**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:* Информатика и Софтуерни Науки

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

Дипломен ръководител: доц. д-р Анна Розева

София 2025

Съдържание

[Списък на фигурите 3](#_Toc200286575)

[Съкращения 4](#_Toc200286576)

[1 Увод 6](#_Toc200286577)

[2 Използвани технологии 7](#_Toc200286578)

[2.1 PostgreSQL 7](#_Toc200286579)

[2.2 ASP.NET Web API 7](#_Toc200286580)

[2.2.1 Entity Framework Core 7](#_Toc200286581)

[2.2.2 AspNetCore Identity 7](#_Toc200286582)

[2.3 FluentValidation 7](#_Toc200286583)

[2.4 Redis 8](#_Toc200286584)

[2.4.1 AspNetCore.OutputCaching.StachExchangeRedis 8](#_Toc200286585)

[2.5 Json Web Token 8](#_Toc200286586)

[2.5.1 AspNetCore.Authentication.JwtBearer 9](#_Toc200286587)

[2.6 FluentEmail 9](#_Toc200286588)

[2.7 Hangfire 9](#_Toc200286589)

[2.8 REST 10](#_Toc200286590)

[2.9 Blazor 10](#_Toc200286591)

[2.9.1 MudBlazor 10](#_Toc200286592)

[2.9.2 JavaScript interoperability 10](#_Toc200286593)

[2.10 Google Charts 11](#_Toc200286594)

[2.11 JetBrains Rider 11](#_Toc200286595)

[2.12 Papercut SMTP 11](#_Toc200286596)

[3 Архитектура и реализация на приложението 12](#_Toc200286597)

[3.1 База данни 12](#_Toc200286598)

[3.1.1 PersonalCategories 14](#_Toc200286599)

[3.1.2 PersonalTransactions 15](#_Toc200286600)

[3.1.3 RecurringTransactions 15](#_Toc200286601)

[3.1.4 Family 16](#_Toc200286602)

[3.1.5 FamilyCategories 16](#_Toc200286603)

[3.1.6 FamilyTransactions 16](#_Toc200286604)

[3.1.7 FamilyInvitations 17](#_Toc200286605)

[3.1.8 Таблици свързани с потребителските лични данни 17](#_Toc200286606)

[3.2 Сървър 18](#_Toc200286607)

[3.2.1 Repository layer 18](#_Toc200286608)

[3.2.2 Service layer 19](#_Toc200286609)

[3.2.3 API layer 19](#_Toc200286610)

[3.3 Потребители 25](#_Toc200286611)

[3.3.1 Токън за достъп 26](#_Toc200286612)

[3.3.2 Опреснителен токън 28](#_Toc200286613)

[3.4 Автоматични транзакции 30](#_Toc200286614)

[3.5 Семейство 32](#_Toc200286615)

[3.6 Потребителски интерфейс 34](#_Toc200286616)

[3.6.1 Начална страница 34](#_Toc200286617)

[3.6.2 Потребители 34](#_Toc200286618)

[3.6.3 Категории 36](#_Toc200286619)

[3.6.4 Потребителски транзакции 38](#_Toc200286620)

[3.6.5 Автоматични транзакции 41](#_Toc200286621)

[3.6.6 Семейство 42](#_Toc200286622)

[3.6.7 Семейни категории 44](#_Toc200286623)

[3.6.8 Семейни транзакции 46](#_Toc200286624)

[3.6.9 Годишен отчет 49](#_Toc200286625)

[3.7 Обмяна на данни 51](#_Toc200286626)

[4 Заключение 52](#_Toc200286627)

[5 Използвана литература 54](#_Toc200286628)

[6 Приложение 55](#_Toc200286629)

# Списък на фигурите

[Фигура 1: Подписване на JWT 9](#_Toc200286523)

[Фигура 2: Примерен JWT 9](#_Toc200286524)

[Фигура 3: Архитектурна диаграма на приложението 12](#_Toc200286525)

[Фигура 4: Структура на DbContext класа 13](#_Toc200286526)

[Фигура 5: ER диаграма на базата данни 14](#_Toc200286527)

[Фигура 6: Архитектурна диаграма на сървъра 18](#_Toc200286528)

[Фигура 7: Пример за Minimal API крайна точка 19](#_Toc200286529)

[Фигура 8: Операции за категориите на потребител 20](#_Toc200286530)

[Фигура 9: Операции за потребителски транзакции 21](#_Toc200286531)

[Фигура 10: Пример за кеширане 22](#_Toc200286532)

[Фигура 11: Операции за автоматични транзакции 22](#_Toc200286533)

[Фигура 12: Операции за семейство 22](#_Toc200286534)

[Фигура 13: Операции за семейни категории 23](#_Toc200286535)

