**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](#_Toc199252956)

[Списък на таблиците 3](#_Toc199252957)

[Съкращения 3](#_Toc199252958)

[1 Увод 5](#_Toc199252959)

[2 Използвани технологии 6](#_Toc199252960)

[2.1 PostgreSQL 6](#_Toc199252961)

[2.2 ASP.NET Web API 6](#_Toc199252962)

[2.2.1 Entity Framework Core 6](#_Toc199252963)

[2.2.2 AspNetCore Identity 6](#_Toc199252964)

[2.3 FluentValidation 6](#_Toc199252965)

[2.4 Redis 7](#_Toc199252966)

[2.4.1 AspNetCore.OutputCaching.StachExchangeRedis 7](#_Toc199252967)

[2.5 Json Web Token 7](#_Toc199252968)

[2.5.1 AspNetCore.Authentication.JwtBearer 8](#_Toc199252969)

[2.6 FluentEmail 8](#_Toc199252970)

[2.7 Hangfire 8](#_Toc199252971)

[2.8 REST 9](#_Toc199252972)

[2.9 Blazor 9](#_Toc199252973)

[2.9.1 MudBlazor 9](#_Toc199252974)

[2.9.2 JavaScript interoperability 9](#_Toc199252975)

[2.10 Google Charts 10](#_Toc199252976)

[2.11 JetBrains Rider 10](#_Toc199252977)

[2.12 Papercut SMTP 10](#_Toc199252978)

[3 Архитектура и реализация на приложението 11](#_Toc199252979)

[3.1 База данни 11](#_Toc199252980)

[3.1.1 PersonalCategories 13](#_Toc199252981)

[3.1.2 PersonalTransactions 14](#_Toc199252982)

[3.1.3 RecurringTransactions 14](#_Toc199252983)

[3.1.4 Family 15](#_Toc199252984)

[3.1.5 FamilyCategories 15](#_Toc199252985)

[3.1.6 FamilyTransactions 15](#_Toc199252986)

[3.1.7 FamilyInvitations 16](#_Toc199252987)

[3.1.8 Таблици свързани с потребителските лични данни 16](#_Toc199252988)

[3.2 Сървър 16](#_Toc199252989)

[4 Заключение 19](#_Toc199252990)

[5 Използвана литература 20](#_Toc199252991)

[6 Приложение 21](#_Toc199252992)

# Списък на фигурите

[Фигура 1: Подписване на JWT 8](#_Toc199252952)

[Фигура 2: Примерен JWT 8](#_Toc199252953)

[Фигура 3: Структура на DbContext класа 11](#_Toc199252954)

[Фигура 4: ER диаграма на базата данни 13](#_Toc199252955)

# Списък на таблиците

# 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 computer code

Description automatically generated

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. Тази библиотека ще позволи изпълняването на автоматични транзакции, които ще са предварително дефинирани от потребителите.

## 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 се състои от три компонента: база данни, сървър и потребителски интерфейс (UI). В базата се съдържат всички данни, които се използват из приложението, като информация за потребителя, неговите категории и транзакции, както и техния аналог в семейната структура. Сървърът е предназначен за обработка на заявки от потребителите, изпълнение на бизнес логика, валидация и достъпване на данните от базата. Потребителският 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. 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 3: Структура на DbContext класа

Всяка една променлива от тип DbSet<T> представя таблица от базата. За реализация на функционалността са създадени 8 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 4: ER диаграма на базата данни

На фигура 4 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 2 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

# 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. |

# Приложение

Линк към кодът на проекта в GitHub: