**A logo with black text

Description automatically generatedТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

Факултет Приложна математика и информатика

**ДИПЛОМНА РАБОТА**

Тема: Уеб базирана система за управление и контрол на лични и семейни финанси

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

Фак. №: 791324005

*Специалност:* Приложна Математика и Информатика

*Образователно- квалификационна степен*: **магистър**

Дипломен ръководител: доц. д-р Анна Розева

София 2025

Съдържание

[Списък на фигурите 3](#_Toc199426770)

[Списък на таблиците 3](#_Toc199426771)

[Съкращения 3](#_Toc199426772)

[1 Увод 5](#_Toc199426773)

[2 Използвани технологии 6](#_Toc199426774)

[2.1 PostgreSQL 6](#_Toc199426775)

[2.2 ASP.NET Web API 6](#_Toc199426776)

[2.2.1 Entity Framework Core 6](#_Toc199426777)

[2.2.2 AspNetCore Identity 6](#_Toc199426778)

[2.3 FluentValidation 6](#_Toc199426779)

[2.4 Redis 7](#_Toc199426780)

[2.4.1 AspNetCore.OutputCaching.StachExchangeRedis 7](#_Toc199426781)

[2.5 Json Web Token 7](#_Toc199426782)

[2.5.1 AspNetCore.Authentication.JwtBearer 8](#_Toc199426783)

[2.6 FluentEmail 8](#_Toc199426784)

[2.7 Hangfire 9](#_Toc199426785)

[2.8 REST 9](#_Toc199426786)

[2.9 Blazor 9](#_Toc199426787)

[2.9.1 MudBlazor 9](#_Toc199426788)

[2.9.2 JavaScript interoperability 10](#_Toc199426789)

[2.10 Google Charts 10](#_Toc199426790)

[2.11 JetBrains Rider 10](#_Toc199426791)

[2.12 Papercut SMTP 10](#_Toc199426792)

[3 Архитектура и реализация на приложението 11](#_Toc199426793)

[3.1 База данни 11](#_Toc199426794)

[3.1.1 PersonalCategories 13](#_Toc199426795)

[3.1.2 PersonalTransactions 14](#_Toc199426796)

[3.1.3 RecurringTransactions 14](#_Toc199426797)

[3.1.4 Family 15](#_Toc199426798)

[3.1.5 FamilyCategories 15](#_Toc199426799)

[3.1.6 FamilyTransactions 15](#_Toc199426800)

[3.1.7 FamilyInvitations 16](#_Toc199426801)

[3.1.8 Таблици свързани с потребителските лични данни 16](#_Toc199426802)

[3.2 Сървър 18](#_Toc199426803)

[3.2.1 Repository layer 18](#_Toc199426804)

[3.2.2 Service layer 19](#_Toc199426805)

[3.2.3 API layer 19](#_Toc199426806)

[3.3 Потребители 24](#_Toc199426807)

[3.3.1 Токън за достъп 26](#_Toc199426808)

[3.3.2 Опреснителен токън 28](#_Toc199426809)

[3.4 Автоматични транзакции 29](#_Toc199426810)

[3.5 Семейство 30](#_Toc199426811)

[3.6 Потребителски интерфейс 30](#_Toc199426812)

[3.7 CORS 30](#_Toc199426813)

[4 Заключение 32](#_Toc199426814)

[5 Използвана литература 33](#_Toc199426815)

[6 Приложение 34](#_Toc199426816)

# Списък на фигурите

[Фигура 1: Подписване на JWT 8](#_Toc199426747)

[Фигура 2: Примерен JWT 8](#_Toc199426748)

[Фигура 3: Архитектурна диаграма на приложението 11](#_Toc199426749)

[Фигура 4: Структура на DbContext класа 12](#_Toc199426750)

[Фигура 5: ER диаграма на базата данни 13](#_Toc199426751)

[Фигура 6: Архитектурна диаграма на сървъра 18](#_Toc199426752)

[Фигура 7: Пример за Minimal API крайна точка 19](#_Toc199426753)

[Фигура 8: Oперации за категориите на потребител 20](#_Toc199426754)

[Фигура 9: Операции за потрелски транзакции 21](#_Toc199426755)

[Фигура 10: Пример за кеширане 22](#_Toc199426756)

[Фигура 11: Операции за повтарящи се транзакции 22](#_Toc199426757)

