващ заявката и тяло, в което се съдържа информацията, в повечето случаи в JSON формат

## Blazor

Blazor [13] e framework, който позволява създаването на интерактивен потребителски интерфейс, базиран на компонентния модел, но написани на езика C#, а не JavaScript (JS). Възможни са 2 вида хостинг модели: Blazor Web Assembly (WASM), където кодът се компилира и изпълнява в браузъра и Blazor Server, където всички операции се изпълняват на сървъра и биват комуникирани към клиента чрез SignalR [14], библиотека позволяваща асинхронно изпращане на информация към клиента.

### MudBlazor

MudBlazor [15] е библиотека с готови Razor компоненти, която улеснява създаването и разбирането на елементите от потребителския интерфейс.

### Java Script interoperability (JSinterop)

JSinterop [16] e функционалност, която позволява на сървъра да изпълни JavaScript код, както и JS кодът да изпълни C# код, когато е необходимо, Такива случаи са когато трябва да се достъпи DOM-a самия браузър или други софтуери от трети страни.

## Google Charts

Google Charts [17] е библиотека за генериране на различни видове диаграми в HTML или SVG формат, чрез JS код.

## DBeaver

DBeaver [18] е инструмент с отворен код, който позволява лесна и визуална интеракция с база данни, независимо от това какъв е нейният тип. Позволява експорт или импорт в различни файлови формати като CSV, XML, JSON.

## Rider

Rider [19] е интегрирана среда за разработка (IDE) на компанията JetBrains, която е предназначена за разработване на софтуер чрез .NET платформата, в която се използват езици като C# и F#, но и други популярни езици като JavaScript и TypeScript.

# Архитектура и реализация на информационната система

Архитектурата се състои от три компонента: база данни, сървър и потребителски интерфейс. В базата се съдържат всички данни, които се използват из приложението, като възможните заглавия, информация за потребителите и техните колекции и поръчки. Сървърната част е предназначена за обработка на заявки от потребителите, изпълнение на бизнес логика, валидация и достъпване на данните от базата. Потребителския интерфейс служи за представяне на информация и взаимодействие със сървъра и неговите функционалности.

## База данни

Базата е генерирана чрез подхода „Първо код“ (Code first), в който първо се създават модели на отделните таблици и релациите между тях. Чрез този метод се позволява лесна поддръжка и разширение на базата в бъдеще.

За да се реализира този подход е нужно да се създаде клас, който да наследи класа DbContext от EF Core. Този клас е отговорен за връзката с базата, както и за управлението на различните таблици в нея.

A screen shot of a computer code

Description automatically generated

Фигура 3 Структура на DbContext

Всяка една променлива от тип DbSet<T> представя таблица от базата. За реализация на функционалността са нужни 4 основни таблици.

* LibraryMangas – в тази таблица ще се съхранява обща информация за възможните произведения от които може да се избира
* UserMangas – таблица с информацията различните произведения, които потребителите са добавили в своите колекции
* Orders – информация за поръчките на потребителите
* Users – информация за самите потребителите, като имейл адрес и парола

Поради наличността на потребители в системата, ще използваме IdentityDbContext, чрез който ще бъдат генерирани таблици, свързани с информацията на потребителя. След като моделът на базата е готов трябва да се създаде „миграция“. Миграцията е генериран клас, в който се съдържат всички операции, които трябва да се изпълнят върху базата. За създаването ѝ се използва командата „dotnet ef migrations add name“, където name е името на миграцията. За първа миграция е препоръчително нейното име да е Initial. След създаването ѝ, миграцията може да бъде приложена върху базата чрез командата „dotnet ef database update“.

A computer screen with many black and white text

Description automatically generated with medium confidence

Фигура 4 Схема на базата данни

На Фиг.4 е показана схемата на базата, която получихме след прилагането на миграцията.