[Фигура 14: Операции за семейни транзакции 23](#_Toc200286536)

[Фигура 15: Операции за статистически данни 23](#_Toc200286537)

[Фигура 16: Операции за оторизация на потребител 24](#_Toc200286538)

[Фигура 17: Пример за оторизация на операция 26](#_Toc200286539)

[Фигура 18: Списък с Claim обекти 27](#_Toc200286540)

[Фигура 19: Генериране на JWT 27](#_Toc200286541)

[Фигура 20: Конфигуриране на проверката на JWT 28](#_Toc200286542)

[Фигура 21: Генериране на опреснителен токън 29](#_Toc200286543)

[Фигура 22: Метод за създаване на транзакции 30](#_Toc200286544)

[Фигура 23: Конфигуриране на recurring job 31](#_Toc200286545)

[Фигура 24: Процес за създаване на семейство 32](#_Toc200286546)

[Фигура 25: Процес за приемане на покана 33](#_Toc200286547)

[Фигура 26: Начална страница 34](#_Toc200286548)

[Фигура 27: Форма за регистрация 35](#_Toc200286549)

[Фигура 28: Форма за вписване 35](#_Toc200286550)

[Фигура 29: Методи за добавяне и премахване на cookie-та 36](#_Toc200286551)

[Фигура 30: Форма за регистрация на нови администратори 36](#_Toc200286552)

[Фигура 31: Страница за категории 37](#_Toc200286553)

[Фигура 32: Форма за създаване на категории 37](#_Toc200286554)

[Фигура 33: Форма за модифициране на категория 38](#_Toc200286555)

[Фигура 34: Табло с основа информация 39](#_Toc200286556)

[Фигура 35: Форма за създаване на транзакции 40](#_Toc200286557)

[Фигура 36: Страница с транзакции 40](#_Toc200286558)

[Фигура 37: Модифициране на транзакция 41](#_Toc200286559)

[Фигура 38: Автоматични транзакции 41](#_Toc200286560)

[Фигура 39: Форма за създаване на автоматични транзакции 42](#_Toc200286561)

[Фигура 40: Табло за контрол на членовете на семейството 43](#_Toc200286562)

[Фигура 41: Екран за изпращане на покани 43](#_Toc200286563)

[Фигура 42: Форма за създаване на семейство 44](#_Toc200286564)

[Фигура 43: Екран за семейни категории за член на семейството 45](#_Toc200286565)

[Фигура 44: Екран за семейни категории за семеен администратор 45](#_Toc200286566)

[Фигура 45: Форма за създаване на семейни категории 45](#_Toc200286567)

[Фигура 46: Форма за промяна на семейни категории 46](#_Toc200286568)

[Фигура 47: Обща информация за семейни транзакции 47](#_Toc200286569)

[Фигура 48: Страница със семейни транзакции 47](#_Toc200286570)

[Фигура 49: Форма за създаване на семейни транзакции 48](#_Toc200286571)

[Фигура 50: Годишен отчет 49](#_Toc200286572)

[Фигура 51: Генериране на диаграма 50](#_Toc200286573)

[Фигура 52: Конфигуриране на CORS политика 51](#_Toc200286574)

# 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

OCR – Optical Character Recognition

ORM - Object-Relational Mapping

REST - Representational State Transfer

SMTP – Simple Mail Transfer Protocol

SVG - Scalable Vector Graphics

WASM – WebAssembly

# Увод

В условията на динамично развиващата 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 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 транзакции 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 други скриптови езици, различни от 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 в 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 приложение, както и възможни права, които потребителят може да притежава.

## FluentValidation

FluentValidation [6] e библиотека, която предоставя лесно валидиране на обекти. Тя позволява дефиниране на правила чрез fluent синтаксис, 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, работеща на принципа „ключ-стойност“. Най често е използвана за кеширане на данни. Дизайнът на Redis позволява бързо изпълнение на операции, защото данните се съхраняват в оперативната памет, а не на диск.

### AspNetCore.OutputCaching.StachExchangeRedis

AspNetCore.OutputCaching.StackExchangeRedis [8] e библиотека, която имплементира кеширане чрез Redis, посредством Output Caching. Output Caching е междинен слой в приложението, чрез който се позволява кеширането на заявката, която е подадена към сървъра като „ключ“, а стойността е отговора, който бива продуциран.