[Фигура 12: Операции за семейство 22](#_Toc199426758)

[Фигура 13: Операции за семейни категории 23](#_Toc199426759)

[Фигура 14: Операции за семейни транзакции 23](#_Toc199426760)

[Фигура 15: Операции за статистически данни 23](#_Toc199426761)

[Фигура 16: Операции за оторизация на потребител 24](#_Toc199426762)

[Фигура 17: Пример за оторизация на операция 25](#_Toc199426763)

[Фигура 18: Списък с Claim обекти 26](#_Toc199426764)

[Фигура 19: Генериране на JWT 27](#_Toc199426765)

[Фигура 20: Конфигуриране на проверката на JWT 28](#_Toc199426766)

[Фигура 21: Генериране на опреснителен токън 28](#_Toc199426767)

[Фигура 22: Метод за създаване на транзакции 29](#_Toc199426768)

[Фигура 23: Конфигуриране за recurring job 30](#_Toc199426769)

# Списък на таблиците

**No table of figures entries found.**

# Съкращения

ACID - Atomicity, Consistency, Isolation, Durability

API - Application Programming Interface

CORS - Cross-Origin Resource Sharing

DOM - Document Object Model

HTML - HyperText Markup Language

HTTP - HyperText Transfer Protocol

IDE - Integrated Development Environment

JSON – Java Script Object Notation

JWT - JSON Web Token

ORM - Object-Relational Mapping

REST - Representational State Transfer

SMTP – Simple Mail Transfer Protocol

SVG - Scalable Vector Graphics

UI - User Interface

WASM - WebAssembly

# Увод

В условията на динамично развиваща се икономика и нарастващите цени на стоки и услуги, потребителите срещат все по-големи предизвикателства при контрола на личните си бюджети. Усложняващата се структура на разходите правят традиционните методи за финансово планиране като бележници и електронни таблици недостатъчно ефективни и непрозрачни. От друга страна технологичния напредък, разпространението на смартфони и компютри създава предпоставки за разработването на интуитивни приложения. Те могат да улеснят процеса на проследяване и анализ на приходите и разходите, предлагайки по-ясна представа за структурата на бюджета.

Без подходящ дигитален инструмент потребителите често нямат пълна прозрачност относно финансовите си потоци. Разходите се разпределят на различни места и се извършват през различни моменти, което затруднява проследяването им и анализа на навиците за харчене. В резултат трудно се установяват точните суми, изразходвани по категории, но и от различните членове на семейството. Освен това, липсата на централизирана система води до затруднена комуникация и координация между членовете, предизвиквайки недоразумения и напрежение при управлението на семейния бюджет.

Целта на разработваното приложение е да предложи цялостно и решение за управлението на лични и семейни финанси. С това приложение потребителите ще могат лесно да създават и организират категории за разходи и приходи, както и да въвеждат всяка транзакция в своя финансов план. Приложението осигурява споделен контрол върху семейния бюджет, тъй като позволява на членовете на семейството да формират група, в рамките на която всеки може да добавя своите транзакции и да наблюдава резултатите от управлението на общия бюджет.

# Използвани технологии

## PostgreSQL

PostgreSQL [1] е релационна база данни с отворен код и позволява на потребителите да създават собствени типове данни под формата на обекти, върху които могат да се прилагат наследяване и полиморфизъм. Поддържат се транзакции с ACID свойства и поддръжка на други езици за писане на скриптове, освен SQL.

## ASP.NET Web API

ASP.NET Web API [2] е рамка за разработване на софтуер, която позволява създаване на уеб приложения чрез езика за програмиране C# [3]. Той е част от ASP.NET Core платформата и предоставя голям брой библиотеки за разработване на REST базирани услуги.

### Entity Framework Core

Entity Framework Core (EF Core) [4] e библиотека, която позволява интеграция на база данни и проект. Библиотеката работи на ORM принципа, при който записите от таблица в базата биват превърнати в обекти, които могат да се използват от приложението.

### AspNetCore Identity

AspNetCore Identity [5] е библиотека която позволява лесно съхранение и обработка на потребителски акаунти, възможните роли в дадено приложени, както и възможни права, които потребителя може да притежава.

## FluentValidation

FluentValidation [6] e библиотека която предоставя лесно валидиране на обектки. Тя позволява дефиниране на правила чрез fluent синтаксит, което представлява свързване на различните методи, познато като method chaining, за да се изрази логиката по естествен и лесно четим начин.

