**A black background with a black square

AI-generated content may be incorrect.ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

Факултет Приложна математика и информатика

**ДИПЛОМНА РАБОТА**

Тема: **“Приложение за управление на лични финанси, интегриращо базиран на изкуствен интелект асистент”**

**Дипломант**: Васил Паскалев Ганев

**Факултетен номер**: 471221010

**Научен ръководител:**  доц. д-р Златко Захариев

**Специалност**: Информатика и софтуерни науки

**ОКС**: Бакалавър

СОФИЯ 2025

**Съдържание**

[**Въведение**……………………………...……………………………………....5](#Въведение)

[**Глава 1: Цел на проекта**…………….,,.……………………………………...7](#Цел_на_проекта)

[1.1 Към какъв тип потребители е насочен проекта?.............................7](#Към_какъв_тип_потребители_е_насочен)

[1.2 Предимства на FinEdge, спрямо подобни налични продуктил…..7](#Защо_потребителите_да_ползват_FinEdge)

[1.3 Какво прави FinEdge по-добрият избор?..........................................8](#Какво_прави_FinEdge_подобрият_избор)

[**Глава 2: Технологии и среди за разработка**…………...…………………..9](#Технологии_и_среди_за_разработка)

[2.1 Използвани Технологии…………………………………………….9](#Използвани_Технологии)

[2.1.1 C#.............................................................................................9](#C_шарп)

[2.1.2 NET Core Web API………………………………………….9](#NET_Core_Web_API)

[2.1.3 Entity Framework……………………………………………9](#Entity_Framework)

[2.1.4 JWT (JSON Web Token)…………………………………….9](#JWT)

[2.1.5 Swashbuckle………………………………………………..10](#Swashbuckle)

[2.1.6 Hangfire…………………………………………………….10](#Hangfire)

[2.1.7 OpenAI Developer API…………………………………….10](#OpenAI_Developer_API)

[2.1.8 Vite…………………………………………………………10](#Vite)

[2.1.9 TypeScript…………………………………………………..10](#TypeScript) [2.1.10 React………………………………………………………10](#React)

[2.1.11 React Router DOM………………………………………..11](#React_Router_DOM)

[2.1.12 TanStack Query (React Query)……………………………11](#TanStack_Query)

[2.1.13 KendoReact……………………………………………….11](#KendoReact)

[2.1.4 Tailwind…………………………………………………….11](#Tailwind)

[**2.2 Среди за разработка**…..………………………………………….12](#Среди_за_разработка)

[2.2.1 Visual Studio………………………………………………..12](#Visual_Studio)

[2.2.2 Visual Studio Code…………………………………………12](#Visual_Studio_Code)

[2.2.3 SQL Server Management Studio (SSMS)………………….12](#SQL_Server_Management)

[2.2.4 Node.js……………………………………………………...12](#Node_js)

[**Глава 3: Анализ на изискванията към проекта…………………………13**](#Анализ_на_изискванията_към_проекта)

[3.1 Функционални изисквания към проекта…………………………13](#Функционални_изисквания_към_проекта)

[3.1.1 Публична част……………………………………………..13](#Публична_част)

[3.2 Нефункционални изисквания към проекта………………………13](#Нефункционални_изисквания_към_проекта)

[3.2.1 Сигурност………………………………………………….13](#Сигурност)

[3.2.1 Производителност………………………………………...13](#Производителност)

[3.2.1 Поддръжка............................................................................14](#Поддръжка)

[3.2.1 Употреба и достъп...............................................................14](#Употреба_и_достъп)

[3.2.1 Съвместимост.......................................................................14](#Съвместимост)

[3.3 Структура на базата данни...............................................................14](#Структура_на_базата_данни)

[3.4 Архитектура на проекта...................................................................17](#Архитектура_на_проекта)

[3.5 Интеграцията с OpenAI API и LLM (Large Language Model).......17](#Интеграцията_с_OpenAI_API)

[**Глава 4: Техническа реализация…………………………………………..19**](#Техническа_реализация)

[4.1 Структура на проекта……………………………………………...19](#Структура_на_проекта)

[4.1.1 FinEdgeFrontend…………………………………………...19](#FinEdgeFrontend)

[4.1.2 FinEdgeBackend……………………………………………21](#FinEdgeBackend)

[4.1.2 FinEdgeData…..……………………………………………21](#FinEdgeData)

[4.1.2 FinEdgeServices……………………………………………22](#FinEdgeServices)

[4.1.3 FinEdgeTests……………………………………………….23](#FinEdgeTests)

[4.2 Създаване на потребители и потребителски профили…………..24](#Създаване_на_потребители_и_потребителски)

[4.3 Създаване на Акаунти (сметки)…………………………………...24](#Създаване_на_акаунти_сметки)

[4.3.1 Сървърна имплементация......…………………………….25](#AccountController)

[4.3.2 Клиентска имплементация…………….………………….27](#GetAccounts_tsx)

[4.4 Създаване на Категории…………………………………………...29](#Създаване_на_категории)

[4.5 Създаване на Транзакции…………………………………………31](#Създаване_на_транзакции)

[4.5.1 Сървърна имплементация……………………………….31](#TransactionController_cs)

[4.5.2 Клиентска имплементация..………………………………33](#GetTransactions_sx)

[4.6 Интеракция с AI асистеснт……….……………………………….35](#Интеграция_на_AI_асистент_в_проекта)

[4.6.1 Сървърна имплементация..……………………………….36](#FinancialRecommendationController_cs)

[4.6.2 Клиентска имплементация.……………………………….37](#CreateFinancialRecommendations_tsx)

[**Глава 5: Ръководство и използване**……………………………………….42](#Ръководство_за_използване)

[**Заключение**…………………………………………………………………..43](#Заключение)

[**Декларация за авторство**…………………………………………………...45](#ДЕКЛАРАЦИЯ)

[**Използвана литература**……………………………………………………..46](#ИЗПОЛЗВАНА_ЛИТЕРАТУРА)

[**Приложение - Изходен код.............................................................................47**](#Приложение_Изходен_код)

**Въведение**

Необходимостта от ефективно управление на личните финанси става все по-осезаема в условията на динамична икономическа среда, нарастваща инфлация и променящи се източници на доходи. За мнозина, поддържането на финансов баланс и изграждането на устойчиви навици за спестяване и планиране е предизвикателство, особено без ясна структура или система за проследяване. В този контекст технологичните решения предлагат мощни възможности за автоматизация и персонализация на финансовото планиране.

Настоящият проект представя **приложение за управление на лични финанси, интегриращо базиран на изкуствен интелект асистент**(накратко **FinEdge) -** уеб базирано приложение, което комбинира класически подходи за бюджетиране с интеграцията наасистент, целящ да подпомага потребителите чрез персонализирани съвети, базирани на техните реални данни и утвърдени финансови методологии.

Системата е изградена с използване на **React** за клиентската част и **.NET Core Web API** за сървърната логика. Приложението позволява на потребителя да:

* въвежда и категоризира разходи и приходи
* следи финансови потоци чрез визуализации (графики, диаграми)
* настройва месечни бюджети
* анализира навиците си чрез автоматични отчети
* получава **персонализирани препоръки** от AI асистент

Приложението анализира поведението на потребителя спрямо утвърдени модели за управление на лични финанси, като:

* **50/30/20 правило** (50% нужди, 30% желания, 20% спестявания),
* **Zero-based budgeting** (всеки лев има предназначение),
* **Pay-yourself-first** (приоритет на спестяванията).

На всеки потребител му се задава дадената методология, базирано на отговорите на дадени въпроси по време на регистрация. Също така след регистриране, потребителят може да смени задеданата си методология. Алгоритмите обработват исторически данни за разходи и приходи, за да установят текущия профил на потребителя (напр. склонност към свръхразход, нисък дял на спестявания и др.).