## Json Web Token

Json Web Token (JWT) [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 алгоритъмa 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 подписа. Чрез единичната проверка, JWT позволява да бъде самостоятелна структура, която съдържа в себе си информация и метода 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. Структурирането на имейлите се осъществява 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 компоненти: HTTP метод като Get, Post, Delete и други; заглавни редове (Headers), в които има информация свързана с оторизация, 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 и написани на езика C#. Blazor предоставя два вида реализация: Blazor Web Assembly (Blazor WASM), при който кодът 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 асинхронно 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 браузъра 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), която е предназначена 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 по време на разработка. Позволява се преглеждане на всички компоненти в един изпратен имейл като съдържание, прикачени файлове, метаданни и други.

# Архитектура и реализация на приложението

Архитектурата на проекта следва „клиент-сървър“ модела, който осигурява разделение на потребителския интерфейс от бизнес логиката. 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. Потребителският 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 необходимо 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 относно 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 за шаблоните на автоматичните транзакции на един потребител
* Users – информация за регистрираните потребители.
* 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 – име на категорията
* Icon – иконка, чрез която категорията може да бъде визуализирана. Възможно е категорията да е без икона.
* Type – тип на категорията. Може да има следните стойности
  + Income – категорията е свързана с доходите на потребителя
  + Expense – категорията е свързана с разходите на потребителя
* Limit – месечен лимит на 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 свързана с транзакциите, които един потребител може да има. Колоните на таблицата са:

* Id – идентификатор; първичен ключ
* Amount – стойност 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; външен ключ

### RecurringTransactions

RecurringTransactions таблицата е предназначена за информацията отностно шаблоните за автоматични транзакции. В нея има следните колони:

* Id – идентификатор, първичен ключ
* Amount – стойност на транзакцията
* Description – кратко описание
* Type – тип на повторение 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
* EndDate – дата на приключване
* NextExecutionDate – следваща дата на изпълнение на автоматичната транзакция
* PersonalCategoryId – идентификатор на категорията на транзакцията; външен 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 – име на категорията
* Icon – иконка, чрез която категорията може да бъде визуализирана. Възможно е категорията да е без икона.
* Type – тип 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.
* FamilyId – 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. Нейните 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 използваме IdentityDbContext, са генерирани множество таблици отностно потребителя. За целите на проекта ще се използват само три таблици: AspNetUsers, AspNetRoles и AspNetUserRoles. В AspNetUsers се съхраня основната информация за потребителя. В таблицата AspNetRoles се намира информация за възможните роли в приложението. В AspNetUserRoles се пази информацията за ролите на потребителите.

## Сървър

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

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, Service и API.

### 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 постига изолиране на функционалността за четене на данни, от бизнес логиката. Дизайна има три основни компонента: модел, дефиниция и имплементация. Моделът представлява структурата на данните за приложението. Дефиницията е интерфейс, в който са дефинирани различните операции, които да се прилагат на данните, като добавяне или изтриване, а в имплементацията се предоставя конкретната 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 съответно на лични и семейни финанси и StatisticsService, който генерира статистически данни, спрямо финансовите навици на потребител или семейство.

### 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 опростен 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. В приложението ще се използват четири 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.

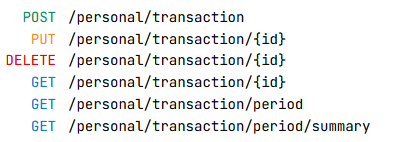
Тъй като използваме HTTP протокола, е възможно на един път да се поставят няколко операции. За да се направи разлика между потребителските и семейните операции, в пътя на всяка операция има добавена представка, която показва към коя от двете структури е обособена операцията. За лични финанси е „/personal/“, а за семейни е „/familial/“.

A close-up of a list of words

AI-generated content may be incorrect.

Фигура 8: Операции за категориите на потребител

На фигура 8 са показани реализираните операции относно различните категории на един потребител. В операциите PUT, DELETE и GET се съдържа елемента {id}, което е параметър към заявката и с него се показва, че е необходимо да се подаде идентификатор на 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 потребителски 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

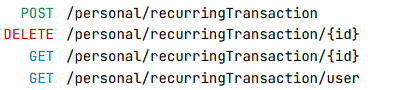
Поради наличието на много транзакции и гъвкавост в избора на период за получаване на информацията за транзакции е имплементирано кеширане в Redis чрез междинния слой (middleware) OutputCaching. При постъпване на модифицираща данните заявка (POST, PUT, DELETE), кешираните данни биват премахнати след изпълнението на операцията, защото няма да са актуални. Кеширането се осъществява чрез CacheOutput метод, който се поставя след 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 автоматичните транзакции



Фигура 11: Операции за автоматични транзакции

Операциите за автоматичните транзакции са аналогични на операциите за обикновените потребителски транзакции, 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 до несъответствие в историята. Вместо актуализация се препоръчва следния подход:

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

A computer code with 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 посочено id чрез GET заявка. Операцията /family/all връща всички възможни семейства в системата и е предназначена за администраторите на платформата. Операцията /family/invite е предназначена за изпращане на покани за присъединяване към дадено семейство. Реализиране е заявка за премахване на член от семейството. При премахването му от семейството, потребителят спира да има достъп до семейството. Транзакциите, които принадлежат на премахнатия член, се запазват в семейната история.

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. Реализирани са 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/category/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, промяна 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. Годишният отчет съдържа информация за приходи и разходи за всеки месец, както и информация за приходите и разходите през годината, разпределени между различните 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 операции за регистриране и вписване в системата. Има реализирана refresh операция, която предоставя нов токън за достъп на потребителя и revoke операция, която премахва текущия достъп. Също има реализирана „GET user/joinFamily/{token}” операция, която се използва за добавяне на потребител към дадено семейство. Тази операция се достъпва чрез имейл покана изпратена към потребител 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. При вписване в приложението, потребителят въвежда своите данни за идентификация, като имейл адрес 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 функционалносттите 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 функционалности 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 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 –роля, която потребителят получа при създаване на семейство.
* FamilyMember – тази роля се прилага на потребител, при приемане на покана за присъединяване към семейство и му позволява да добавя транзакции към семейната история.

Ролите на един потребител се задават в момента на успешна регистрация, създаване на семейство или приемане на покана за присъединяване към семейство.

За операциите, на които е необходимо потребителя да има дадена роля, се използва RequireAuthorization метод, който се поставя след дефинирането на операцията и в него се задава коя роля е необходима, за да се изпълни. За операции, при които не е необходима 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 два 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. Токъните за достъп реализирани от приложението имат време на живот 5 минути, а токъните за опресняване имат време на живот 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 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 има информация за самия потребител като неговото потребителско име, идентификатора му и ролите, които са приложими за него, както и поле за идентификатор на токъна. 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, преди което токънът не е активен

В обекта се добавят също времето на живот чрез полето Expires (exp), ключа за подписване и алгоритъма за подписване. Методът CreateToken на JwtTokenHandler позволява да се генерира токън от 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 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 низ.

## Автоматични транзакции

За реализирането на автоматичните транзакции, които се създават от шаблоните е използвана библиотеката 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 стойносттите им, 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 класът, в който се намира логиката за създаване на транзакции от шаблоните и който ще бъде изпълняван от Hangfire. В него се прочитат всички шаблони на автоматични транзакции, на които датата им на изпълнение съвпада с датата за изпълнение на процедурата. Всеки един шаблон бива превърнат 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 шаблон в базата. След това се показва кратко съобщение в логовете на приложението, чрез което може да се разбере дали има създадена транзакция.

За да може Hangfire да разбере как да изпълни този метод е необходимо да бъде конфигуриран процеса по изпълнение. 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 изпълнение. Видът на изпълнението се задава чрез cron expression, което е формат за задаване на график за автоматично изпълнение. За целите на приложението ще използваме „0 0 \* \* \* \* “, който показва, че процедурата ще се изпълни в 0 часа и 0 минути всеки ден.

A computer screen shot of text

AI-generated content may be incorrect.

Фигура 23: Конфигуриране на recurring job

## Семейство

За да е възможно организирането на семеен бюджет е реализирана функционалност за създаване на семейство, което представлява група от хора, които имат достъп до общи категории и обща история за транзакции. За тази цел има реализиран процес по създаване на семейство показан на фигура 24.

A white background with black text

AI-generated content may be incorrect.

Фигура 24: Процес за създаване на семейство

Потребителят може да стартира процеса по създаване на семейство от потребителския интерфейс, където ще трябва да въведе име на семейството. След успешното създаване на семейството потребителя получава ролята FamilyAdmin, която му позволява да добавя и премахва членове на семейството, както и да създава различни категории, които да бъдат използвани от членовете на семейството. След това потребителят ще трябва да изпрати поне една покана. Поканите се изпращат чрез имейл. След въвеждане на имейл адреса, системата изпраща покана, която има валидност от 24 часа за приемане. Ако семейният администратор е избрал да изпрати повече от една покана, процесът ще се повтори.

A diagram of a user registration

AI-generated content may be incorrect.

Фигура 25: Процес за приемане на покана

На фигура 25 е показан процесът за приемане на покана от потребител. Ако поканата бъде приета се прави проверка, дали получателят на поканата е регистриран потребител. Ако потребителят е регистриран, той получава ролята FamilyMember, която му позволява да добавя транзакции към семейната история, както и права за четене (read only rights) на семейните категории. Членовете на семейството не могат да добавят, променят и изтриват категории. Ако не е регистриран, потребителят е препратен към страницата за регистриране. След успешна регистрация потребителят получава ролята FamilyMember и процеса приключва.