## Redis

Redis [7] е нерелационна база данни, работещата на принципа „ключ-стойност“. Най често използвана е за кеширане на данни. Дизайнът на Redis позволява бързо изпълнение на операции, защото данните се съхраняват в паметта, а не на диск.

### AspNetCore.OutputCaching.StachExchangeRedis

AspNetCore.OutputCaching.StackExchangeRedis [8] e библиотека която имплементира кеширане чрез Redis. Output Caching е междиден слой в приложението, чрез който е възможно кеширането на заявката, която е подадена на съвръра като „ключ“, а стойността е отговора който бива продуциран.

## Json Web Token

Json Web Token [9] e стандарт за размяна на информация между две страни в JSON формат. Токънът съдържа три части, които са кодирани в Base 64 формат, които са разделени с точка. Частите са: Заглавен ред (Header), Тяло (Payload) и Подпис (Signature).

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

Подписът се генерира като неподписаният токън, който представлява комбинация от кодирания заглавен ред и кодираното тяло разделени с точка, се криптира с таен ключ от 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 number

AI-generated content may be incorrect.

Фигура 2: Примерен JWT

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

### AspNetCore.Authentication.JwtBearer

AspNetCore.Authentication.JwtBearer [10] е библиотека, в която има имплементация за валидиране и създаване на JWT, както и възможността да бъде реализирана оторизация на потребителите, посредством JWT.

## FluentEmail

FluentEmail [11] e библиотека която позволява генериране на имейли с поддръжка за HTML както и изпращането им чрез различни доставчици като SMTP или SendGrid. Позволява структурирането на имейлите чрез fluent синтаксиса.

## Hangfire

Hangfire [12] е библиотека, чрез която може да бъдат реализирани задачи, които трябва да се изпълнят в задния план (background jobs). Поддържат се различни видове задачи: моментни, отложени и периодични. Също така предоставя и табло за наблюдение в което може да се следят различните задачи.

## REST

REST [13] е архитектурен стил за реализиране на уеб услуги между клиент и сървър, които комуникират с HTTP протокола. Заявките носят в себе си следните компоненети: HTTP метод като Get, Post, Delete и тн; заглавни редове (Headers) в която има информация свързана с оторизирация, кеширане и други мета данни и тяло (Body) в което се съдържа основната информация. Отговорите имат в себе си статус код, който показва какво се е случило, Headers в които има информация за сървъра който изпълнява заявката и тяло, в което се съдържа информация, в повечето случаи в JSON формат.

## Blazor

Blazor [14] e рамка за разработване на интерактивен потребителски интерфейс, базиран на компонентния модел написани на езика C#. Blazor позволява два вида хостинг Blazor Web Assembly (WASM) при който кода се компилира и се изпълнява в браузъра или Blazor Server, където всички операции се изпълняват на обособен сървър и те биват комуникирани към клиента чрез SignalR [15], библиотека която позволява асинхорнно изпращане на информация.

### MudBlazor

MudBlazor [16] е компонентна библиотека която предоставя реализирани компоненти като таблици, бутони полета за въвеждане на данни и други.

### JavaScript interoperability

JavaScript interoperability или JSInterop [17] е функционалност която позволява на Blazor Server да изпълни JavaScript код в браузъра, като и JS код да изпълни C# код, когато е необходимо. Такива случаи са когато трябва да се достъпи целия DOM на браузра или интегриране на други софтуери от трети страни, за които е възможно единствено интеграция чрез JavaScript.

## Google Charts

Google Charts [18] е JavaScript библиотека която позволява генериране на различни диаграми като стълбовидна или кръгова диаграма в HTML или SVG формат.

## JetBrains Rider

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

## Papercut SMTP

Papercut SMTP [20] е софтуер който позволява преглеждане и изпращане на имейли повреме на разработка. Позволява се преглеждане на всичко в един изпратен имейл като съдържание, прикачени файлове и други.

# Архитектура и реализация на приложението

Архитектурата на проекта следва „клиент-сървър“ модела, който осигурява разделение на потребителския интерфейс от логиката за обработка на данните. Проекта се състои от три компонента: база данни, сървър и потребителски интерфейс. Базата данни отговаря за съхранението на всички данни които се използват като информация за потребителя, неговите категории и транзакции, както и техните аналози в семейната структура. Сървърът е предназначен за обработка на заявките от клиента, изпълнение на бизнес логика, валидация и достъпване на данни от базата. Потребителския интерфейс служи за взаимодействие с потребителя, изпращане на заявки към сървъра и визуализация на информацията, предоставена от сървъра.