В следващите параграфи е обяснено по-подробно каква е структурата на основните таблици

### LibraryMangas и Authors

Таблицата LibraryMangas има следните колони

* Id – идентификатор, който служи за първичен ключ (primary key)
* TitleRomaji[[1]](#footnote-1) – заглавието на латиница. На повечето места в системата това поле ще бъде основното заглавие.
* TitleEnglish – заглавието на английски. Възможно е заглавието да го няма, защото творбата може да не е преведена.
* TitleJapanese – заглавието на японски
* DemographicType – демографска група, към която е насочена творбата. Възможните групи са:
  + Shounen –за момчета до 18 години
  + Shoujo –за момичета до 18 години
  + Seinen –за мъже над 18 години
  + Josei –за жени над 18 години
* Type – тип на творбата. Възможни са 3 стойности
  + Мanga – манга
  + LightNovell – новела
  + OneShot – самостоятелна история в един том
* PublishingStatus – статус на публикуване с 3 възможни стойности
  + Publishing – все още се публикува
  + Finished – публикуването е приключило
  + OnHiatus –публикуването е спряно временно
* TotalVolumes – обща бройка томове. Възможно е бройката да я няма, защото творбата се публикува.
* MainImageURL – Линк към снимка на първия том. Възможно е да няма линк, защото творбата да няма все още издаден първи том.
* Synopsis – Описание

Таблицата Authors е предназначена за авторите и информацията за тях. Колоните са

* Id – идентификатор, първичен ключ
* FirstName – Първо име на автора. В някои случаи това име може да го няма, защото автора използва псевдоним.
* LastName – Фамилно име на автора. В тази колона се записа псевдонима на автора, ако е използван такъв
* Role – неговата роля в творбата. Възможни са 3 стойности в тази колона
  + Story – автора е отговорен само за историята
  + Art – автора е отговорен само за рисуването и арт стила
  + StoryAndArt – aвтора отговаря и за двете

Между тези две таблици съществува релация от вида много към много (many to many), осъществена чрез междинната таблица AuthorLibraryManga

### UserMangas

В тази таблица се съдържа информация за взаимоотношението между дадена творба и даден потребител. Съдържат се следните колони

* Id – идентификатор, първичен ключ
* ReadingStatus – статус на четене с възможни стойности
  + Reading – все още произведението се чете
  + Finished – произведението е прочетено от потребителя
  + OnHold – потребителя е спрял временно да го чете
  + Dropped – потребителя се е отказал да дочете произведението
  + PlanToRead – потребителя планира да чете произведението в бъдещето
* CollectionStatus – статус на колекциониране. Възможни са стойностите
  + Collected – творбата е събрана
  + InProgress – творбата е в процес на събиране
  + PlanToCollect – потребителя планира да събере творбата в бъдеще
* ReadVolumes – брой прочетени томове
* CollectedVolumes брой събрани томове
* PricePerVolume – цена за един том
* UserId – идентификатор на потребителя, към който е асоциирана информацията, външен ключ (foreign key)
* LibraryMangaId идентификатор на творбата, за която се отнася потребителската информация; външен ключ

### Orders

Таблицата Orders е предназначена за информацията, относно поръчките на потребителите. Нейните колони са

* Id – идентификатор, първичен ключ
* Date – дата на поръчване
* Status – статус на поръчката, с възможни стойности
  + Created – поръчката е създадена
  + OnTheWay – поръчката е в процес на доставяне
  + Delivered - доставена
* Description – описание за закупените артикули
* Place – място на направа на поръчката
* Amount – стойност на поръчката
* NumberOfItems – брой артикули, които са закупени
* UserId – идентификатор на потребителя; външен ключ

### Таблици, свързани с потребителска информация

Защото използвахме IdentityDbContext, са генерирани множество таблици които са за мениджмънт на потребителската информация. За целите на нашия проект ние ще използваме само две таблици: AspNetUsers и AspNetRoles. В таблицата AspNetUsers се намира основната информация за потребителя като потребителско име, имейл адрес и парола, която е хеширана. В таблицата AspNetRoles се намира информация за възможните роли в приложението. Между тези две таблици има релация много към много, осъществена чрез таблицата AspNetUserRoles.

### Първоначални данни за LibraryMangas

За да се реализира готовият набор от заглавия, които потребителите ще могат да избират, ще използваме следния набор от данни (data set) от Kaggle. [20]

Данните са организирани в CSV формат, от който може да бъде извлечена необходимата за приложението информация. Но поради големия брой колони и повторяемостта на някои данни е необходимо данните да бъдат преструктурирани.