Интегрираният AI асистент, базиран на технологии на OpenAI, използва както структурирани данни (финансови транзакции, избрана методология, бюджетни цели), така и поведенчески показатели (честота на разходи по категории, спонтанни покупки), за да генерира смислени и приложими съвети. Примерни съвети могат да включват:

* „Въз основа на последните три месеца, надвишавате лимита за разходи по категория 'Забавление' с 25%. Помислете дали да не го ограничите спрямо бюджета си.“
* „Не сте заделили спестявания този месец. Прилагайки правилото 50/30/20, се препоръчва минимум 20% от нетния ви доход да бъде насочен към спестяване.“
* „Можете да намалите разходите си за храна с 15%, ако преминете към седмично планиране на покупки.“

Препоръките са адаптивни, т.е. AI системата се „учи“ от поведението на потребителя и отчита промените в навиците му, предоставяйки контекстуално релевантни предложения в реално време.

**Глава 1: Цел на проекта**

**Към какъв тип потребители е насочен проекта?**

FinEdge е насочен към хората, които искат да управляват по-ефективно своите лични финанси, като следят приходите и разходите си, създават бюджети и планират бъдещи финансови цели. Приложението предоставя удобен интерфейс и автоматизирани инструменти, които помагат на потребителите да анализират финансовите си навици, да намалят ненужните разходи и да подобрят финансовата си стабилност.

Основната цел на проекта е да създаде интуитивна и достъпна платформа, която комбинира гъвкавост, сигурност и модерни технологии за ефективно управление на личните финанси.

Интегрираният AI асистент допълнително служи за анализ на финансовите навици, предоставяне на персонализирани съвети и помощ при създаването на бюджети и финансови планове. Той може да идентифицира модели на разходи, да предлага оптимизации за спестяване и да отговаря на въпроси, свързани с управлението на личните финанси, спрямо **методологията** на потребителя. Също така може да предупреждава за потенциални финансови рискове и да предлага алтернативни стратегии за постигане на дългосрочни финансови цели.

**Предимства на FinEdge спрямо подобни налични продукти**

FinEdge предлага модерно и интуитивно решение за управление на личните финанси, като комбинира удобен интерфейс, мощни аналитични инструменти и AI асистент. В сравнение с популярните приложения като 1Money, Spendee и Monefy, FinEdge предоставя по-широк набор от функции и адресира някои от основните им недостатъци:

* **Spendee** – Приложението предлага разнообразни функционалности, но не е user-friendly за нови потребители и може да бъде трудно за настройка. Освен това, липсва персонализиран AI асистент, който да помага с анализи и финансови съвети.
* **Monefy** - Има опростен интерфейс, но липсва разширена аналитика и възможност за детайлно бюджетиране. Също така, няма AI асистент.
* **1Money** – Приложението беше една от най-добрите алтернативи, но от ноември 2024 е премахнато от Google Play и Apple App Store, което прави достъпът до него ограничен. Освен това, няма AI асистент.

**Какво прави FinEdge по-добрият избор?**

* **Интуитивен интерфейс**, подходящ както за начинаещи, така и за напреднали потребители.
* **AI асистент**, който анализира разходите и предлага персонализирани финансови съвети.
* **Мощни аналитични инструменти** – диаграми, статистики и детайлни отчети за финансите.
* **Десктоп версия**, която позволява удобен достъп и управление на финансите на по-голям екран.
* **Сигурност и гъвкавост**, с интеграция на модерни технологии за защита на потребителските данни.

FinEdge не само улеснява управлението на личните финанси, но също така предоставя интелигентна подкрепа, която липсва при конкурентите.

**Целева аудитория**

Крайните потребители на FinEdge са хора, които редовно управляват личните си финанси и търсят удобство, бързина и лесен достъп до финансовата си информация.

Те могат бързо и ефективно да следят своите приходи, разходи и бюджети, като това е гарантирано чрез интуитивния интерфейс и интегрирания AI асистент, който предоставя персонализирани анализи и препоръки.

**Глава 2: Технологии и среди за разработка на приложението**

**Използвани Технологии**

**C#**

C# е обектно ориентиран език за програмиране, разработен от Microsoft като част от софтуерната платформа .NET. [[1]](#Ръководство_за_използване_CSharp)

**.NET Core Web API**

**.NET Core Web API** е програмна рамка за създаване на уеб услуги (RESTful APIs), която позволява разработката на мащабируеми и високопроизводителни сървърни приложения. Той предоставя инструменти за обработка на HTTP заявки, управление на бази данни, сигурност и автентикация, което го прави подходящ за модерни уеб и мобилни приложения. [[2]](#Ръководство_за_използване_NetCoreAPI)

**Entity Framework**

Entity Framework е ORM (Object-Relational Mapper) за .NET, който улеснява работата с бази данни, като позволява да се използват C# обекти вместо SQL заявки. Той автоматично преобразува обектите в таблици и обратно, което ускорява разработката и намалява риска от грешки при работа с данни. [[3]](#Ръководство_за_използване_EntityFramewor)

**JWT (JSON Web Token)**

JSON Web Token е интернет стандарт за създаване на данни с незадължителен подпис и/или незадължително криптиране, чийто полезен товар (payload) съдържа JSON. Токените се подписват или с помощта на частна тайна, или с публичен/личен ключ. [[4]](#Ръководство_за_използване_JWT)

**Swashbuckle**

**Swashbuckle** е библиотека за автоматично генериране на Swagger документация за ASP.NET API-та. Позволява създаване на интерактивна документация, чрез която API-тата могат лесно да бъдат тествани. [[5]](#Ръководство_за_използване_Swashbuckle)

**Hangfire**

Hangfire е библиотека за изпълнение на фонови задачи (background jobs) в .NET. Тя позволява лесно планиране, отложено изпълнение и периодично стартиране на задачи. [[6]](#Ръководство_за_използване_Hangfire)

**OpenAI Developer API**

Този пакет осигурява достъп до **OpenAI API**, което позволява използване на модели като GPT-4.o за обработка и анализ на данни, автоматизация и други. [[7]](#Ръководство_за_използване_OpenAIAPI)

**Vite**

Vite е локален сървър за разработка на клиентската част на приложения и използван за шаблони на проекти на React, Vue, Svelte и други. [[8]](#Ръководство_за_използване_Vite)

**TypeScript**

TypeScript е език за програмиране на високо ниво с отворен код, разработен от Microsoft, който добавя статично въвеждане с незадължителни анотации за тип към JavaScript. Той е предназначен за разработване на големи приложения и се превежда в JavaScript. [[9]](#Ръководство_за_използване_TypeScript)

**React**

React е библиотека с отворен код, която има за цел да направи изграждането на потребителски интерфейси, базирани на компоненти, по-безпроблемно. Поддържа се от Meta и от отделни разработчици и компании. [[10]](#Ръководство_за_използване_React)

**React Router DOM**

React Router е библиотека за маршрутизация в React, която позволява създаване на навигация между различни страници в приложението. [[11]](#Ръководство_за_използване_ReactRouterDOM)

**TanStack Query (React Query)**

TanStack Query (известен преди като React Query) е библиотека за извличане на данни за уеб приложения. Библиотеката прави извличането, кеширането, синхронизирането и актуализирането на състоянието на сървъра във уеб приложения по-лесно. [[12]](#Ръководство_за_използване_ReactQuery)

**Axios**

Axios е базиран на обещания (promises) HTTP клиент за node.js и браузъра. Той е изоморфен (може да работи в браузъра и nodejs със същата кодова база). От страна на сървъра той използва http модул node.js, докато на клиента (браузър) използва XMLHttpRequests. [[13]](#Ръководство_за_използване_Axios)

