

Uniwersytet Rzeszowski

Wydział Nauk Ścisłych i Technicznych



Jakub Płocica 134960  
Radosław Misiołek 134950  
Kacper Ręczak 134968

# CashFlow – System Zarządzania Budżetem Osobistym i Analizy Wydatków

**Dokumentacja techniczna - Backend**

Projekt zaliczeniowy przedmiotu Bazy Danych

**Prowadzący:**  
mgr inż. Aleksander Wojtowicz

Rzeszów, 2026 r.

# Spis treści

<b>1</b>	<b>Wstęp i Kontekst Biznesowy</b>	<b>4</b>
1.1	Abstrakt . . . . .	4
1.2	Uzasadnienie podjęcia tematu . . . . .	4
1.3	Cele projektu . . . . .	4
<b>2</b>	<b>Analiza Wymagań Systemowych</b>	<b>6</b>
2.1	Wymagania funkcjonalne . . . . .	6
2.1.1	Moduł Użytkownika i Bezpieczeństwa . . . . .	6
2.1.2	Moduł Finansowy . . . . .	6
2.1.3	Moduł Automatyzacji i Analityki . . . . .	7
2.2	Wymagania niefunkcjonalne . . . . .	7
<b>3</b>	<b>Środowisko Programistyczne i Technologie</b>	<b>8</b>
3.1	Platforma .NET 9 (Backend) . . . . .	8
3.2	Baza danych PostgreSQL . . . . .	8
3.3	Kluczowe biblioteki (NuGet Packages) . . . . .	9
<b>4</b>	<b>Architektura Aplikacji i Wzorce Projektowe</b>	<b>10</b>
4.1	Struktura warstwowa . . . . .	10
4.1.1	Warstwa Domeny (CashFlow.Domain) . . . . .	10
4.1.2	Warstwa Aplikacji (CashFlow.Application) . . . . .	10
4.1.3	Warstwa Infrastruktury (CashFlow.Infrastructure) . . . . .	10
4.1.4	Warstwa API (CashFlow.Api) . . . . .	11
4.2	Zastosowane wzorce projektowe . . . . .	11
4.2.1	Repository Pattern . . . . .	11
4.2.2	Unit of Work . . . . .	11
4.2.3	Dependency Injection (DI) . . . . .	11
<b>5</b>	<b>Mechanizmy Bezpieczeństwa</b>	<b>12</b>
5.1	Uwierzytelnianie i Autoryzacja (JWT) . . . . .	12
5.2	Ochrona haseł . . . . .	12
5.3	Walidacja danych wejściowych . . . . .	12

<b>6</b>	<b>Specyfikacja Bazy Danych i Diagram ERD</b>	<b>13</b>
6.1	Diagram Związków Encji (ERD)	13
6.2	Szczegółowa specyfikacja tabel	14
6.2.1	Tabela <code>users</code>	14
6.2.2	Tabela <code>accounts</code>	14
6.2.3	Tabela <code>transactions</code>	14
6.2.4	Tabela <code>rec_transactions</code>	15
6.2.5	Tabela <code>limits</code>	15
6.2.6	Tabela <code>key_words</code>	15
6.2.7	Tabela <code>currencies</code>	15
<b>7</b>	<b>Implementacja Kluczowych Komponentów</b>	<b>16</b>
7.1	Wzorzec Unit of Work	16
7.2	Złożona Logika Transakcyjna ( <code>TransactionService</code> )	17
7.3	Automatyzacja Cykliczna ( <code>RecTransactionService</code> )	19
7.4	Serwis Użytkownika i Seeding Danych ( <code>UserService</code> )	21
7.5	Integracja z Zewnętrznym API ( <code>CurrencyFetcher</code> )	23
7.6	Zaawansowane Mechanizmy i Algorytmika	23
7.6.1	Agregacja Danych i Normalizacja Walutowa	23
7.6.2	Generowanie Tokenów Bezpieczeństwa (JWT)	24
7.6.3	Algorytm Dopasowania Słów Kluczowych (Repozytorium)	25
7.7	Procedury Bezpieczeństwa i Zarządzanie Kontem	25
7.7.1	Zmiana Hasła ( <code>ModifyPassword</code> )	25
7.7.2	Procedura Resetowania Hasła	26
7.7.3	Zmiana Adresu E-mail	27
<b>8</b>	<b>Warstwa Infrastruktury i Walidacji</b>	<b>28</b>
8.1	Konfiguracja Relacji Obiektowo-Relacyjnych (ORM)	28
8.2	Walidacja Danych Wejściowych	30
8.3	Komunikacja SMTP (E-mail)	30
<b>9</b>	<b>Konfiguracja i Harmonogramowanie</b>	<b>32</b>
9.1	Konfiguracja Hangfire	32
9.2	Inicjalizacja Aplikacji ( <code>Program.cs</code> )	32
<b>10</b>	<b>Interfejs Programistyczny (API)</b>	<b>34</b>
10.1	Implementacja Kontrolerów	34
10.1.1	Kontroler Transakcji ( <code>TransactionsController</code> )	34
10.1.2	Kontroler Użytkowników ( <code>UsersController</code> )	35
10.2	Kontrakty Danych (DTO)	36

