Politechnika
Wrocławska

Sprawozdanie końcowe
Finance Tracker

Platformy Programistyczne

Aleksander Żołnowski - 272536
Kamil Fiącek - 272612

17 kwietnia 2025

1 Wstęp

Finance Tracker to aplikacja webowa napisana w .NET wspierająca użytkowników w zarządzaniu finansami osobistymi. Umożliwia planowanie wydatków, ustalanie celów oszczędnościowych, zarządzanie budżetami oraz śledzenie dochodów i transakcji. Projekt został stworzony w ramach zajęć na kierunku Informatyczne Systemy Automatyki z przedmiotu Platformy Programistyczne na specjalności Zastosowania Technologii Informacyjnych (IZT).

2 Funkcjonalności

2.1 Główne funkcjonalności aplikacji

W ramach projektu zaimplementowano następujące funkcjonalności:

- System użytkowników wykorzystując ASP.NET Identity
- Uwierzytelnianie i autoryzacja z wykorzystaniem tokenów JWT
- Tworzenie i zarządzanie planerami wydatków
- Tworzenie i zarządzanie celami oszczędnościowymi
- Tworzenie i zarządzanie dochodami
- Automatyczne generowanie budżetów miesięcznych dla użytkowników z wykorzystaniem HangFire
- Obsługa transakcji (z przeliczaniem walut przez API NBP)
- Historie budżetów i transakcji
- Interfejs użytkownika przez Blazor WASM z wykorzystaniem Mud Blazor'a

2.2 Opisy funkcjonalności użytkownika

Opisy funkcjonalności wykorzystywanych przez użytkowników aplikacji. Dla poniższych funkcjonalności został zaimplementowany pełen zestaw operacji CRUD w celu ustandaryzowania obsługi danych oraz zapewnienia pełnej elastyczności korzystania z systemu.

1. **Plannery wydatków** - Użytkownicy mogą tworzyć szczegółowe plannery wydatków, definiując walutę, kategorie wydatków oraz planowane koszty. Do każdego plannera można wpłacać środki za pomocą transakcji. Planner przechowuje historię transakcji, co pozwala śledzić, jak użytkownik wydaje pieniądze w czasie.
2. **Cele oszczędnościowe** – Użytkownik może ustawić konkretny cel finansowy, określając kwotę docelową i kategorię. Pieniądze można wpłacać na cel poprzez transakcje. Funkcjonalność umożliwia monitorowanie postępów w oszczędzaniu i pomaga skupić się na realizacji wyznaczonych celów.
3. **Dochody** – Użytkownicy mogą dodawać i zarządzać swoimi źródłami dochodu. Na ich podstawie aplikacja automatycznie wylicza miesięczny budżet, pomagając użytkownikowi ocenić jego możliwości finansowe i lepiej planować wydatki.
4. **Budżety** – Budżety tworzone są automatycznie pierwszego dnia każdego miesiąca. Użytkownik może z budżetu przydzielać środki do plannerów wydatków oraz celów oszczędnościowych. Historia budżetów jest zapisywana, dzięki czemu można analizować finanse z poprzednich miesięcy.
5. **Transakcje** – Transakcje służą do przesyłania środków między budżetem użytkownika a jego plannerami lub celami oszczędnościowymi. Aplikacja integruje się z API NBP, aby automatycznie przeliczać waluty, zapewniając zgodność operacji finansowych z walutą danego plannera lub celu.