За тази цел е написана малка програма която помага за генерирането на 3 отделни CSV файла, в които се съдържа информация за различните манга заглавия, авторите и тяхната релация. За целта са използвани двата класа CSVReader и CSVWritter, които позволяват лесна интеракция с файлове от този тип.

A close-up of a computer screen

Description automatically generated

Фигура 5: Прочитане на CSV файл

С метода Read започва четенето на файла, а с ReadHeader се прочита заглавния ред, който носи в себе си имената на отделните колони във файла. Чрез използването на ReadHeader метода се позволява търсене на данни чрез името на колоната

A screen shot of a computer

Description automatically generated

Фигура 6: Прочитане на нужните данни

След форматиране на прочетените данни е възможно да се състави обект, който да е във формата на таблицата LibraryMangas. За авторите подхода е малко по различен, тъй като информацията за авторите е записана в JSON формат и е необходима допълнителна стъпка,в която трябва да се прочете списъкът от обекти и да се преобразуват в обекти от типа Author. За да се избегне повторяемостта на авторите в базата е направена следната проверка.

A computer screen shot of text

Description automatically generated

Фигура 7 Проверка за съществуващи автори

Проверката представлява търсене в списъка с досега намерените автори дали вече съществува такъв автор. Ако не съществува такъв, той бива добавен към списъка с автори и се добавя релация между него и текущото произведение. Ако съществува се създава само релацията.

След това чрез функцията WriteRecords на CSVWriter класа е възможно записването на трите файла.

След генерирането на трите файла е възможно те да бъдат внесени в базата чрез DBeaver.

## Сървърна част

Сървърната част е реализирана с фреймуърк-а ASP.NET Web API и езика C#. Тази част е реализирана като независим от потребителския интерфейс компонент.

A diagram of service level

Description automatically generated

Фигура 8 Архитектура на сървърната част

На фиг 8 е показана архитектурата на сървърната част. То има в себе си 3 основни нива: Repository, Service, Endpoint.

### Repository

Repository [21] е шаблон за дизайн (design pattern) в който се прави абстракция на източника на данни, чрез предварително дефинирани операции. Чрез имплементацията на този шаблон се позволява работа с обекти, вместо с SQL скриптове или друг вид операции и се изолира от бизнес логиката. Този дизайн има три основни компонента: модел, дефиниция и имплементация. Модела представлява самите данни за приложението. Дефиницията е интерфейс в който са дефинирани различните методи (например добавяне и изтриване от базата) а в имплементацията се предоставя конкретната логика на дефинираната в интерфейса функционалност.

### Service

Service нивото служи като „мост“ между Repository нивото и потребителя, като изпълнява функциите, предоставени от Repository-то, но може да обработи допълнително предоставената информацията, като да приложи валидации, да извършва преобразувания между обекти. В това ниво се случва и генерирането на информация, като създаването на токъните, които се предоставят на потребителя.

### Endpoint

Endpoint нивото представлява интерфейса за комуникация, чрез който ще се приемат заявки и информация. Той е реализиран чрез Minimal API подхода. Minimal API [22] е опростен начин за изграждане на крайни точки (endpoints).

A computer screen shot of a computer code

Description automatically generated

Фигура 9 Пример за Minimal API

На фиг.9 е показан пример за endpoint, който връща текста „Hello World“, когато се извика главния път на сървъра (root).

За реализация на функциите е необходимо да се обработват заявки от различни видове. Вида на заявката се определя по HTTP метода. Ще се използват 4 основни метода

* GET – с този метод показваме, че искаме да вземем данни от сървъра.
* POST – използва се за изпращане на данни към сървъра.
* PUT – използва се за актуализация на съществуващ ресурс.
* DELETE – с този метод показваме, че искаме да изтрием конкретен ресурс.

Тъй като използваме HTTP протокола, е възможно на един път да се поставят няколко операции. В зависимост от модела са реализирани следните интерфейси за комуникация.