**KendoReact**

KendoReact е UI библиотека от със 120+ компонента, проектирана и създадена за разработване на бизнес приложения с React. [[14]](#Ръководство_за_използване_KendoReact)

**Tailwind**

Tailwind CSS е CSS рамка (framework) с отворен код. Тя създава списък от полезни CSS класове, които могат да се използват за стилизиране на всеки елемент чрез смесване и съпоставяне. [[15]](#Ръководство_за_използване_Tailwind)

**Среди за разработка**

**Visual Studio**

Visual Studio е интегрирана среда за разработка (IDE), разработена от Microsoft. Използва платформи за разработка на софтуер на Microsoft, включително Windows API, Windows Forms, Windows Presentation Foundation (WPF), Microsoft Store, Microsoft Silverlight и други. [[16]](#Ръководство_за_използване_VisualStudio)

**Visual Studio Code**

Visual Studio Code е модерна, лека и мощна интегрирана среда за разработка създадена от Microsoft. Средата за разработка е налична за всички операционни системи - Linux, Windows и macOS. [[17]](#Ръководство_за_използване_VisualStudioCo)

**SQL Server Management Studio (SSMS)**

SQL Server Management Studio (SSMS) е софтуерно приложение, което се използва за конфигуриране, управление и администриране на всички компоненти в SQL Server. [[18]](#Ръководство_за_използване_SSMS)

**Node.js**

Node.js е междуплатформена среда за изпълнение на JavaScript с отворен код, която може да работи на Windows, Linux, Unix, macOS и др. [[19]](#Ръководство_за_използване_NodeJS)

**Глава 3: Анализ на изискванията към проекта**

**Функционални изисквания към проекта**

**Публична част**

Чрез публичната част, клиентите трябва да имат възможност за:

* Създаване/Влизане в своя профил и редакция на данните в него
* Създаване/Редакция/Филтриране на Акаунти/Категории/Транзакции
* Визуализация на създадените Акаунти/Категории/Транзакции
* Интеракция с AI асистент (AIPrompt), който да дава персонализирани съвети към потребителя, спрямо неговите данни, навици за харчене и избрана финансова методология
* Да не се затруднява с използването на приложението (приложението трябва да има разбираем UI/UX)
* Да може да ползва приложението и на телефона си (приложението трябва да е mobile responsive)

**Нефункционални изисквания към проекта**

**Сигурност**

Системата трябва да предоставя надеждни методи за удостоверяване и контрол на достъпа за потребители. Данни като пароли следва да се съхраняват във вид на хеширани стойности.

**Производителност**

Бързото зареждане на страниците и ефективната обработка на данни, дори при високо натоварване, осигуряват качествено и приятно потребителско изживяване.

**Поддръжка**

Кодът и архитектурата на платформата трябва да бъдат ясно документирани и структурирани, с цел да се улесни поддръжката и бъдещите актуализации от страна на разработчиците.

**Употреба и достъп**

Потребителският интерфейс трябва да бъде интуитивен и удобен за работа, с ясно разположени функции и лесна навигация. Приложението следва да бъде оптимизирано за коректна работа на различни устройства и браузъри.

**Съвместимост**

Системата трябва да поддържа лесна интеграция с външни модули/библиотеки/API като React Router DOM, KendoReact и OpenAI API. Освен това, тя трябва да позволява лесен начин за разработчика да обновява добавените пакети, за да може винаги да ползва най-актуалните им версии.

**Структура на базата данни**

Изборът на Microsoft SQL Server, управляван чрез SQL Server Management Studio (SSMS), пред други релационни бази данни като MySQL или PostgreSQL, е поради няколко ключови предимства, които го правят особено подходящ за бизнес и академични приложения, изискващи висока производителност, сигурност и интеграция с .NET екосистемата.

SQL Server предлага пълна поддръжка на ACID (Atomicity, Consistency, Isolation, Durability) транзакции, което гарантира надеждността и целостта на данните – изключително важно при финансови приложения. Също така, платформата осигурява вградени механизми за сигурност, мащабируемост и удобство при работа със сложни заявки, тригери, съхранени процедури и изгледи.

Допълнително предимство е тясната интеграция с .NET и Entity Framework, което улеснява изграждането на сървърната логиката и осигурява безпроблемна комуникация между приложението и базата данни.

SQL Server Management Studio предоставя интуитивен интерфейс за разработка, тестване и администриране на бази данни, което ускорява процеса на разработка и поддръжка на FinEdge.

На фигура 1 са показани връзките между таблиците на приложението:

A computer screen shot of a computer

AI-generated content may be incorrect.

**Фигура 1.** Връзки между таблиците от базата данни на приложението

В основата на приложението стои релационна база данни, която съхранява информация за потребителите и техните финансови дейности. Основната таблица Users съдържа данни за потребителите - име, имейл, парола, общ баланс и методология. Всяко действие и обект в системата е свързан с конкретен потребител чрез UserID.

Таблицата Accounts (Акаунти) съхранява информация за различните сметки на потребителя – банкови, парични, инвестиционни и др. Към всяка сметка могат да бъдат асоциирани множество Transactions (Транзакции), които описват конкретни финансови операции – разходи, приходи, повтарящи се плащания и др. Всяка транзакция е свързана едновременно със сметка и категория.

Categories (Категории) дефинират различните типове разходи и приходи (например "Храна", "Заплата", "Транспорт"), а към тях могат да бъдат създавани подразделения в таблицата Subcategories (Събкатегории).

Notifications (Нотификации) са предназначени за известяване на потребителя – например за важни събития или препоръки.

Финансови препоръки (FinancialRecommendations) съхранява съвети и анализи, предоставени от AI асистента, като всеки запис съдържа заглавие и отговор на конкретна заявка.

За сигурността на автентикациягьа е включена таблицата RefreshTokens (Рефреш токени), където се пазят активните токени на потребителите, с информация дали са оттеглени и докога са валидни.

Цялостната структура е организирана с ясно дефинирани зависимости и външни ключове, което осигурява стабилна, надеждна и разширяема основа за управлението на личните финанси в приложението FinEdge.

**Архитектура на проекта**

При първоначалното проектиране на приложение, от изключително значение е да се вземе под внимание дали се предвиждат интеграции с външни системи. На фигура 2 е представена контекстната архитектура на приложението.

A diagram of software components

AI-generated content may be incorrect.

**Фигура 2.** Контекстнаархитектура на приложението

**Интеграцията с OpenAI API и LLM (Large Language Model)**

За да бъде успешно интегрирана услугата на OpenAI, трябва да бъдат разгледани основните й елементи и как те взаимодействат помежду си в детайли. Това включва конфигурация на API клиент, обработка на заявки, управление и отговори. На фигура 3 е показан процеса при създаване на препоръкити, генерирани от изкуствен интелект.

A diagram of a software application

AI-generated content may be incorrect.

**Фигура 3.** Процес при създаване на препоръкити, генерирани от изкуствен интелект

Стъпки за създаване на AI генерирани препоръки

1. Клиентът изпраща заявка с въведен prompt и избрана дата.
2. Сървърът получава заявката и взима данните на текущия потребител.
3. Тези данни се сериализират и се комбинират с допълнителни инструкции.
4. Сървърът изпраща съобщението към OpenAI API.
5. LLM връща отговор – кратка финансова препоръка.
6. Сървърът записва препоръката в базата и изпраща отговора обратно към клиента.

**Глава 4: Техническа реализация**

**Структура на проекта**

Приложението се състои от няколко главни директории:

* **FinEdgeFrontend**
* **FinEdgeBackend**
* **FinEdgeData**
* **FinEdgeServices**
* **FinEdgeTests**

**FinEdgeFrontend**

Директорията съдържа папки и файлове, които се отнасят за клиентската част на приложението.

**node\_modules -** Тази папка съдържа всички зависимости на проекта, които са инсталирани чрез npm или yarn. Това включва библиотеки като React, React Router, KendoReact и други, които проектът използва.

**src -** Това е основната папка, където се намира клиентската част от кода на проекта. Тук се съдържат компонентите на React, логиката за рендериране на страниците, и други важни части от приложението:

* **Components -** Тази папка съдържа отделни React компоненти, които могат да бъдат използвани на различни места в приложението. Всеки компонент е TypeScript файл, който позволява използването на JSX синтаксис (TSX) за визуализация на част от интерфейса.
* **Hooks -** В тази папка се намират custom React hooks. Те са функции, които позволяват да използваме React State и други функционалности в компонентите без употребата на класове. Основно куките са създадени за взимане на информацията от сървъра
* **Pages** - В тази папка се съдържат компонентите, които съответстват на различни страници в приложението.
* **Utils** - Тази папка съдържа помощни функции, endpoints за връзка със сървъра и TypeScript типове, които се използват в различни части на приложението. Те могат да включват функции за форматиране на данни, валидиране на входни данни, манипулиране с дати и т.н. Пример: formatDate.js, validateInput.js.

A screenshot of a computer program

AI-generated content may be incorrect.Останалата част от файловете за автоматично генерирани от Vite и служат основно за конфигуриране на клиентската част на проекта. На фигура 4 е представена файловата структура на frontend часста от проекта:

**Фигура 4.** Файлова структура

**FinEdgeBackend**

Tова е главният проект на приложението, отговорен за стартирането на API-то и приемането на HTTP заявки:

* **Controllers** - Съдържа контролери, които обработват входящи HTTP заявки (GET, POST, PUT, DELETE). Всеки контролер е свързан с определен ресурс или логическа функционалност и използва съответни методи за отговор.
* **Program.cs** - Конфигурира приложението, маршрутизацията и всички компоненти.

Този проект служи като входна точка на приложението и свързва останалите два слоя – услугите и достъпа до данни.

На фигура 5 е представена файловата структура:

A screenshot of a computer program

AI-generated content may be incorrect.

**Фигура 5. Файлова структура**

**FinEdgeData**

Този проект служи на достъпа до данни и споделените модели, използвани в приложението:

* **Data** - Съдържа конфигурации за връзка с базата данни чрез Entity Framework Core.
* **DTOs** - Data Transfer Objects (Обекти за пренос на данни). Използват се за пренасяне на данни между слоевете, без да съдържат бизнес логика.
* **Enums** - Дефинират изброими типове (enum), използвани за фиксирани стойности като статуси, роли и други.
* **Migrations** - Съдържа автоматично генерирани файлове за миграции на базата данни.
* **Models** - Съдържа домейн моделите, които отразяват структурата на таблиците в базата данни.

На фигура 6 е представена файловата структура:

A screenshot of a computer

AI-generated content may be incorrect.

**Фигура 6. Файлова структура**

**FinEdgeServices**

Този проект реализира бизнес логиката и осигурява абстракции чрез интерфейси.

* **Interfaces** - Съдържа интерфейси, които дефинират бизнес операциите. Те осигуряват абстракция между контролерите и конкретните имплементации.
* **Services** - Съдържа конкретните имплементации на услугите (бизнес логика). Тук се извършват валидации, обработка на данни, извикване на репозитории и т.н.

На фигура 7 е представена файловата структура:

A screenshot of a computer

AI-generated content may be incorrect.

**Фигура 7.** Файлова структура

**FinEdgeTests**

Този проект е създаден с цел юнит тестване на различните контролери от FinEdgeBackend, и по-специално е насочен към проверка на логиката, която се изпълнява в тях, без да се включват реални зависимости като бази данни или външни услуги.

Използван е FakeItEasy – библиотека за създаване на фалшиви обекти, която прави лесно създаването на зависимости. За сървърната част на приложението има направени 56 теста.

На фигура 8 е представена файловата структура:

A screenshot of a computer

AI-generated content may be incorrect.

**Фигура 8.** Файлова структура

Може да разгледате пълния код как са имплементирани тестове на следния линк:

* <https://github.com/P1l3T0/FinEdge/tree/master/FinEdgeTests/Controller>

**Създаване на потребители и потребителски профили**

За да могат потребителите да се регистрират и виждат своите профили е необходимо първо да се създаде инстанция на User класа. На фигура 9 е показана последователността на действия, които се извършват като потребител се регистрира.

A screenshot of a computer screen

AI-generated content may be incorrect.

Фигура 9. Последователност от действия за регистрацията на нов потребител

Може да разгледате пълния код на/за това как е имплементирана регистрацията на следните линкове:

* <https://github.com/P1l3T0/FinEdge/blob/master/FinEdgeFrontend/src/Components/HomePage/Cards/UserProfile.tsx#L8> – Клиентска част
* <https://github.com/P1l3T0/FinEdge/blob/master/FinEdgeBackend/Controllers/AuthenticationController.cs#L12> – Сървърна част

**Създаване на акаунти (сметки)**

След регистрацията, потребителите могат да си създават свои сметки (акаунти), за които си задават име, текущ баланс, тип и цвят. След създаването му се появява “карта” с информацията за акаунта. Потребителят по всяко време може да актуалицира/изтрие своите акаунти.

**AccountController.cs**

**[HttpPost]**

**[Route("create")]**

**public async Task<IActionResult> CreateAccount([FromBody] AccountDTO accountDto)**

**{**

**User currentUser = await \_userService.GetCurrentUserAsync();**

**if (!\_accountService.Validate(accountDto))**

**{**

**...**

**return BadRequest();**

**}**

**Account account = await \_accountService.GetAccountForCurrentUserByNameAsync(accountDto.Name!, currentUser);**

**if (account is not null)**

**{**

**...**

**return BadRequest();**

**}**

**currentUser.TotalBalance += accountDto.Balance;**

**await \_accountService.CreateAccountAsync(new Account()**

**{**

**UserID = currentUser.ID,**

**User = currentUser,**

**Name = accountDto.Name,**

**Balance = accountDto.Balance,**

**Currency = accountDto.Currency,**

**Color = accountDto.Color,**

**DateCreated = DateTime.Now,**

**AccountType = (AccountType)Enum.Parse(typeof(AccountType), accountDto.AccountType!),**

**});**

**await \_notificationService.CreateNotificationAsync(new Notification()**

**{**

**Title = $"Successfully created Account '{accountDto!.Name}'!",**

**Description = $"Your account '{accountDto!.Name}' has been successfully created and is now ready to use.",**

**NotificationType = NotificationType.Success,**

**IsRead = false,**

**User = currentUser,**

**UserID = currentUser.ID**

**});**

**return Ok(account);**

**}**

Методът CreateAccount позволява на потребителя да създаде нова финансова сметка (Account). Първо се извлича текущият потребител чрез ***\_userService.GetCurrentUserAsync()***, след което се валидират въведените от него данни.

Ако данните не са коректни (напр. липсва име или балансът е отрицателен), се създава уведомление за грешка и се връща BadRequest.

След това се проверява дали вече съществува сметка със същото име за този потребител. Ако има, отново се изпраща уведомление за грешка и заявката се прекратява.

При успешно преминаване на валидацията, балансът на потребителя се увеличава, създава се новата сметка, и се изпраща уведомление за успешно създаване.