3 Architektura systemu

Aplikacja została zaprojektowana zgodnie z zasadami **Clean Architecture**, z wyraźnym podziałem na odpowiedzialności między warstwami. Taki układ zapewnia lepszą skalowalność, testowalność oraz separację logiki biznesowej od detali implementacyjnych.
Struktura projektu została podzielona na następujące warstwy:

- `API` – warstwa odpowiedzialna za obsługę żądań HTTP i udostępnianie *endpointów*. Znajduje się tutaj konfiguracja routingu oraz integracja z warstwą aplikacji oraz dodatkowe konfiguracje.
- `Application` – zawiera logikę biznesową aplikacji, komendy, zapytania, walidacje, seedery oraz implementacje interfejsow wykorzystywanych do komunikacji z infrastrukturą. Stanowi trzon funkcjonalności systemu.
- `Client` – projekt frontendowy stworzony w technologii Blazor WebAssembly z wykorzystaniem Mud Blazor. Odpowiada za interfejs użytkownika i komunikację z backendem za pomocą HTTP.
- `Domain` – warstwa z modelami, obiektami, interfejsami oraz podstawowymi definicjami encji. Zawiera również DTO wykorzystywane do transferu danych.
- `Infrastructure` – warstwa z infrastruktura bazy danych (Entity Framework Core, DbContext, Migracje, ASP.NET Identity)
- `NbpRates` – integracja zewnętrznym API Narodowego Banku Polskiego (NBP). Odpowiada za pobieranie i przeliczanie kursów walutowych używanych przy transakcjach.

4 Wykorzystane narzędzia i biblioteki

Cała aplikacja został stworzona z wykorzystaniem języka programowania C#.

4.1 Backend

- .NET 9
- ASP.NET Minimal API
- Entity Framework Core

4.2 Baza danych

W ramach projektu została użyta baza danych **Microsoft SQL Server**.

4.3 Frontend

- .NET 7
- Blazor WebAssembly (WASM)
- Mud Blazor

4.4 Dodatkowe biblioteki, narzędzia i frameworki

- **FluentValidation** - walidacja danych wejściowych
- **FluentResult** - do obsługi zapytań i wyjątków
- **HangFire** - do obsługi background serwisów i zadań wykonywanych w tle (zadania cykliczne)
- **JwtBearer** - Tokeny JWT do uwierzytelniania i autoryzacji użytkownika
- **Seedery** - Seedery do testowania bazy danych z wykorzystaniem biblioteki `Bogus`
- **OpenAPI** - Dokumentacja swagger backendu
- **ASP.NET Identity** - Do zarządzania użytkownikami
- **Refit** - Komunikacja z API NBP

- **Polly** - Retry policy
- **Cache** - w celu minimalizacji zapytan do API i poprawienia wydajności
- **Blazored.LocalStorage** - Przechowywanie tokenu JWT w pamięci przeglądarki po zalogowaniu użytkownika

5 Podsumowanie

Projekt Finance Tracker pozwolił nam w praktyce zastosować wiele nowoczesnych technologii wykorzystywanych we współczesnym tworzeniu aplikacji webowych. Dzięki zastosowaniu wzorca Clean Architecture, aplikacja jest przejrzysta, łatwa do rozbudowy i testowania.

W trakcie realizacji projektu zdobyliśmy praktyczne doświadczenie w zakresie:

- projektowania i implementacji systemu opartego o architekturę warstwową,
- integracji z zewnętrznymi usługami (API NBP),
- pracy z systemem uwierzytelniania i autoryzacji użytkowników (ASP.NET Identity + JWT),
- obsługi zadań cyklicznych w tle (HangFire),
- budowania aplikacji SPA przy użyciu Blazor WASM.

Jednym z głównych wyzwań było pogodzenie automatycznego generowania budżetów z dynamiczną strukturą danych użytkownika oraz zapewnienie prawidłowej konwersji walut w różnych kontekstach. Projekt stanowi dobrą podstawę do dalszego rozwoju – np. dodania integracji z kontami bankowymi, rozbudowania systemu raportów lub aplikacji mobilnej.

6 Bibliografia

1. Dokumentacja ASP.NET: `https://learn.microsoft.com/en-us/aspnet/core`
2. Dokumentacja MudBlazor: `https://mudblazor.com/docs/overview`
3. Dokumentacja FluentValidation: `https://docs.fluentvalidation.net/en/latest/`
4. Dokumentacja Hangfire: `https://www.hangfire.io/`
5. API NBP: `http://api.nbp.pl`