A diagram of a server

AI-generated content may be incorrect.

Фигура 3: Архитектурна диаграма на приложението

## База данни

Базата е създадена чрез подхода „Първо код“ (Code first), в който първо се създават модели на отделните таблици и релациите между тях. Този метод 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

AI-generated content may be incorrect.

Фигура 4: Структура на DbContext класа

Всяка една променлива от тип DbSet<T> представя таблица от базата. За реализация на приложението са създадени осем таблици.

* PersonalCategories - таблица отностно категориите на потребителя.
* FamilyCategories – таблица с информация за категориите на дадено семейство.
* PersonalTransactions – таблица за транзакциите на потребителя
* FamilyTransactions – информация за транзакциите на семейство
* RecurringTransactions – таблица с детайли за повтарящите се транзакции на един потребител
* Users – информация за регистрираните потребители, като имейл адрес, потребителско име и парола.
* Family – информация за семейството
* FamilyInvitations –информация за всички изпратени покани за различните семейства.

Поради наличието на потребители в системата, ще използваме IdentityDbContext, чрез който ще бъдат генерирани таблици за 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 да се създадат таблиците в базата. За създаването ѝ се използва командата „dotnet ef migrations add name“, където name е името на миграцията.. След създаването ѝ, миграцията може да бъде приложена върху базата чрез командата „dotnet ef database update“.

A screenshot of a computer screen

AI-generated content may be incorrect.

Фигура 5: ER диаграма на базата данни

На фигура 5 е показана схемата на базата данни, която се получава след прилагането на миграцията.

### PersonalCategories

Таблицата PersonalCategories съдържа информация за различните категории които един потребител може да има. Таблицата има следните колони

* Id – идентификатор, който служи за първичен ключ (primary key)
* Name – името на категорията
* Icon – иконка, чрез която категорията може да бъде визуализирана. Възможно е категорията да е без икона.
* Type – типът на категорията. Може да има следните стойностти
  + Income – категорията е свързана с дохода на потребителя
  + Expense – категорията е свързана с разходите на потребителя
* Limit – месечен лимит на за категорията. Възможно е да няма зададен лимит за дадена категория.
* UserId – идентификатор на потребителя, към който се асоциира категорията, външен ключ (foreign key)

### PersonalTransactions

Таблицата PersonalTransactions е предназначена за информацията за различните транзакции които един потребител може да има. Колоните на таблицата са:

* Id – идентификатор, първичен ключ
* Amount – стойността на транзакцията
* Description – кратко описание. Възможно е описанието да не бъде въведено.
* TransactionDate – дата на изпълнение на транзакцията
* PersonalCategoryId – идентификатор към категорията за която тази транзакция е въведена, външен ключ
* UserId – идентификатор на потребителя който е създал транзакцията

### RecurringTransactions

RecurringTransactions таблицата е предназначена за информацията отностно повтарящите се транзакции. В нея има следните колони:

* Id – идентификатор, първичен ключ
* Amount – стойността на транзакцията
* Description – кратко описание
* Type – типът на транзакцията. Може да бъде една от следните стойностти
  + Weekly – транзакцията да се повтаря всяка седмица
  + BiWeekly – транзакцията да се повтаря на две седмици
  + Monthly – транзакцията да се повтаря всеки месец
* StartDate – дата на започване на повторенията
* EndDate – дата на приключване на повторенията
* NextExecutionDate – следваща дата на която тази транзакция да бъде въведена в системата
* PersonalCategoryId – идентификатор на категорията на транзакцията, вънщен ключ
* UserId – идентификатор на потребителя, външен ключ

### Family

В таблицата Family се съдържа основната информация за семейството. Таблицата има две колони: Id – идентификатор на семейството и Name – името на семейството.