За предоставените заглавия са реализирани методите за комуникация показани в Таблица 1

Таблица 1: Операции за модела LibraryManga

|  |  |  |  |
| --- | --- | --- | --- |
| Път | Метод | Параметри | Описание |
| /libraryManga | POST | обект | Създава запис в базата. |
| /libraryManga | GET | id | Показва информацията за записа. |
| /libraryManga | PUT | id, обект | Променя запис в базата с предоставения обект. |
| /libraryManga | DELETE | id | Изтрива записа от базата. |
| /libraryManga**s** | GET | pageIndex, entriesCount | Показва информация за заглавия с размер entriesCount, които трябва да се покажат на страница с номер pageIndex. |
| /libraryManga/search | GET | title | Показва всички творби, които съдържат в заглавието си title. |

Поради наличието на много заглавие е имплементирано кеширане на за всички GET операции на модела LibraryManga в Redis чрез OutputCaching слоя. При постъпване на заявка, която модифицира данните в базата (POST, PUT, DELETE), кешираните данни ще бъдат премахнати, тъй като няма да са актуални, след постъпилата промяна. Кеширането се осъществява чрез CacheOutput метод, който се закача за MapGet метода, чийто резултат искаме да кешираме. В CacheOutput метода се конфигурира колко време ще бъде кеширан резултата.

За потребителските заглавия са необходими следните операции, описани в Таблица 2.

Таблица 2: Операции за модела UserManga

|  |  |  |  |
| --- | --- | --- | --- |
| Път | Метод | Параметри | Описание |
| /userManga | POST | обект | Създаване в базата. |
| /userManga | GET | user\_id | Показва информацията за всички записи на потребителя |
| /userManga | GET | id | Показва информацията за конкретен записа. |
| /userManga | PUT | id, обект | Променя записа в базата с предоставения обект. |
| /userManga | DELETE | id | Изтрива запис по подадено id |

За потребителските поръчки са реализирани операциите, показани в Таблица 3.

Таблица 3: Операции за модела Order

|  |  |  |  |
| --- | --- | --- | --- |
| Път | Метод | Параметри | Описание |
| /order | POST | обект | Създаване в базата. |
| /order | GET | id | Показва информация за конкретен запис. |
| /order | GET | user\_id | Показва информация за всички записи на потребителя. |
| /order | PUT | id, обект | Променя записа в базата с предоставения обект. |
| /order | DELETE | id | Изтрива запис по подадено id. |

В Таблица 4 са показани възможните функционалности за генериране на статистика за колекцията и поръчките на потребителя. Общо за всяка една операция е метода на изпълнение, който е GET и необходимия параметър за изпълнение на заявката, който е идентификатора на потребителя (user\_id).

Таблица 4:Операции за статистиките

|  |  |
| --- | --- |
| Път | Описание |
| /statistics/userManga/demographic | Предоставя информация за броя на произведенията спрямо различните демографии. |
| /statistics/userManga/type | Показва информация за колекцията спрямо различните типове творби. |
| /statistics/userManga/publishingStatus | Показва информация за колекцията спрямо различните статуси за публикуване. |
| /statistics/userManga/readingStatus | Предоставя информация за броя на произведенията спрямо различните статуси за четене. |
| /statistics/userManga/collectionStatus | Показва информация за колекцията спрямо различните статуси за събиране. |
| /statistics/userManga/totalSpending | Показва информацията за похарчените средства за всяко едно заглавие. |
| /statistics/userManga/general | Предоставя обща информация за потребителската колекция. |
| /statistics/order/year | Показва информация за броя поръчки, направени през годините. |
| /statistics/order/place | Показва информация за всички поръчки направени в различните места. |

Операциите, които са свързани с мениджмънта на потребители са описани в Таблица 5.

Таблица 5: Операции за оторизация на потребител

|  |  |  |  |
| --- | --- | --- | --- |
| Път | Метод | Параметри | Описание |
| /user/register | POST | обект | Регистрира се потребител. |
| /user/login | POST | обект | Вписва потребителя. |
| /user/refresh | POST | обект | Опреснява достъпа. |
| /user/revoke | DELETE |  | Премахва текущия достъп до системата. |

## Мениджмънт на потребители

За целите на приложението е реализиран механизъм за удостоверяване, базиран на жетони (Token based authentication) [23]. Това е метод за проверка на идентичността на потребителя. Потребителя въвежда своите данни за идентификация, като потребителско име и парола. Тези данни биват проверени от сървъра към който се иска достъп. Ако са правилни се генерира жетон, чрез който потребителя може, за период от време да достъпва функционалности на сървъра. Този жетон се връща на потребителя и се съхранява в браузъра.