10.2.1 Modele Żądań (Requests) . . . . .	36
10.2.2 Modele Odpowiedzi (Responses) . . . . .	36
<b>11 Interfejs Programistyczny (API)</b>	<b>37</b>
11.1 Moduł Użytkownika (UsersController) . . . . .	37
11.2 Moduł Rachunków (AccountsController) . . . . .	38
11.3 Moduł Transakcji (TransactionsController) . . . . .	38
11.4 Moduł Transakcji Cyklicznych (RecTransactionsController) . . . . .	38
11.5 Moduł Kategorii (CategoryController) . . . . .	39
11.6 Moduł Słów Kluczowych (KeywordController) . . . . .	39
11.7 Moduł Limitów i Budżetowania (LimitController) . . . . .	39
11.8 Moduł Powiadomień (NotificationController) . . . . .	40
11.9 Moduł Walutowy (CurrencyController) . . . . .	40
<b>12 Instrukcja Instalacji i Konfiguracji Środowiska</b>	<b>41</b>
12.1 Wymagania wstępne . . . . .	41
12.2 Konfiguracja bazy danych . . . . .	41
12.3 Inicjalizacja schematu (Migracje) . . . . .	42
12.4 Uruchomienie serwera . . . . .	42
<b>13 Wnioski</b>	<b>43</b>
<b>A Słownik pojęć i skrótów</b>	<b>44</b>

# Rozdział 1

## Wstęp i Kontekst Biznesowy

### 1.1 Abstrakt

Niniejszy dokument stanowi techniczną specyfikację implementacyjną warstwy serwerowej systemu "CashFlow". System ten jest zaawansowaną platformą klasy FinTech, dedykowaną do zarządzania budżetem domowym, analizy przepływów pieniężnych oraz automatyzacji zobowiązań finansowych. Dokument opisuje architekturę rozwiązania, zastosowane wzorce projektowe, model danych oraz szczegóły implementacyjne kluczowych algorytmów.

### 1.2 Uzasadnienie podjęcia tematu

W dobie rosnącej inflacji i złożoności instrumentów finansowych, brak precyzyjnej kontroli nad wydatkami jest jednym z głównych problemów ekonomicznych gospodarstw domowych. Istniejące na rynku rozwiązania często oferują jedynie podstawową ewidencję wydatków, pomijając aspekty automatyzacji czy zaawansowanej analityki wielowalutowej. Projekt "CashFlow" adresuje te luki, dostarczając narzędzie integrujące funkcje ewidencji, budżetowania i prognozowania w jednym, spójnym ekosystemie.

### 1.3 Cele projektu

Głównym celem prac inżynierskich było stworzenie wysoce wydajnego, skalowalnego i bezpiecznego backendu, który spełnia następujące założenia:

- **Wydajność:** Czas odpowiedzi API poniżej 100ms dla standardowych operacji CRUD.
- **Bezpieczeństwo:** Pełna ochrona danych osobowych i finansowych zgodnie z wytycznymi OWASP (hashowanie haseł, zabezpieczenie przed SQL Injection).

- **Automatyzacja:** Eliminacja konieczności ręcznego wprowadzania powtarzalnych danych dzięki zastosowaniu zadań w tle (Background Jobs).
- **Interoperacyjność:** Możliwość łatwej integracji z różnymi klientami front-endowymi (Web, Mobile) poprzez standard REST.

# Rozdział 2

## Analiza Wymagań Systemowych

### 2.1 Wymagania funkcjonalne

Na etapie analizy zidentyfikowano następujące kluczowe funkcjonalności systemu, które zostały zaimplementowane w warstwie backendowej:

#### 2.1.1 Moduł Użytkownika i Bezpieczeństwa

- Rejestracja konta z walidacją unikalności adresu e-mail i nazwy użytkownika.
- Bezpieczne logowanie z wykorzystaniem mechanizmu JWT (JSON Web Token).
- Procedura odzyskiwania hasła z wykorzystaniem jednorazowych tokenów przesyłanych drogą mailową.
- Weryfikacja tożsamości poprzez link aktywacyjny.

#### 2.1.2 Moduł Finansowy

- Zarządzanie wieloma rachunkami (portfelami) w różnych walutach.
- Rejestracja transakcji przychodzących i wychodzących z zachowaniem spójności sald.
- Definiowanie i edycja kategorii wydatków.
- Definiowanie limitów budżetowych dla kategorii w zadanych przedziałach czasowych.

### 2.1.3 Moduł Automatyzacji i Analityki

- Obsługa transakcji cyklicznych (zdefiniowana częstotliwość: dziennie, tygodniowo, miesięcznie, rocznie).
- Automatyczna kategoryzacja transakcji na podstawie słów kluczowych w opisie (Rule Engine).
- Generowanie raportów analitycznych (dziennych, miesięcznych, kategoryzowanych) normalizowanych do waluty bazowej (PLN).
- Synchronizacja kursów walut z API NBP w czasie rzeczywistym (lub cyklicznym).

## 2.2 Wymagania нефunkcjonalne

- **Skalowalność:** Architektura musi umożliwiać łatwą rozbudowę o nowe moduły bez konieczności refaktoryzacji jądra systemu.
- **Dostępność:** System powinien być odporny na błędy w komunikacji z serwisami zewnętrznymi (np. NBP API).
- **Bezpieczeństwo:** Hasła muszą być hashowane algorytmem odpornym na ataki typu rainbow table (zastosowano BCrypt). Komunikacja API musi być szyfrowana.

## Rozdział 3

# Środowisko Programistyczne i Technologie

### 3.1 Platforma .NET 9 (Backend)

Wybór platformy .NET 9 podyktowany był jej dojrzałością, wydajnością oraz bogatym ekosystemem. Wykorzystanie języka C# w wersji 13.0 pozwoliło na użycie nowoczesnych konstrukcji językowych, takich jak słowo kluczowe `required` dla właściwości obiektów DTO, co zwiększa bezpieczeństwo typów na etapie kompilacji.

### 3.2 Baza danych PostgreSQL

Jako warstwę persystencji danych wybrano PostgreSQL. