**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 1](#_Toc167636601)

[3 Съкращения 2](#_Toc167636602)

[4 Абстракт 3](#_Toc167636603)

[5 Структура на приложението и използвани технологии 4](#_Toc167636604)

[5.1 Сървърна част 4](#_Toc167636605)

[5.1.1 PostgreSQL 4](#_Toc167636606)

[5.1.2 ASP.NET Web API 4](#_Toc167636607)

[5.1.3 Redis 5](#_Toc167636608)

[5.1.4 Json Web Token (JWT) 5](#_Toc167636609)

[5.1.5 REST 6](#_Toc167636610)

[5.2 Потребителски интерфейс 6](#_Toc167636611)

[5.2.1 Blazor 6](#_Toc167636612)

[5.2.2 Google Charts 7](#_Toc167636613)

[6 Структура на базата данни 7](#_Toc167636614)

[6.1 LibraryMangas и Authors 8](#_Toc167636615)

[6.2 UserMangas 9](#_Toc167636616)

[6.3 Orders 10](#_Toc167636617)

[6.4 Таблици , свързани с потребителска информация 10](#_Toc167636618)

[7 Първоначални данни за LibraryMangas 11](#_Toc167636619)

[8 Използвана литература 13](#_Toc167636620)

# Списък на фигурите

[Фигура 1 Подписване на JWT 5](#_Toc167636621)

[Фигура 2 Примерен JWT 6](#_Toc167636622)

[Фигура 3 Структура на DbContext 7](#_Toc167636623)

[Фигура 4 Схема на базата данни 8](#_Toc167636624)

[Фигура 5: Прочитане на CSV файл 11](#_Toc167636625)

[Фигура 6: Прочитане на нужните данни 11](#_Toc167636626)

[Фигура 7 Проверка за съществуващи автори 12](#_Toc167636627)

# Съкращения

REST

JWT

API

ACID

HTTP

ORM

DOM

HTML

SVG

CSV

IDE

UI

# 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 (EF Core)

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 (JWT)

JWT [8] е стандарт за размяна на информация между две страни в JSON формат. Самият жетон 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 HS256.

В тялото се съдържа информация за потребителя 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

### 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 неща: HTTP метод (GET, POST, PUT, DELETE), Заглавни редове (Headers) с информация свързана за оторизиране, кеширане и други мета данни и 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.

# Реализация на информационната система

## Архитектура

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 за избор, информация за потребителите и техните колекции и поръчки. Сървърната част служи обработка на заявки от потребителите, изпълнение на бизнес логика, валидация и достъпване на данните от базата. Потребителския интерфейс служи представяне на информация и взаимодействие със сървъра и неговите функционалности.

## База данни

За да се имплементират всички функционалности са нужни 4 основни таблици

* LibraryManga – в тази таблица ще се съдържа информация за възможните заглавия.
* UserManga – таблица за отделните творби, които са свързани с потребител
* Orders – поръчките на потребителите
* User – информация за потребителите

## Сървърна част

## Потребителски интерфейс

# Структура на базата данни

Базата е генерирана чрез подхода „Първо код“ (Code first), в който първо се създават модели на отделните таблици и релациите между тях в кода и след това се създава миграция, която описва промените, които трябва да бъдат изпълнени върху базата. Чрез този метод се позволява лесна поддръжка и разширение на базата.

За да се реализира този подход е нужно да се напише клас, който да наследи DbContext. 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. Таблиците на потребителя ще бъдат генерирани от родителския клас IdentityDbContext<User>, където User е моделът за потребителски данни. След генериране на миграцията и нейното прилагане към базата получаваме показаната на Фиг. 4 схема на базата данни.

A computer screen with many black and white text

Description automatically generated with medium confidence

Фигура 4 Схема на базата данни

## LibraryMangas и Authors

Таблиците LibraryMangas и Authors са предназначени за възможните заглавия, от които потребителите могат да избират, както и информация за авторите.