За управление на достъпа до различните функции е реализиран контрол на достъпа, чрез роли (Role-Based Access Control) [24]. С този подход на всеки потребител бива дадена една или повече роли, като всяка роля има предварително дефинирани разрешения за достъп до ресурси и извършване на действия. В системата има две роли.

* User – роля, която се прилага на всеки потребителски профил в системата.
* Admin – администраторска роля, която позволява допълнителни операции, недостъпни за стандартния потребител.

Информацията за това дали един потребител е администратор се задава в когато той се регистрира.

За методите, които се нуждаят от оторизация се използва RequireAuthorization метод, който се закача след необходимите „Map“ методи (MapPost, MapGet и тн), в който се задава каква роля е необходима за изпълнението ѝ. За функционалности, които не се нуждаят от оторизация, за да бъдат изпълнени се използва метода AllowAnonymous, с който се позволява функцията да бъде извикана от всеки който има достъп, независимо дали е регистриран потребител или не.

Когато потребителя се вписва в системата се генерират 2 жетона. За достъп и за опресняване. Те имат време за живот, в което могат да бъдат използвани. Жетонът за достъп трябва да е със сравнително кратно време на живот, докато опреснителният трябва да е с голям живот. Добра практика е жетоните за достъп да са с време на живот между 30 секунди и 5 минути, докато опреснителният да е с време между 14 и 30 дни.

### Жетон за достъп

Жетонът за достъп (access token) е в JWT формат и се съдържа информация за потребителя, неговите роли и времето за живот. За неговото създаване е използвани класовете SecurityTokenDescriptor и JwtTokenHandler. Данните които ще запишем в тялото на JWT жетона трябва да списък от тип Claim, който е вграден тип в ASP.NET

A computer screen shot of text

Description automatically generated

Фигура 10 Списък с Claim обекти

В списъкът се съдържат обекти в които се носи информация за самия потребител като неговото потребителско име, идентификатора му и ролите които са приложими за него, както и поле за идентификатор на жетона. След генерирането на този списък се създава SecurityTokenDescriptor обект, в който се описват всички необходими стандартни полета.

A computer screen shot of code

Description automatically generated

Фигура 11: Генериране на JWT

Данните за потребителя се поставят в Subject полето. Стандартните полета които трябва да се дефинират са

* Issuer (iss) – кой е създал и подписал жетона
* Audience (aud) – кой е предназначения получател
* IssuedAt (iat) – кога е издаден
* NotBefore(nbf) – време, след създаването, до настъпването на което жетона не е валиден

В обекта се добавят също времето на живот чрез Expires (exp) полето и ключът и алгоритъма за подписване. Метода CreateToken на JwtTokenHandler позволява да генерираме жетон, чрез SecurityTokenDescriptor обекта.

Генерираният жетон можем да декодиране с помощта на jwt.io [25], за да се разгледа неговото съдържание.

A screenshot of a computer code

Description automatically generated

Фигура 12: Декодиране на JWT

За да е възможна проверката е необходимо да се зададе какво трябва да се проверява. В ASP.NET това е възможно с AddJwtBearer метода.

A computer screen shot of text

Description automatically generated

Фигура 13: Конфигурация за проверка на JWT

След като жетонът за достъп изтече, потребителския интерфейс може да използва опреснителния жетон, за да генерира нов.

### Опреснителен жетон

Жетона за опресняване (refresh token) не носи в себе си информация и се запазва в базата заедно с времето на живот, за да е възможна проверка на неговата валидност. След изтичането на опреснителния жетон потребителят трябва да се впише отново в системата.

A screenshot of a computer program

Description automatically generated

Фигура 14: Генериране на опреснителен жетон

На фиг. 13 е показан как се създава опреснителен жетон. Чрез генератор на случайни числа се генерира масив от 64 числа в диапазона 0-255, които след това са конвертирани в Base64 стринг.

## Потребителски интерфейс

## CORS

Поради наличието на два различни сървъра, размяна на данни между тях и възможността тези два сървъра да са в два различни домейна се сблъскваме с проблем. Браузърите не позволяват на уеб страници да изпълняват заявки до други домейни, освен до този който е предоставил страницата. Този механизъм се нарича политика за същия произход (same origin policy). В случаите в които трябва да се правят заявки до източници от друг произход, трябва да се позволи, чрез споделяне на ресурси от други източници (Cross Origin Resource Sharing) [26]. Чрез CORS е възможно сървърът да обработва заявки от предварително дефинирани източници и да отхвърля останалите.

За да се позволи CORS е необходимо в сървърната част да се зададе на кой адрес се намира потребителския интерфейс. За тази цел ASP.NET предоставя методите AddCors, в който се настройва самата политика и UseCors, чрез който се активира конкретна CORS политика.

A computer screen shot of a code

Description automatically generated

Фигура 15: Настройване на CORS политика

# Използвана литература

|  |  |
| --- | --- |
| [1] | „PostgreSQL,“ PostgreSQL Global Development Group, [Онлайн]. Available: https://www.postgresql.org/. |
| [2] | „ASP.NET Web API,“ Microsoft, [Онлайн]. Available: https://dotnet.microsoft.com/en-us/apps/aspnet/apis. |
| [3] | „Entity Framework Core,“ Microsoft, [Онлайн]. Available: https://learn.microsoft.com/en-us/ef/core/. |
| [4] | „Automapper,“ .NET Foundation, [Онлайн]. Available: https://automapper.org/. |
| [5] | „ASP.NET Core Identity,“ Microsoft, [Онлайн]. Available: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio. |
| [6] | „Redis,“ Redis, [Онлайн]. Available: https://redis.io/. |
| [7] | „AspNetCore OutputCaching StackExchange Redis,“ Microsoft, [Онлайн]. Available: https://www.nuget.org/packages/Microsoft.AspNetCore.OutputCaching.StackExchangeRedis/8.0.3. |
| [8] | „JWT,“ [Онлайн]. Available: https://auth0.com/docs/secure/tokens/json-web-tokens. |
| [9] | „AspNetCore.Authentication.JWT.Bearer,“ Microsoft, [Онлайн]. Available: https://www.nuget.org/packages/Microsoft.AspNetCore.Authentication.JwtBearer. |
| [10] | „Swagger,“ SmartBear, [Онлайн]. Available: https://swagger.io/. |
| [11] | „ОpenAPI,“ [Онлайн]. Available: https://www.openapis.org/. |
| [12] | „RESTful API,“ [Онлайн]. Available: https://restfulapi.net/. |
| [13] | „Blazor,“ Microsoft, [Онлайн]. Available: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor. |
| [14] | „SignalR,“ Microsoft, [Онлайн]. Available: https://dotnet.microsoft.com/en-us/apps/aspnet/signalr. |
| [15] | „Mudblazor,“ [Онлайн]. Available: https://mudblazor.com/. |
| [16] | „JavaScript interoperability,“ Microsoft, [Онлайн]. Available: https://learn.microsoft.com/en-us/aspnet/core/blazor/javascript-interoperability/?view=aspnetcore-8.0. |
| [17] | „Google Charts,“ Google, [Онлайн]. Available: https://developers.google.com/chart. |
| [18] | „DBeaver,“ [Онлайн]. Available: https://dbeaver.io/. |
| [19] | „Rider,“ JetBrains, [Онлайн]. Available: https://www.jetbrains.com/rider/. |
| [20] | A. V. Hernàndez, „Kaggle,“ [Онлайн]. Available: https://www.kaggle.com/datasets/andreuvallhernndez/myanimelist/data?select=manga.csv. |
| [21] | „Repository Design Pattern,“ GeekForGeeks, [Онлайн]. Available: https://www.geeksforgeeks.org/repository-design-pattern/. |
| [22] | „Minimal APIs,“ Microsoft, [Онлайн]. Available: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis/overview?view=aspnetcore-8.0. |
| [23] | „Token based authentication,“ [Онлайн]. Available: https://www.okta.com/identity-101/what-is-token-based-authentication/. |
| [24] | „Role-Based Access Control,“ [Онлайн]. Available: https://auth0.com/docs/manage-users/access-control/rbac. |
| [25] | „jwt.io,“ [Онлайн]. Available: https://jwt.io/. |
| [26] | „Cross Origin Resource Sharing,“ Mozilla, [Онлайн]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. |

1. Romaji е системата, чрез която японските знаци се преобразуват в латински букви. [↑](#footnote-ref-1)