## Потребителски интерфейс

За реализиране на потребителския интерфейс е използвана рамката Blazor Server. Тази рамка предоставя лесно реализиране на визуална среда, която потребителите могат да използват да достъпът операциите, реализирани от сървъра.

### Начална страница

Приложението притежава начална страница, която приветства потребителите, независимо от това дали са вписани в системата. В случай, че потребителят е вписан се появява панел с информация за дохода и разходите направени през настоящия месец.

Приложението притежава меню за навигация, разположено в дясната страна на екрана. От него могат да се достъпят страниците за личните категории и транзакции на потребителя. При наличие на семейство, страниците за семейни категории и транзакции се появяват. Последна е страницата за годишна статистика. Реализирано е потребителско меню, в което се съдържат функциите за регистриране, вписване на потребителя (login) и опция за отписване (logout), ако вече има вписан потребител.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 26: Начална страница

### Потребители

Приложението притежава реализирани страници за регистрация и вписване. Те са достъпни от потребителското меню, което се намира в горната част на екрана. При регистрация, потребителят трябва да предостави потребителско име, имейл адрес и парола, която трябва да повтори.

A screenshot of a register

Description automatically generated

Фигура 27: Форма за регистрация

При успешно регистриране, потребителят бива препратен към страницата за вписване.

A screenshot of a login screen

Description automatically generated

Фигура 28: Форма за вписване

При успешно вписване в системата от сървъра се изпращат двойка токъни, които трябва да бъдат записани в паметта на браузъра, за да могат да бъдат достъпвани при изпращането на заявки от потребителския интерфейс към сървъра. Те се запазват под формата на бисквита (cookie). В Blazor Server няма директен достъп до cookie-тата, защото кодът се изпълнява на сървъра. За операции, които изискват взаимодействие с браузъра, като достъпването на document.cookie се използва JsInterop, който да изпълни JavaScript код от страна на клиента. За тази цел са реализирани два метода: addCookie и deleteCookie, които се извикват съответно при вписване в системата, която добавя получените токъни в браузъра и функция, която премахва cookie-тата при отписване от системата. За да е възможно едно cookie да бъде изтрито, трябва неговата валидност да е в миналото. Добра практика е да се използва 01.01.1970 като дата на изтичане.

A computer code with text

Description automatically generated

Фигура 29: Методи за добавяне и премахване на cookie-та

Реализирана е и страница за регистрация на нови администратори, която е достъпна само от текущи администратори.

A screenshot of a login form

Description automatically generated

Фигура 30: Форма за регистрация на нови администратори

### Категории

За потребителските категории са реализирани три екрана. Екран, в който потребителят може да достъпи всичките си категории, екран, в който може да се създаде нова категория и екран за модифициране на категория.

В екрана за всички категории е реализиран панел, в който са изброени всички категории, всяка в отделен елемент с информация като име, икона, какъв е типа на категорията и лимит. Всеки елемент има два бутона: бутон „Update“, който позволява промяна на данните на категорията и бутон „Delete“, който изтрива категорията.

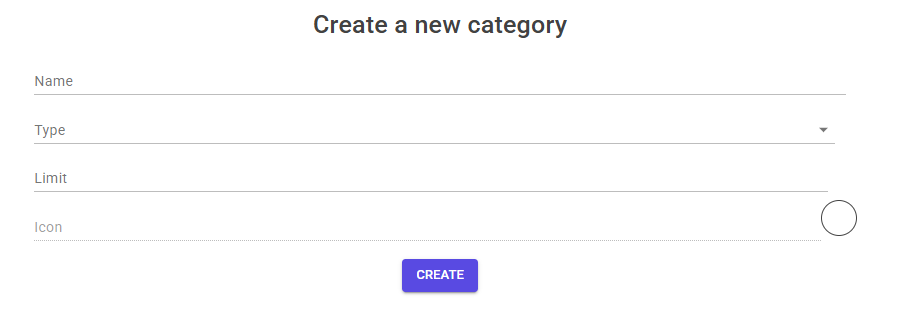
A screenshot of a computer

AI-generated content may be incorrect.

Фигура 31: Страница за категории

За добавяне на категория се използва бутонът „Create”, който отваря формата за добавяне на категория. Във формата за добавяне потребителя може да добави информация като името на категорията, нейния тип, да зададе лимит и да избере икона, с която да се визуализира категорията. Реализирани са следните контроли:

* Потребителя трябва да въведе име на категорията.
* Потребителя трябва да избере тип на категорията.



Фигура 32: Форма за създаване на категории

При създаване на категория потребителя ще получи информативно съобщение, че категорията е създадена и може да продължи да създава категории.

При модифициране на категорията се генерира форма с предварително попълнени данни. Контролите реализирани във формата за модифициране са същите като формата за създаване. При успешно модифициране потребителя ще бъде препратен към страницата с всички категории.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 33: Форма за модифициране на категория

### Потребителски транзакции

При отваряне на страницата за транзакции, потребителят има достъп до табло, което съдържа основна информация за доходите и разходите му за текущия месец, както и предходни месеци. Потребителят може да достъпи информацията за предишни месеци чрез бутоните „Previous“ и „Next“, разположени в горната част на таблото.

Информацията е структурирана под формата на панел в който се съдържа всички доходи и всички разходи, като разходите са разделени спрямо категориите. Реализирана е и кръгова диаграма, която визуално представя всички разходи на потребителя.

A screenshot of a pie chart

AI-generated content may be incorrect.

Фигура 34: Табло с основа информация

При натискане на бутонът „Add transaction“ потребителят отваря формата за добавяне на транзакции. Във формата потребителят може да въведе информация за датата на транзакцията, стойността ѝ, кратко описание. Реализирани са следните контроли:

* Датата на извършване на транзакцията трябва да бъде попълнена и да не трябва да е в бъдещето.
* Стойността на транзакцията трябва да бъде попълнена и трябва да бъде положително число.
* Категорията на транзакцията трябва да бъде избрана от падащото меню. В зависимост дали транзакцията трябва да бъде свързана с приход или доход има реализиран бутон с две опции, на база на който се зареждат категориите на потребителя.

След успешно създаване на транзакция, потребителят ще получи информативно съобщение, че категорията е създадена и има възможността да продължи да създава транзакции.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 35: Форма за създаване на транзакции

При натискане на бутона „More information“ потребителя ще бъде препратен към страница, съдържаща всички транзакции за текущия месец. Транзакциите са структурирани в таблица, съдържаща информация за дата на транзакция, описание на транзакцията, стойност на транзакцията, категорията както и бутони за различните действия приложими на транзакцията. Транзакциите са сортирани по дата на транзакция в низходящ ред. При натискане на бутонът „To Summary“ потребителят ще се върне към страницата с обща информация.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 36: Страница с транзакции

С бутона „Update“ потребителят ще отвори форма за модифициране на транзакция, в която предварително ще бъдат заредени данните на транзакцията. Контролите са същите като създаване на транзакция. При успешна промяна потребителят ще бъде препратен към екрана с транзакции.

A screenshot of a computer screen

AI-generated content may be incorrect.

Фигура 37: Модифициране на транзакция

### Автоматични транзакции

За автоматичните транзакции на потребителя са реализирани две страници: страница с всички възможни автоматични транзакции и страница за добавяне. Страница с всички автоматични транзакции представлява таблица с основна информация като начална и крайна дата, описание и стойност на автоматичната транзакция, категорията на транзакцията и типът на повторение. Реализиран е бутон, който да изтрие автоматичната транзакция.

A screenshot of a computer screen

AI-generated content may be incorrect.

Фигура 38: Автоматични транзакции

Потребителят може да добави нова автоматична транзакция чрез бутонът „Create”. С натискането му потребителят ще бъде препратен към форма за създаване. В нея потребителят може да въведе информация като начална и крайна дата, стойност на транзакцията и описание. Може да избере категорията, към която е транзакцията. Реализирани са следните контроли

* Стойността трябва да бъде въведена и да е повече от 0.
* Описанието трябва да бъде въведено
* Началната дата трябва да бъде въведена и трябва да бъде в бъдещето
* Крайната дата трябва да бъде въведена и трябва да е след датата на създаване.
* Категорията трябва да бъде избрана.
* Типът на повторение трябва да бъде избран.

A screenshot of a computer screen

AI-generated content may be incorrect.

Фигура 39: Форма за създаване на автоматични транзакции

### Семейство

Ако текущият потребител е член на семейство, то той има достъп до табло с обща информация за членовете на семейството. Това табло позволява на семейния администратор да премахне член от семейството.

A screenshot of a family member form

AI-generated content may be incorrect.

Фигура 40: Табло за контрол на членовете на семейството

Семейният администратор може да добави допълнителни членове, чрез „Add Members“ бутона, който ще го отведе то страницата за изпращане на покани. В нея той има възможността да въведе имейл адреса на потребителя, който иска да бъде поканен, както и да избере дали ще изпраща повече от една покана.

A screenshot of a website

AI-generated content may be incorrect.

Фигура 41: Екран за изпращане на покани

Ако потребителят не е член на семейство, то той има възможността да създаде семейство. При създаване на семейство е необходимо да предостави име на семейството, с което то да бъде създадено в системата

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 42: Форма за създаване на семейство

След като потребителя създаде семейство ще бъде препратен към страницата за изпращане на покани, показана на фигура 41.

### Семейни категории

Функционалността за семейните категории е разпределена в три страници: екран за категориите, екран за създаване и екран за промяна.

Екранът с всички категории е достъпен от всички членове на семейството. В него са реализирани три раздел: всички категории, категории за доход и категории за разход. В зависимост от това кой раздел е отворен са изредени категории. Всяка категория е представена в панел, в който се съдържа информация като име, икона и лимит.

Визуализацията на категориите е различна спрямо видът на семейната роля на потребителя. Ако потребителят е семеен администратор, то той ще може да види във всяка категория два бутона: „Update“, с който може да се промени информацията за категорията и „Delete“, чрез който може да се изтрие категорията. Също така ще се появи още един бутон „Create“, който се намира над изброените категории и позволява създаването на нови семейни категории.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 43: Екран за семейни категории за член на семейството

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 44: Екран за семейни категории за семеен администратор

Екранът за създаване на нови семейни категории е аналогичен на този за създаване на персонални категории. Реализирана е форма, в която потребителя може да въведе информация като името на категорията, нейния тип, да зададе лимит и да избере икона, с която да се визуализира категорията. Реализирани са следните контроли:

* Потребителя трябва да въведе име на категорията.
* Потребителя трябва да избере тип на категорията.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 45: Форма за създаване на семейни категории

След успешно създаване на категорията потребителят ще получи информативно съобщение и ще може да продължи да добавя нови категории. При промяна на категория се зарежда формата с предварително заредени данните на категорията. Контролите за промяна на категория са същите като при добавяне.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 46: Форма за промяна на семейни категории

### Семейни транзакции

За семейните транзакции са реализирани четири екрана: екран с основна информация за финансовите движения през текущия месец, екран с транзакциите за текущия месец, екран за добавяне на транзакция и екран за промяна.

Екранът с общата информация е аналогичен с този от персоналните транзакции. Реализирано е табло с информация за предходните. Месеците могат да се сменят с бутоните „Previous“ и „Next“ разположени в горната част. Информацията е представена в панел, в който се съдържа всички доходи и всички разходи, като разходите са разделени спрямо категориите. Под панелът е реализирана кръгова диаграма, която представя всички разходи на потребителя. С бутонът „More information“, потребителят може да достъпи екран, в който са изредени всички транзакции през текущият месец.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 47: Обща информация за семейни транзакции

Страницата за транзакции през текущият месец е реализирана като таблица, в която са подредени всички транзакции. В таблицата има информация за датата на изпълнение на транзакцията, кой член на семейството е изпълнил тази транзакция, както нейното описание, стойността ѝ и категорията. Има реализирани и два бутона: за промяна на данните и за изтриване на транзакцията. Семейният администратор има права да промени или да изтрие всяка транзакция, а членовете - само собствените си транзакции.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 48: Страница със семейни транзакции

За създаване на нова транзакция е реализирана форма, която може да се достъпи чрез бутона „Add Transaction“, намиращ се в горната част на екрана. В тази форма потребителят може да въведе информация като дата на транзакцията, нейната стойност, кратко описание и да избере категорията. Реализирани са следните контроли

* Датата на транзакция трябва да не е в бъдещето.
* Стойността трябва да бъде положителна.
* Описанието трябва да бъде въведено
* Категорията трябва да бъде избрана. В зависимост от избрания тип на категория в падащото меню се зареждат всички семейни категории, отговарящи на типът.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 49: Форма за създаване на семейни транзакции

Процесът е сходен на този за създаване на персонални транзакции с изключението, че в семейните транзакции описанието на транзакцията е задължително. Това е направено с цел по добра отчетност в семейната история.

Реализирана е и форма за промяна на съществуваща транзакция. Формата е сходна на тази за добавяне на транзакция. Контролите са същите както тези при създаване на транзакция.

### Годишен отчет

Реализирана е страница за годишни отчети. Тя представлява два панела с информация за персоналните финансови движения и за семейните. Ако потребител не е част от семейство то панелът за семейна информация не се вижда. Всеки панел съдържа информация, представена в две диаграми: една за доходи и разходи за всеки месец от годината и една представяща всички приходи и разходи разделени в различните категории на потребителя. Всеки панел има два бутона „Previous” и „Next“, с който се контролира за коя година да се показват данни.

A screenshot of a graph

AI-generated content may be incorrect.