**GetAccounts.tsx**

**const GetAccounts = () => {**

**const [filter, setFilter] = useState<CompositeFilterDescriptor>({**

**logic: "and",**

**filters: [],**

**});**

**const { data, isLoading, isError, error } = useGetAccounts();**

**const handleFilterChange = (*newFilter*: CompositeFilterDescriptor) => {**

**setFilter(*newFilter*);**

**};**

**if (isLoading) return <p>Loading...</p>;**

**if (isError) return <p>Error: {error!.message}</p>;**

**return (**

**<>**

**<AccountFilter *onFilterChange*={handleFilterChange} /> <br />**

**<div *className*="flex flex-col lg:flex-row gap-4">**

**<div *className*="flex-grow">**

**<div *className*="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-5 gap-4">**

**<AccountCards *accounts*={filterBy(data!, filter)} />**

**</div>**

**</div>**

**</div>**

**</>**

**);**

**};**

**export default GetAccounts;**

Компонентът **GetAccounts** отговаря за визуализирането на всички създадени сметки на потребителя. Използва се custom hook (useGetAccounts) за извличане на данните от сървъра, като междувременно се показва съобщение за зареждане или грешка, ако е необходимо.

Компонентът също така включва функционалност за филтриране чрез **AccountFilter**, като избраните филтри се съхраняват в локалното състояние. След това сметките се визуализират в мрежа чрез AccountCards, като се прилагат избраните филтри с помощта на **filterBy**.

A screenshot of a computer screen

AI-generated content may be incorrect.На фигура 11 е показана последователността на действия, които се извършват когато потребителят си създаде акаунт.

**Фигура 11**. Последователност от действия при създаването на акаунт

На **фигура 12** е показано визуално как изглежда страницата **Акаунти.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Фигура 12**. Страница **Акаунти**

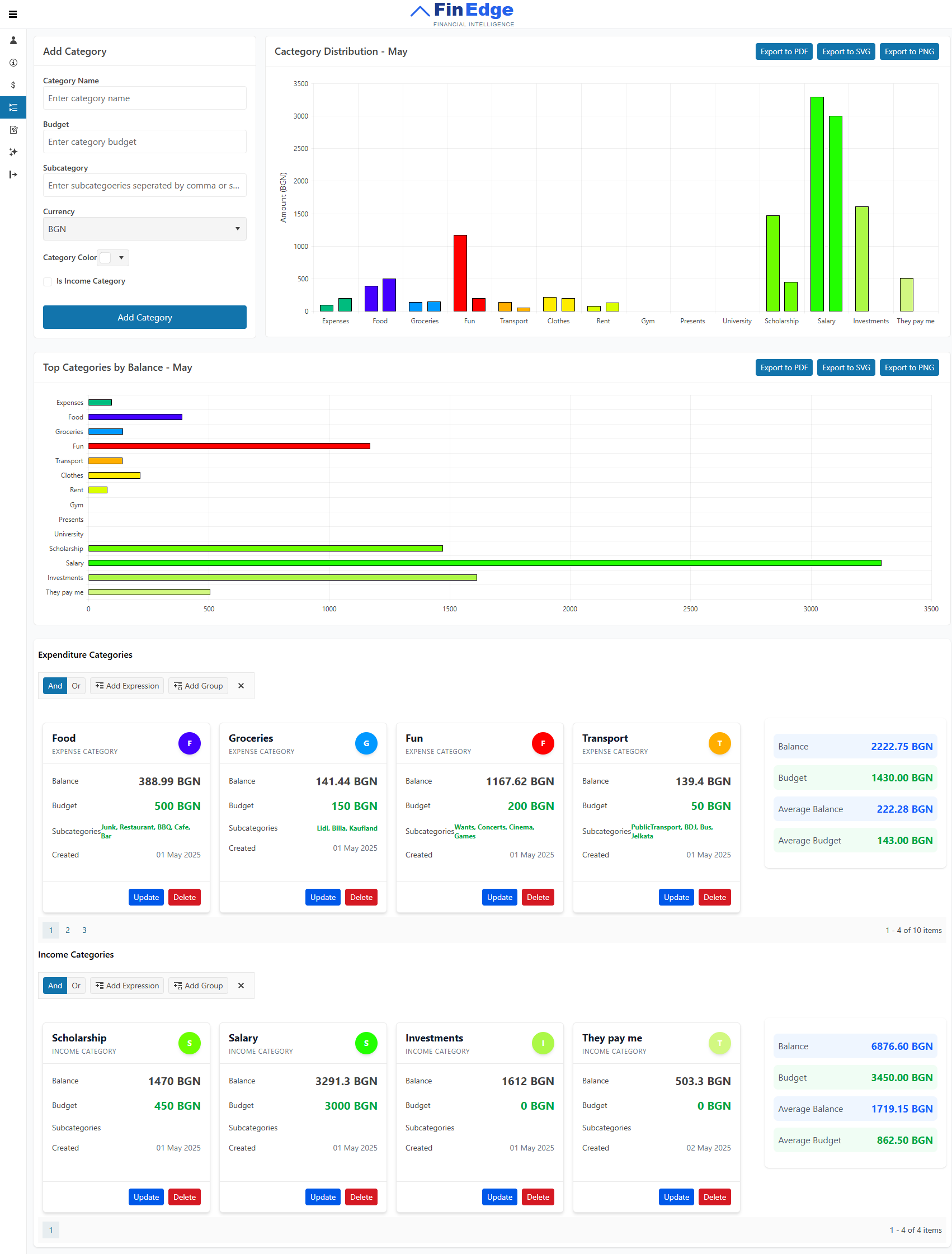
**Създаване на категории**

След като създаде акаунт, потребителят може да навигира към страницата “Категории” и да си създаде съответната категория, за която си задават име, текущ бюджет, цвят и дали е доходоносна.

Създаването на Категории е аналогично също като създаването на Акаунти, кодът за имплементация е подобен на Акаунтите и може да бъде разгледан на слидните линкове:

* <https://github.com/P1l3T0/FinEdge/blob/master/FinEdgeBackend/Controllers/CategoryController.cs#L19> – Клиентска част
* <https://github.com/P1l3T0/FinEdge/blob/master/FinEdgeFrontend/src/Components/CategoriesPage/CRUD/IncomeCategories.tsx#L8> – Сървърна част

На фигура 13 е показано визуално как изглежда страницата **Категории.**



**Фигура 13**. Страница **Категории.**

**Създаване на транзакции**

След като създаде акаунт и категории, потребителят може да навигира към страницата “Транзакции” и да си създаде нова транзакция, за която си задават име, сума, към кой акаунт/категория принадлежи и дали е повтаряща се.

**TransactionController.cs**

**[HttpPost]**

**[Route("create")]**

**public async Task<IActionResult> CreateTransaction([FromBody] TransactionDTO transactionDto)**

**{**

**User currentUser = await \_userService.GetCurrentUserAsync();**

**if (!\_transactionService.Validate(transactionDto))**

**{**

**await \_notificationService.CreateNotificationAsync(new Notification()**

**{**

**Title = "Please fill in the Transaction fields!",**

**Description = $"The Transaction with name '{transactionDto.Name}' must have valid properties",**

**NotificationType = NotificationType.Error,**

**IsRead = false,**

**User = currentUser,**

**UserID = currentUser.ID**

**});**

**return BadRequest();**

**}**

**Category category = await \_categoryService.GetCategoryForCurrentUserByNameAsync(transactionDto.CategoryName!, currentUser);**

**Account account = await \_accountService.GetAccountForCurrentUserByNameAsync(transactionDto.AccountName!, currentUser);**

**await \_transactionService.UpdateUserBalanceAsync(true, transactionDto, null, currentUser, category, account);**

**await \_transactionService.CreateTransactionAsync(new Transaction()**

**{**

**Name = transactionDto.Name,**

**Amount = transactionDto.Amount,**

**UserID = currentUser.ID,**

**User = currentUser,**

**AccountID = account.ID,**

**AccountName = account.Name,**

**Account = account,**

**CategoryID = category.ID,**

**Category = category,**

**CategoryName = category.Name,**

**Color = category.Color,**

**IsRepeating = transactionDto.IsRepeating,**

**NextRepeatDate = transactionDto.IsRepeating ? DateTime.UtcNow.AddMonths(1) : null**

**});**

**await \_notificationService.CreateNotificationAsync(new Notification()**

**{**

**Title = $"Transaction {transactionDto.Name} created successffuly",**

**Description = $"Your transaction '{transactionDto.Name}' has been successfully created!",**

**NotificationType = NotificationType.Success,**

**IsRead = false,**

**User = currentUser,**

**UserID = currentUser.ID**

**});**

**return Created();**

Методът **CreateTransaction** се използва за създаване на нова транзакция от потребителея. Първо се извършва валидация на подадените данни чрез ***\_transactionService***, като при невалидна транзакция се връща грешка и се създава съответното известие. След това се извличат категорията и акаунтът по име, към които ще бъде обвързана транзакцията.