### FamilyCategories

Таблицата FamilyCategories е аналогична на PersonalCategories. В нея се съхранява информация за категорията която се използва в дадено семейство. Има следните колони

* Id – идентификатор, първичен ключ
* Name – името на категорията
* Icon – иконка, чрез която категорията може да бъде визуализирана. Възможно е категорията да е без икона.
* Type – типът на категорията. Може да има следните стойностти
  + Income – категорията е свързана с дохода на потребителя
  + Expense – категорията е свързана с разходите на потребителя
* Limit – месечен лимит на за категорията. Възможно е да няма зададен лимит за дадена категория.
* Family – идентификатор на семейството, външен ключ

### FamilyTransactions

FamilyTransactions таблицата също е аналогична на PersonalTransactions таблицата. В нея се съхраняват транзакциите които са на членовете на едно семейство.Неините колони са

* Id – идентификатор, първичен ключ
* Amount – стойност на транзакцията
* Description – кратко описание
* TransactionDate – дата на изпълнение на транзакцията
* FamilyCategoryId – идентификатор на категорията на транзакцията, външен ключ
* UserId – идентификатор на потребителя създал транзакцията, външен ключ
* FamilyId – идентификатор на семейството, на което принадлежи тази транзакция, външен ключ.

### FamilyInvitations

Таблицата FamilyInvitations съдържа информация за изпратените покани към потребители за присъединяване към семейство

* Id – идентификатор на поканата, първичен ключ
* Email – имейл към който е изпратена поканата
* FamilyId – идентификатор на семейството към което трябва потребителя да се присъедини
* UserInApplication – булева променлива която показва дали поканата е изпратена към съществуващ в системата потребител
* CreatedOnUtc – дата на създаване на поканата
* ExpiresOnUtc – дата на изтичане на поканата

### Таблици свързани с потребителските лични данни

Тъй като използвахме IdentityDbContext, са генерирани множество таблици отностно потребителя и неговите лични данни. За целите на проекта ще се използват само три от таблиците: AspNetUsers, AspNetRoles и AspNetUserRoles.

В AspNetUsers се съхраня основната информация за потребителя като потребителско име, имейл адрес, парола която е хеширана и други. В таблицата AspNetRoles се намира информация за възможните роли в приложението. Релация между потребителя и ролите е много към много и тя е осъществена чрез таблицата AspNetUserRoles. В нея се съхранява информацията за кой потребител какви роли има.

## Сървър

Сървърът е реализиран чрез ASP.NET Web API и езика C#. Той следва многослойна архитектура, в която функционалносттите са обособени в различни слоеве. Сървърът има за задача да обработва заявки от клиента, да валидира получените данни и да позволява оторизацията на потребителя в система.

A diagram of a api

AI-generated content may be incorrect.

Фигура 6: Архитектурна диаграма на сървъра

На фигура 6 е представена архитектурата на сървъра. Състои се от 3 основни слоя: Repository layer, Service layer и API layer.

### Repository layer

Repository layer представлява имплементация на шаблона за дизайн Repository [21]. В него се прави абстракция на източника на данни и се осигурява централизиран начин за управление на операциите с данни. С имплементацията на този шаблон се позволява работа с обекти, вместо с SQL скриптове или друг вид операции и се изолира от бизнес логиката. Дизайна има три основни компонента: модел, дефиниция и имплементация. Моделът представлява самите данни за приложението. Дефиницията е интерфейс в който са дефинирани различните операции които да се прилагат на данните, като добавяне или изтриване, а имплементацията се предоставя конкретна логика как да бъде извършена дадена операция.

### Service layer

Service слоя представлява бизнес логиката на приложението. В това ниво се обработват заявките от клиента, валидират се предоставените данни както и се взима информация от Repository слоя. Като компоненти на това ниво имаме UserService в който се намира логиката за потребители, като регистрация и вписване в системата, PersonalFinanceServices и FamilyFinanceServices, в които се намира логиката за обработване съотвено на лични и семейни финанси и StatisticsService който генерира статистически данни спрямо финансовите навици на потребителя.

### API layer

API слоя е най горното ниво, което служи за комуникация с външния свят. Неговата цел е да приема различните заявки от потребителя, да препраща информация към Service слоя и да връща отговори. Това ниво е реализирано чрез Minimal API моделът. Minimal API [22] e oпростен начин за изграждане на крайни точки (endpoints) които да приемат заявки от потребителя.

A computer screen shot of a computer code

Description automatically generated

Фигура 7: Пример за Minimal API крайна точка

На фиг.7 е показан пример за endpoint, който връща текста „Hello World“, когато се извика главният път на сървъра (root).