Таблицата LibraryMangas има следните колони

* Id – идентификатор, който служи за първичен ключ (primary key)
* TitleRomaji – заглавието на латиница
* TitleEnglish – заглавието на английски. Възможно е заглавието да го няма, защото творбата може да не е преведена.
* TitleJapanese – заглавието на японски
* DemographicType – тип на демография, към която е насочена творбата. Възможни са стойностите
  + 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 все още се публикува творбата.
* MainImageURL – Линк към снимка на първия том. Възможно е да няма линк, защото творбата да няма все още издаден първи том.
* Synopsis – Описание

За таблицата Authors колоните са

* Id – идентификатор, първичен ключ
* FirstName – Първо име на автора. В някои случаи това име може да го няма, защото автора е 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 само 3 стойности в тази колона
  + Story – автора е отговорен само за историята
  + Art – автора е отговорен само за рисунките 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 използваме AspNetCode Identity, 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, парола, която е хеширана и опреснителен жетон (refresh token). 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.

# Структура и реализация на сървърната част

Сървърната част е реализирана чрез фреймуърк-а ASP.NET Web API и езика C#. Тази част е реализирана като независим от потребителския интерфейс компонент. Неговата цел е да обработва данните, които получава чрез REST заявки.

Архитектурата на

A close-up of a service layer

Description automatically generated

Фигура 8: Архитектура на сървърната част

На фиг. 8 е показаха архитектурата на сървърната част. Тя се състои от три основни слоя. В Endpoint слоя е имплементиран мениджмънта на HTTP заявки, оторизацията на потребители и кеширането. В Service слоя се намира бизнес логиката, валидацията на данни и генерирането на токъни. Repository слоят е имплементация на шаблона Repository, чрез който се предоставят различни операции за изпълнение към базата данни.

A screenshot of a computer

Description automatically generated

Фигура 9: Структура на сървърната част

На фиг. 9 е показана структурата на сървърната част. Съдържат се следните папки

* Properties – тук се намират всички необходими файлове за стартиране на проекта
* Configuration – в тази папка се съдържат всичката информация за конфигурирането на базата данни
* Constants – съдържат се всички константи, свързани с бизнес логиката, като различните статуси, видовете демография, както и технически константи, като таговете за кеширане, имена на потребителски полици и роли
* Contracts – обектите чрез които се определя формата на данните, които ще бъдат очаквани, както и формата на резултати които ще бъдат връщани в отговор.
* Data – в тази папка се намира имплементацията на DbContext, чрез която се осъществява връзката с базата.
* Endpoints – тук са всички крайни точки, които се използват за обработка на HTTP заявки
* Entities – класове, които представляват различните таблици в базата.
* Exceptions – персонализирани изключения, които се използват в проекта.
* Interfaces – дефиниции на различните хранилища (repository) и сервизни (service) класове
* Mapping – дефиниции на това как обектите биват трансформирани
* Messages – тук се съдържат различни съобщения, свързани предимно с валидацията на данни
* Migrations – класове, генерирани чрез EF, които служат за дефиниране и управление на схемата на базата.
* Repositories – имплементации на шаблона Repository, чрез който се предоставя различни операции за изпълнение към базата данни
* Services – сервизни класове, в които се изпълнява бизнес логика, валидация, преобразуване към
* Swagger – конфигурационни файлове, необходими на Swagger за реализация на документацията.
* Util – помощни класове и методи
* Validation – класове за валидиране на данни.

Във файла appsettings.json се намира информация, като линк за базата, информация за генерирането на JWT токъните и линк за Redis инстанцията, необходима за кеширането на данни.

В Program.cs се намират конфигурациите, за голяма част от допълнителните функционалности, като мениджмънт на потребители, кеширание и връзка с базата данни.

# 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,“ IETF, [Онлайн]. Available: https://jwt.io/. |
| [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. |