След успешно извличане се актуализира общият баланс на потребителя чрез метода ***UpdateUserBalanceAsync***, и се създава нов обект от тип **Transaction**, който се записва в базата данни.

Ако транзакцията е отбелязана като повтаряща се, автоматично се задава дата за следващо повторение (същата дата от следващия месец).

**GetTransactions.tsx**

**const GetTransactions = () => {**

**const [filter, setFilter] = useState<CompositeFilterDescriptor>({**

**logic: "and",**

**filters: [],**

**});**

**const { data, isLoading, isError, error } = useGetTransactions();**

**const handleFilterChange = (*newFilter*: CompositeFilterDescriptor) => {**

**setFilter(*newFilter*);**

**};**

**if (isLoading) return <p>Loading...</p>;**

**if (isError) return <p>Error: {error?.message}</p>;**

**return (**

**<>**

**<TransactionFilter *onFilterChange*={handleFilterChange} />**

**<div *className*="flex flex-col lg:flex-row gap-4">**

**<div *className*="flex-grow">**

**<div *className*="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-5 gap-4">**

**<TransactionCards *transactions*={filterBy(data?.incomeTransactions!, filter)} />**

**<TransactionCards *transactions*={filterBy(data?.expenditureTransactions!, filter)} />**

**</div>**

**</div>**

**</div>**

**</>**

**);**

**};**

**export default GetTransactions;**

Компонентът **GetTransactions** се използва за визуализиране на всички транзакции, регистрирани от потребителя. Данните се зареждат чрез ***useGetTransactions*** *hook-a*.

Потребителят може да прилага филтри чрез **TransactionFilter**, а резултатите се визуализират с компонента **TransactionCards**, като се разделят по тип – приходи (incomeTransactions) и разходи (expenditureTransactions).

Може да разгледате пълния код как са имплементирани транзакциите тук:

* <https://github.com/P1l3T0/FinEdge/blob/master/FinEdgeFrontend/src/Components/TransactionsPage/CRUD/GetTransactions.tsx#L7> - Клиентска част
* <https://github.com/P1l3T0/FinEdge/blob/master/FinEdgeBackend/Controllers/TransactionController.cs#L21> – Сървърна част

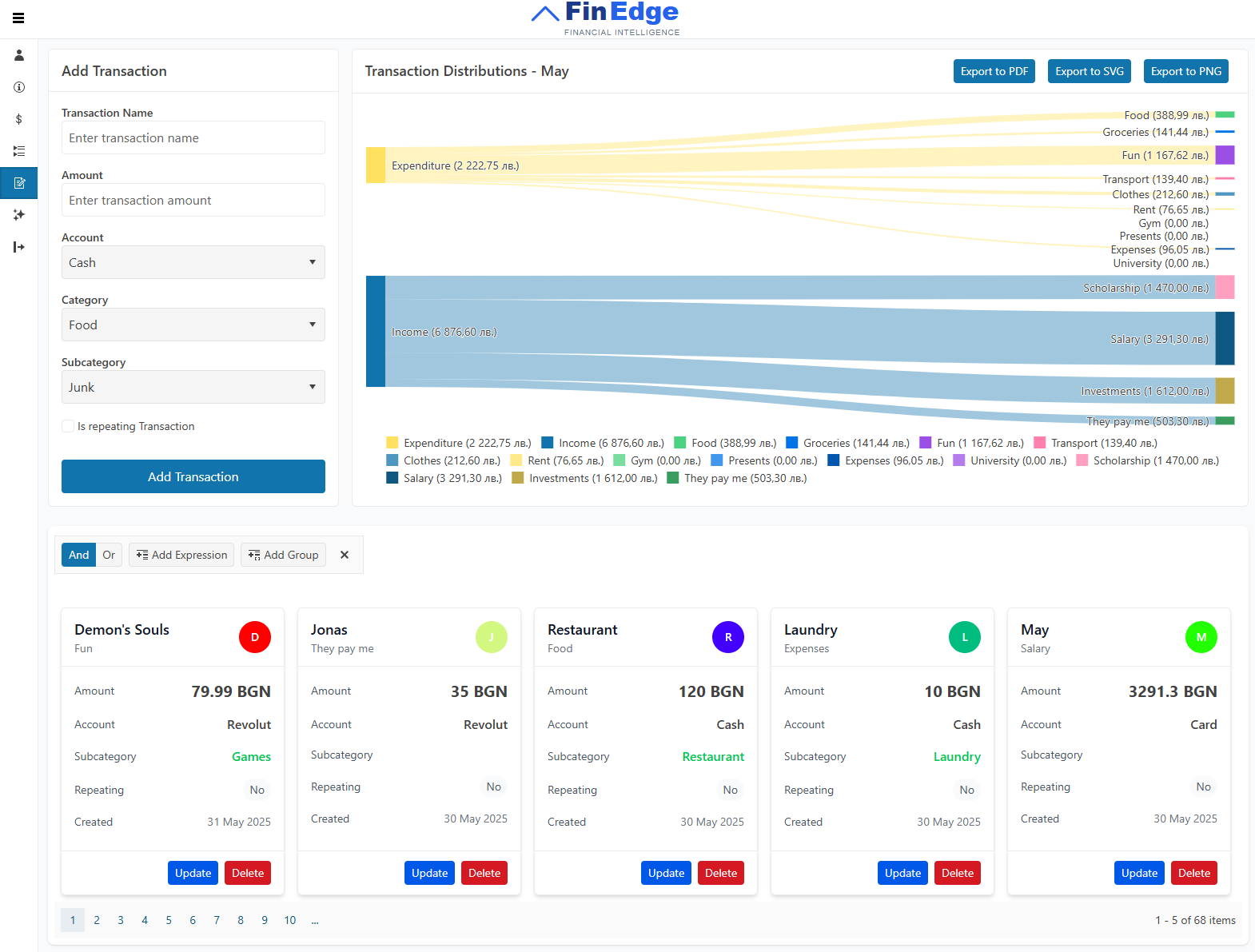
На фигура 14 е показана последователността на действия, които се извършват когато потребителят си създаде акаунт.

A screenshot of a diagram

AI-generated content may be incorrect.

**Фигура 14** . Последователност от действия при създаването на транзакция

На фигура 15 е показано визуално как изглежда страницата **Транзакции**.



**Фигура 15**. Страница **Транзакции.**

**Интеграция на AI асистент**

В проекта е реализирана AI функционалност, която предоставя финансови препоръки на потребителите въз основа на техните данни. За целта е използван GPT модел чрез външно API, като интеграцията е реализирана в два основни етапа:

* Сървърът подготвя потребителските данни и правила, генерира prompt и го изпраща към GPT4-о, след което връща отговора обратно.
* Потребителят въвежда въпрос чрез интерфейс, получава отговор от AI асистента и го вижда във визуализиран формат.