За реализация на функционалносттите 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.

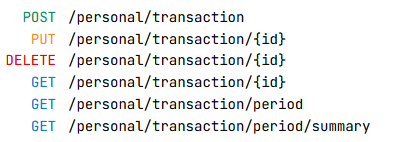
За да се направи разлика между потребителските и семейните обекти в пътя на всяка операция има добавена представяка която показва към коя от двете структури е обособена операцията. За лични финанси е „/personal/“, а за семейни е „/familial/“.

A close-up of a list of words

AI-generated content may be incorrect.

Фигура 8: Oперации за категориите на потребител

На фигура 8 са показани реализираните операции отностно различните категории на един потребител. В операциите PUT, DELETE и GET {id}, което е параметър към заявката и се показва че е необходимо да се подаде идентификатор на дадена категория за да се изпълни операцията. Операциите POST и PUT също така се нуждаят от тяло (request body) в което да се съдържат необходимите данни съответно за създаване или промяна на категорията. Операцията „GET /personal/category/user“ връща всички категории които принадлежат на оторизирания потребител.



Фигура 9: Операции за потрелски транзакции

На фигура 9 са показани операциите свързани с транзакциите на един потребител. Операциите са аналогични на тези за категориите, с изключение на последните две:

* /personal/transaction/period
* /personal/transaction/period/summary

И двете операции имат като параметри две дати които са началната и крайната дата на период.

* /personal/transaction/period – връща всички транзакции за посочения период
* /personal/transaction/period/summary – връща обобщена информация за периода, включително
  + Обща стойност на приходи и разходи
  + Разбивка на разходите по категории

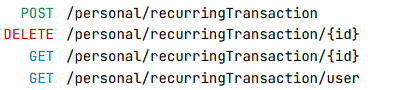
Поради наличието на много транзакции и гъвкавост в избора на период за получаване на информацията за транзакции е имплементирано кеширане в Redis чрез междинния слой (middleware) OutputCaching. При постъпване на модифицираща данните заявка (POST, PUT, DELETE), кешираните данни биват премахнати, защото няма да са актуални, след изпълнението на операцията. Кеширането се осъществява чрез CacheOutput метод, който се закача за MapGet метода, който дефинира каква функционалност да се изпълни при получаване на GET заявка на дадения път и чийто резултат искаме да бъде кеширан. В CacheOutput се конфигурира колко време да бъде кеширан резултата.

A close-up of a word

AI-generated content may be incorrect.

Фигура 10: Пример за кеширане

На фигура 11 са показани всички операции отностно повтарящите се транзакции



Фигура 11: Операции за повтарящи се транзакции

Операциите за повтарящите се транзакции са аналогични с изключението че няма функционалност за промяна на информацията. Това е съзнателно решение, тъй като промяна на информация като типът на повторение или началната дата може да доведе несъответствие в историята. Вместо актуализация се препоръчава следния подход:

1. Изтриване на старата повтаряща се транзакция чрез операцията DELETE
2. Създаване на нова с актуализирани стойностти чрез операцията POST

A group of black text

AI-generated content may be incorrect.

Фигура 12: Операции за семейство

Операциите за семейство са представени на фигура 12. Реализирани са създаване и изтриване на семейство посредством POST и DELETE заявки както и получаване на информация за семейство с посоченото id чрез GET заявка. Заявката /family/all връща всички възможни семейства във системата и е предназначена за администраторите на платформата. Заявката /family/invite е предназначена за изпращане на покани за присъединяване към дадено семейство.

A close-up of a computer screen

AI-generated content may be incorrect.

Фигура 13: Операции за семейни категории

Операциите за семейните категории са аналогични на персоналните категории. Има операции PUT, DELETE и GET които приемат идентификатор като параметър за да изпълнят конкретното действие. За операциите POST и PUT е необходимо да се предостави request body. Операцията „GET /familial/FamilyCategory/family“ е аналогична на „GET /personal/category/user“ при която се връщат всички категории на семейството към което потребителя е член.

A close-up of a computer screen

AI-generated content may be incorrect.

Фигура 14: Операции за семейни транзакции

Операциите за семейните транзакции са аналогични на операциите за персонални транзакции. Реализирани са операции POST, PUT и DELETE съответно за добавяне, променяне на информация и изтриване на транзакция. Също така две GET операции за получаване на информация за период, дефиниран чрез два параметъра.