Фигура 50: Годишен отчет

Диаграмите се генерират с библиотеката Google Charts. Тъй като библиотеката е разработена на JavaScript, за нейното използване в Blazor е необходимо използването на JsInterop. За генериране на една диаграма са необходими три елемента: данни за диаграмата, настройки на диаграмата и тип на диаграмата. Данните трябва да са заредени в DataTable обект. Това е клас, който представлява двумерна таблица, и се използва от всички видове диаграми, които се подържат от библиотеката. В настройките на диаграмата се конфигурират различните визуални аспекти от диаграмата като заглавие, цвят, легенда и други. Тези опции са представени като JSON обект. За реализирането на диаграмата е необходимо да се създаде обект, който да отговаря на типа диаграма. Например за кръгова диаграма се използва PieChart, а за стълбовидна - ColumnChart. При създаването на обект е необходимо да се подаде идентификатор на елемент от екрана, най често това е div елемент, в който да бъде поставена диаграмата при неиното генериране. Чрез метода draw се генерира графиката по зададените данни и опции.

A screen shot of a computer code

AI-generated content may be incorrect.

Фигура 51: Генериране на диаграма

## Обмяна на данни

В приложението са реализирани клиент, в който се намира потребителския интерфейс и сървър, който отговорен за обработка на потребителските заявки. Комуникацията между сървъра и клиента се извършва през HTTP. Обикновено клиента се хоства на различен домейн от сървъра. Поради това, че те са на различни домейни възниква проблем, свързан със сигурността, наложена от браузърите, наречен политика за същия произход (Same-Origin Policy).

Тази политика е вградена в повечето съвременни браузъри и представлява защитен механизъм, който предотвратява възможността на уеб страниците да изпращат заявки към домейн, който е различен собствения домейн, на който принадлежи уеб страницата. Целта на тази политика е да предотврати атаки като кражба на чувствителни данни от друг сайт чрез JavaScript код.

За да е възможна тази комуникация е необходимо изрично да се позволи изпращането на такива заявки. Това може да бъде постигнато чрез механизъма споделяне на ресурси от други източници (Cross-Origin Resource Sharing – CORS). CORS [25] дава възможност на сървъра да приема заявки, идващи от предварително дефинирани външни източници, като в същото време продължава да защитава достъпа от неоторизирани потребители.

За реализация е необходимо да се конфигурира CORS политика в сървъра. За тази цел в ASP.NET се използват два метода: AddCors и UseCors. В методът AddCors се конфигурират допустимите източници, а с UseCors се прилага дадена политика при стартиране на приложението

A computer screen shot of a code

Description automatically generated

Фигура 52: Конфигуриране на CORS политика

# Заключение

Целта на настоящата дипломна работа бе да се разработи уеб базирано приложение за управление на лични и семейни финанси. Основната идея зад системата е да предостави удобен начин за въвеждане и проследяване на приходи и разходи, категоризация на транзакции и получаване на полезна информация за финансовото състояние на потребителя.

След проектиране на архитектурата и базата от данни, беше изградена сървърната част на приложението, в която се реализира основната бизнес логика, както и уеб интерфейс, позволяващ работа със системата. Всички основни цели на проекта бяха постигнати, като приложението покрива основните нужди на един потребител при управление на финанси.

С оглед на бъдещото развитие на системата могат да бъдат реализирани редица функционални и технологични подобрения, които биха повишили нейната стойност и удобство за крайните потребители. На първо място, логично продължение на проекта е създаването на мобилно приложение, което да използва вече реализирания сървър, като по този начин осигури по-голяма достъпност и гъвкавост. Мобилната версия ще позволи на потребителите да въвеждат транзакции в движение, както и да следят бюджетите си във всеки момент.

Друг важен аспект, който значително би подобрил потребителското изживяване, е интегрирането на оптично разпознаване на символи (OCR). Чрез него потребителите ще могат само със снимка на касова бележка или фактура автоматично да добавят разход към системата, без да е необходимо ръчно въвеждане. Това не само спестява време, но и редуцира риска от човешка грешка.

Системата може да бъде допълнена и с възможност за работа с различни валути. Това ще позволи на потребителите да въвеждат транзакции в различни валути според техните реални обстоятелства (например при пътуване в чужбина), като впоследствие информацията може да бъде конвертирана и обобщавана спрямо избрана основна валута.

Подобряване на потребителския интерфейс, като се използва по-интуитивен дизайн и се добавят възможности за персонализация (например тъмен режим, различни цветови теми, пренареждане на елементи). Тези подобрения ще направят приложението по-лесно за използване и ще подобрят цялостното потребителско изживяване.

Накрая, важно направление за бъдещото развитие остава и повишаването на сигурността на приложението чрез прилагане на механизми като двуфакторна автентикация и криптиране на чувствителни 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] | „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. |
| [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] | „Cross Origin Resource Sharing,“ Mozilla, [Онлайн]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. |

# Приложение

Линк към проекта в GitHub: https://github.com/Exonault/MastersProject