**FinancialRecommendationController.cs**

**ICollection<Transaction> transactions = \_transactionService.GetTransactionsFromSpecifiedDate(currentUser.Transactions, parsedDate);**

**JsonSerializerOptions options = new JsonSerializerOptions { WriteIndented = true };**

**string accountsJson = JsonSerializer.Serialize(currentUser.Accounts, options);**

**string categoriesJson = JsonSerializer.Serialize(currentUser.Categories, options);**

**string transactionsJson = JsonSerializer.Serialize(transactions, options);**

**string allData = string.Join(",", accountsJson, categoriesJson, transactionsJson);**

**string rules = @$"I want your response to always be in the scope of the project, that is Financial Recommendations, any other out-of-scope prompts need to turneed down with a sentence like 'This request is out of scrope for me',**

**start with something like 'Recommendations based on data from {parsedDate.ToShortDateString()} to today (or similar sentences).";**

**string textTweaks = @$"Please respond in short plain text (maximum of 3-4 sentences only), without any formatting, such as bold text, dashes, slashes, numbering, ordered/unordered list, or special characters.";**

**string prompt = string.Join(" ", promptRequestData.Prompt!, allData, rules, textTweaks).Trim();**

**GPTResponseDTO gPTResponse = await \_gPTService.Ask(prompt, currentUser);**

От транзакциите на текущия потребител се взимат само тези, направени след определена дата. След това акаунтите, категориите и транзакциите се сериализират в JSON. Към тези данни се добавят инструкции (като например отговорите да бъдат кратки и свързани само с финансови препоръки). Всичко това се комбинира в един prompt, който по-късно ще бъде подаден към GPT4-о.

**GPTService.cs**

**private readonly IConfiguration \_configuration = configuration;**

**public async Task<GPTResponseDTO> Ask(string prompt, User currentUser)**

**{**

**string apiKey = \_configuration.GetSection("Appsettings:OpenAIAPIKEY").Value!;**

**string model = \_configuration.GetSection("Appsettings:Model").Value!;**

**ChatClient chatClient = new ChatClient(model, apiKey);**

**ClientResult<ChatCompletion> completion = await chatClient.CompleteChatAsync(prompt);**

**string response = completion.Value.Content[0].Text.Trim();**

**return new GPTResponseDTO**

**{**

**Response = response**

**};**

**}**

От конфигурационен файл се взимат API ключът и моделът. С тях се създава ChatClient, който подава prompt-а към GPT. Полученият отговор се връща като обект от тип GPTResponseDTO, съдържащ текста на генерирания отговор.

**CreateFinancialRecommendations.tsx**

**const CreateFinancialRecommendations = () => {**

**const queryClient = useQueryClient();**

**const [promptRequestData, setPromptRequestData] = useState<PromptRequestData>({**

**prompt: "",**

**dateString: `${new Date().getDate()}-${new Date().getMonth() + 1}-${new Date().getFullYear()}`**

**});**

**const handleDateChange = (*e*: DatePickerChangeEvent) => {**

**const dateString: string = `${*e*.value!.getDate()}-${*e*.value!.getMonth() + 1}-${*e*.value!.getFullYear()}`;**

**setPromptRequestData({**

**...promptRequestData,**

**dateString: dateString**

**});**

**}**

**const createRecommendations = async () => {**

**const response = await axios**

**.post(createFinancialRecommendationEndPoint, promptRequestData, { withCredentials: true })**

**.catch((*error*: AxiosError) => {});**

**return response?.data as FinancialRecommendation;**

**}**

**const { mutateAsync } = useMutation({**

**mutationFn: createRecommendations,**

**onSuccess: () => {**

**queryClient.invalidateQueries({ queryKey: ["recommendations"] });**

**},**

**});**

**const generateFinancialRecommendation = async (*prompt*: string): Promise<FinancialRecommendation> => {**

**setPromptRequestData({**

**...promptRequestData,**

**prompt: *prompt*,**

**});**

**const newRecommendation = await mutateAsync();**

**return newRecommendation;**

**}**

**return (**

**<>**

**<div *className*="mb-3">**

**<DatePicker *width*={200} *max*={new Date()} *onChange*={handleDateChange} *className*='m-6' />**

**</div>**

**<div>**

**<AIPromptComponent *generateFinancialRecommendation*={generateFinancialRecommendation} />**

**</div>**

**</>**

**)**

**}**

**export default CreateFinancialRecommendations;**

**CreateFinancialRecommendations.tsx** служи като родителски компонент, който управлява логиката по създаване на AI препоръки. Той пази локално състояние с дата и prompt, позволява избор на дата чрез DatePicker и при натискане на бутон или избор на предложение, изпраща заявка към backend-а чрез **axios**. След успешна заявка, автоматично се обновяват свързаните данни с **useQueryClient**.

**AIPromptComponent.tsx**

**const AIPromptComponent = ({ *generateFinancialRecommendation* }: { generateFinancialRecommendation: (*prompt*: string) => Promise<FinancialRecommendation> }) => {**

**const { data, isLoading, isError, error } = useGetPromptSuggestions();**

**const [activeView, setActiveView] = useState<string>(promptViewDefaults.name);**

**const [outputs, setOutputs] = useState<AIPromptOutputInterface[]>([]);**

**const handleActiveViewChange = (*viewName*: string) => {**

**setActiveView(*viewName*);**

**}**

**const handlePromptRequest = (*prompt*?: string) => {**

**if (!*prompt*) {**

**return;**

**}**

**const recommendation = generateFinancialRecommendation(*prompt*);**

**recommendation.then((*data*) => {**

**setOutputs([{**

**id: *data*.id,**

**title: *prompt*,**

**responseContent: *data*?.responseContent!,**

**prompt,**

**},**

**...outputs,**

**]);**

**setActiveView(outputViewDefaults.name);**

**}).catch((*error*) => {**

**console.log(*error*);**

**});**

**}**

**if (isLoading) return <p>Loading...</p>;**

**if (isError) return <p>Error: {error!.message}</p>;**

**return (**

**<>**

**<AIPrompt *className*="px-5 w-70 sm:w-90 md:w-139 shadow-sm" *activeView*={activeView} *onActiveViewChange*={handleActiveViewChange} *onPromptRequest*={handlePromptRequest}>**

**<AIPromptView *promptSuggestions*={data} />**

**<AIPromptOutputView *outputs*={outputs} *showOutputRating*={true} />**

**</AIPrompt>**

**</>**

**);**

**}**

**export default AIPromptComponent;**

**AIPromptComponent.tsx** е отговорен за визуалната част на AI асистента. Той показва списък с предложени prompt-и, позволява потребителят да избере или въведе свой, изпраща го към родителския компонент чрез **generateFinancialRecommendation**, и след това визуализира получения отговор под формата на изход, който може да се оценява или преглежда повторно.

На фигура 16 е представена последователност от действия при интеракция на AIPrompt компонент с OpenAI API ва страницата Financial Recommendation

A diagram of a software program

AI-generated content may be incorrect.

**Фигура 16.** Последователност от действия при интеракция на AIPrompt компонент с OpenAI API ва страницата Financial Recommendation

На фигура 17 е показана визуализация на AI асистента (AIPrompt).

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

**Фигура 17.** Визуализация на AI асистента

Може да разгледате пълния код как е имплементиран AI асистента тук:

* <https://github.com/P1l3T0/FinEdge/blob/master/FinEdgeFrontend/src/Components/RecommendationsPage/CreateFinancialRecommendations.tsx#L9> – Клиентска част
* <https://github.com/P1l3T0/FinEdge/blob/master/FinEdgeFrontend/src/Components/RecommendationsPage/AIPromptComponent.tsx#L6> - Клиентска част
* <https://github.com/P1l3T0/FinEdge/blob/master/FinEdgeBackend/Controllers/FinancialRecommendationController.cs#L13> - Сървърна част

**Глава 5: Ръководство за използване**

**Стъпки за локална инсталация**

Предварително трябва да имате инсталирани [SQL Server Management Studio](https://learn.microsoft.com/en-us/ssms/download-sql-server-management-studio-ssms), [.NET 9.0](https://dotnet.microsoft.com/en-us/download) и [Vite](https://vite.dev/guide/), и да имате [API key за OpenAI](https://www.youtube.com/watch?v=SzPE_AE0eEo)

1. Направете си локално копие на хранилището (repositoy) от GitHub (<https://github.com/P1l3T0/FinEdge>)
2. Отворете FinEdge.sln файла и стартирайте проекта, като първо го построите (Build - F6), за да инсталирате нужните Nuget пакети
3. Направете локална база данни в SQL Server Management Studio и ѝ вземете connection string-a и го сложете в appsettings.json в DefaultConnection (подробно упътване за инсталация може да се намери на посочения линк - <https://www.youtube.com/watch?v=EmV_IBYIlyo&list=PL82C6-O4XrHdiS10BLh23x71ve9mQCln0видео> от 1:00 до 5:20 ако се затруднявате със създаването на базата данни)
4. Отворете Package Manager конзолата и напишете Add-Migration initialCreate, изчакайте да се създадат миграциите и след това напишете Update-Database
5. За да ползвате AI интеграцията, навигирайте към **.env** файла и сложете вашия OpenAI API ключ на съответното място
6. Отворете основната директория във Visual Studio Code (може да го направите от терминала/powershell/git bash)
7. Влезте в папката FinEdgeFrontend (cd FinEdgeFrontend) и напишете **npm i**, за да инсталирате нужните пакети (ако даде грешка за version mismatch на някои пакети, напишете **npm i –force**)
8. След като пакетите са инсталирани, стартирайте сървъра и напишете **npm run dev** за да стартирате клиентската част.
9. A screen shot of a computer program

   AI-generated content may be incorrect.A screenshot of a computer

   AI-generated content may be incorrect.Накрая в конзолата в която сте писали команди трябва да се появи локалния адрес на който да влезете

**Заключение**

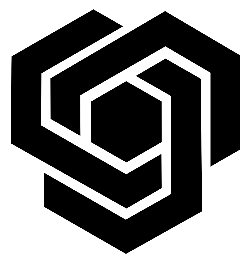
В заключение, разработката на системата Finedge демонстрира завършен и практичен подход към изграждането на съвременна уеб платформа за управление на лични финанси. Интеграцията на AI асистент, използващ OpenAI, добавя интелигентна стойност към приложението чрез персонализирани финансови препоръки, базирани на реални данни и потребителско поведение.

Проектът е създаден с внимание към добрите практики, скалируемост и модулност, което го прави отлична основа за бъдещо развитие. Сред планираните надграждания са:

* създаването на мобилно приложение чрез React Native, което ще разшири достъпността и ще подобри потребителското изживяване на различни устройства.
* автоматично преобразуване на валутите (напр. от лева в евро).
* имплементиране на утвърдени стандарти за уеб достъпност, включително:
  + **Web Content Accessibility Guidelines (WCAG) 2.2**  - насоки за създаване на съдържание, достъпно за хора със зрителни, слухови, когнитивни и двигателни затруднения. [[20]](#Ръководство_за_използване_WCAG)
  + **Section 508** - изисквания за достъпност на цифрови продукти и услуги в публичния сектор, особено в САЩ. [[21]](#Ръководство_за_използване_Section508)
  + **Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA)** - набор от атрибути, които подобряват достъпността на динамични уеб интерфейси за потребители на асистивни технологии. [[22]](#Ръководство_за_използване_WAI_ARIA)
* пускане на продукта на пазара като реална услуга за крайни потребители.

С реализирането на проекта е постигната цялостна интеграция между съвременни технологии и практическо приложение в сферата на личните финанси. Платформата предоставя на потребителите възможност за ефективно следене и управление на разходите, получаване на персонализирани препоръки въз основа на реални финансови данни, както и улеснен достъп до полезна аналитика и визуализации.

Благодарение на модулния подход и спазването на добри практики в разработката, системата е подготвена за лесно разширяване и адаптация към нови нужди и функционалности. По този начин се създава устойчива основа за бъдещо развитие, включително чрез внедряване на мобилно приложение, автоматична конверсия на валути и предлагане на услугата на широк кръг от крайни потребители.

**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

Факултет Приложна математика и информатика

**ДЕКЛАРАЦИЯ**

**ЗА АВТОРСТВО НА ДИПЛОМНА РАБОТА**

Долуподписаният: Васил Паскалев Ганев

специалност : Информатика и софтуерни науки фак. № 471221010

**ДЕКЛАРИРАМ:**

Представената от мен дипломна работа на тема: **Приложение за управление на лични финанси интегриращо базиран на изкуствен интелект асистент** е лична моя авторска разработка, резултат от собствени изследвания.

Потвърждавам, че тя в нейната цялост и отделни части не е била използвана за придобиване на образователна и/или научна степен в ТУ - София или в други университети.

Формулировки, идеи и текстове, взети от други източници, са цитирани точно и с коректно посочване на техните автори. Дипломната работа не е публикувана на друго място.

Декларирам, че предоставям правото на Факултет Приложна

математика и информатика при ТУ - София, съгласно процедурите и правилниците на университета, да архивира и съхранява тази дипломна работа с цел доказване във времето на моето авторство.

Дата: 20.06.2025г. Дипломант:

София

**ИЗПОЛЗВАНА ЛИТЕРАТУРА**

1. [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) - [1]
2. [.NET Core Web API - - [2]](https://dotnet.microsoft.com/en-us/apps/aspnet/apis)
3. [Entity Framework Core](https://learn.microsoft.com/en-us/ef/core/) - [3]

1. [JWT (JSON Web Token)](https://en.wikipedia.org/wiki/JSON_Web_Token) - [4]

1. [Swashbuckle](https://swagger.io/tools/swagger-ui/) - [5]

1. [Hangfire](https://docs.hangfire.io/en/latest/) - [6]

1. [OpenAI Developer API](https://platform.openai.com/docs/overview) - [7]

1. [Vite](https://vite.dev/guide/) - [8]

1. [TypeScript](https://www.typescriptlang.org/) - [9]
2. [React](https://react.dev/) - [10]
3. [React Router DOM](https://reactrouter.com/home) - [11]
4. [TanStack Query (React Query)](https://tanstack.com/query/latest/docs/framework/react/overview) - [12]
5. [Axios](https://axios-http.com/docs/intro) - [13]
6. [KendoReact](https://www.telerik.com/kendo-react-ui/components/introduction) - [14]
7. [Tailwind](https://v2.tailwindcss.com/docs) - [15]
8. [Visual Studio](https://visualstudio.microsoft.com/) – [16]
9. [Visual Studio Code](https://code.visualstudio.com/) – [17]
10. [Node.js](https://nodejs.org/en) – [18]
11. [SQL Server Management Studio (SSMS)](https://learn.microsoft.com/en-us/ssms/sql-server-management-studio-ssms) - [19]

1. [Web Content Accessibility Guidelines (WCAG) 2.2](https://www.w3.org/TR/WCAG22/) – [20]

1. [Section 508](https://www.section508.gov/) – [21]

1. **[WAI-ARIA](https://www.w3.org/WAI/ARIA/apg/) –** [22]

**Приложение - Изходен код**

Пълния изходен код е наличен като приложение с отворен код на <https://github.com/P1l3T0/FinEdge>