A close up of a text

AI-generated content may be incorrect.

Фигура 15: Операции за статистически данни

Отностно статистическите данни са реализирани операции за генериране на годишен отчет за личните финанси на потребител, както и за семейство. Годишния отчет съдържа информация за приходи и доходи за всеки месец, както и информация за цялосната сума разпределена по категории.

A screenshot of a computer screen

AI-generated content may be incorrect.

Фигура 16: Операции за оторизация на потребител

Операциите за потребителя са стандартни POST заявки за регистриране и вписване в системата. Има реализиран refresh операция която предоставя нови токъни за достъп на потребителя и revoke операция която премахва текущия достъп. Също има реализирана „GET user/joinFamily/{token}” заявка, която се използва за добавяне на потребител към дадено семейство. Тази операция се достъпва чрез имейл покана изпратена към потребил от семейния администратор на дадено семейство.

## Потребители

За целите на приложението е реализиран механизъм за удостоверяване, базиран на токъни (Token based authentication) [23]. Това е метод за проверка на идентичността на потребителя. Потребителят 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 токън за достъп, чрез който потребителят може за определен период от време да достъпва функционалности на сървъра. Този токън се връща на потребителя и се съхранява в браузъра или в устройството, ако е мобилно приложение.

За управление на достъпа до различните функциионалностти е реализиран контрол на достъпа, чрез роли (Role-Based Access Control) [22]. С този подход на всеки потребител бива дадена една или повече роли, като всяка роля има предварително дефинирани права за достъп до ресурси и извършване на действия. В системата има четири роли.

* User – това е роля която се прилага на всеки потребител в системата.
* Admin – администраторска роля, която позволява допълнителни операции, които са недостъпни за стандартния потребител (пр. достъп до информацията за всички семейства).
* FamilyAdmin – това е роля която потребител получа при създаване на семейство и му позволява да добавя, променя и премахва семейни категории, както и да добавя и премахва членове на семейството.
* FamilyMember – тази роля позволява на потребителя да добавя транзакции към семейната история. Потребители с тази роля нямат права за добавяне, промяна и премахване на категории.

Информацията за ролите на един потребител се задават в момента на регистрация или при успешно създаване на семейство.

За операциите на които е необходимо да потребителя да има дадена роля се използва RequireAuthorization метод, който се използва след дефинирането на операцията и в него се задава коя роля е необходима, за да се изпълни. За операции при които не е обходима оторизация (например регистрация и вписване) се използва AllowAnonymous метод с който операцията може да бъде изпълнена от всеки който има достъп до приложението, независимо дали е оторизиран.

A screenshot of a computer program

Description automatically generated

Фигура 17: Пример за оторизация на операция

Когато потребителят се вписва в системата се генерират 2 токъна - за достъп и за опресняване. Те имат време на живот, в което могат да бъдат използвани. Токънът за достъп трябва да е със сравнително кратно време на живот, докато опреснителният трябва да е с голям живот. Добра практика е токъните за достъп да са с време на живот между 30 секунди и 5 минути, докато опреснителните да са между 14 и 30 дни.

### Токън за достъп

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

A computer screen shot of text

Description automatically generated

Фигура 18: Списък с Claim обекти

В списъка се съдържат обекти, в които се съдържа 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 19: Генериране на JWT

Списъкът с информацията за потребителя се поставя в Subject полето. Стандартните полета които трябва да се дефинират са

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

В обекта се добавят също времето на живот чрез Expires (exp) полето и ключът и алгоритъмът за подписване. Методът CreateToken на JwtTokenHandler позволява да се генерира токън чрез SecurityTokenDescriptor обекта

За да е възможна проверката е необходимо да се конфигурира какво трябва да се проверява. Това е възможно с AddJwtBearer метода. За да е валидна една проверка трябва полетата Issuer и Audience да съществуват в токъна и да имат предварително дефинираните стойности. Също трябва да е с валиден подпис и да е с валидно време на живот, като е позволен толеранс от пет секунди.

A computer screen shot of text

Description automatically generated

Фигура 20: Конфигуриране на проверката на JWT

След като токънът за достъп изтече, потребителският интерфейс може да използва опреснителния токън, за да генерира нов.

### Опреснителен токън

Токънът 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 неговото изтичане, потребителят трябва да се впише отново в системата.

A screenshot of a computer program

Description automatically generated

Фигура 21: Генериране на опреснителен токън

На фиг. 15 е показано как се създава опреснителен токън. Чрез генератор на случайни числа се генерира масив от 64 числа в диапазона 0-255, които след това са конвертирани в Base64 стринг.

## Автоматични транзакции

За реализирането на автоматичните транзакци, които се създават от повтарящите се транзакции е използвана библиотекатa Hangfire.

Hangfire използва сериализация за да съхрани информация за задачите, които трябва да се изпълнят. При дефинирането на задача Hangfire сериализира кода и го съхранява в база данни, като така задачите не се губят. Информацията която се съхранява в базата са името на метода и неговата сигнатура, параметри и техните стойностти, типът на задачата и други.

A screenshot of a computer code

AI-generated content may be incorrect.

Фигура 22: Метод за създаване на транзакции

На фигура 22 е показан класа в който се намира логиката и който ще бъде изпълняван от Hangfire. В него се прочитат всички повтарящи се транзакции, на които датата им на изпълнение е днес. Всяка една повтаряща се транзакция бива превърната в PersonalTransaction обект с помощта на метода ToPersonalTransaction. Новата транзакция бива добавена в базата с помощта на PersonalTransactionRepository класа. След успешното създаване на транзакцията се пресмята новата дата за изпълнение и се променя съществуващия запис в базата. След това се печата кратко съобщение, чрез което може да се разбере дали има създадена транзакция или е настъпил някакъв проблем.

За да може Hangfire да разбере как да изпълни този метод е необходимо да бъде конфигуриран така наречения RecurringJob. Това става посредством RecurringJob класа от Hangfire и метода AddOrUpdate. Този метод приема идентификатор на процеса, функцията която трябва да се изпълни, както и вида повторение. За целите на приложението вида на повторение ще бъде на дневно, т.е всеки ден ще се повтаря.

A computer screen shot of text

AI-generated content may be incorrect.

Фигура 23: Конфигуриране за recurring job

## Семейство

## Потребителски интерфейс

## CORS

# Заключение

# 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] | „C# language documentation,“ Microsoft, [Онлайн]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/. |
| [4] | „Entity Framework Core,“ Microsoft, [Онлайн]. Available: https://learn.microsoft.com/en-us/ef/core/. |
| [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] | Jeremy Skinner , „FluentValidation,“ [Онлайн]. Available: https://docs.fluentvalidation.net/en/latest/. |
| [7] | „Redis,“ Redis, [Онлайн]. Available: https://redis.io/. |
| [8] | „AspNetCore OutputCaching StackExchange Redis,“ Microsoft, [Онлайн]. Available: https://learn.microsoft.com/en-us/aspnet/core/performance/caching/output?view=aspnetcore-8.0. |
| [9] | „JWT,“ [Онлайн]. Available: https://auth0.com/docs/secure/tokens/json-web-tokens. |
| [10] | „AspNetCore.Authentication.JWT.Bearer,“ Microsoft, [Онлайн]. Available: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-8.0. |
| [11] | „FluentEmail,“ [Онлайн]. Available: https://github.com/lukencode/FluentEmail. |
| [12] | Hangfire. [Онлайн]. Available: https://www.hangfire.io/. |
| [13] | „RESTful API,“ [Онлайн]. Available: https://restfulapi.net/. |
| [14] | „Blazor,“ Microsoft, [Онлайн]. Available: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor. |
| [15] | „SignalR,“ Microsoft, [Онлайн]. Available: https://dotnet.microsoft.com/en-us/apps/aspnet/signalr. |
| [16] | „Mudblazor,“ [Онлайн]. Available: https://mudblazor.com/. |
| [17] | „JavaScript interoperability,“ Microsoft, [Онлайн]. Available: https://learn.microsoft.com/en-us/aspnet/core/blazor/javascript-interoperability/?view=aspnetcore-8.0. |
| [18] | „Google Charts,“ Google, [Онлайн]. Available: https://developers.google.com/chart. |
| [19] | „Rider,“ JetBrains, [Онлайн]. Available: https://www.jetbrains.com/rider/. |
| [20] | „Papercut SMTP,“ ChangemakerStudios, [Онлайн]. Available: https://github.com/ChangemakerStudios/Papercut-SMTP. |
| [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. |

# Приложение

Линк към кодът на проекта в GitHub: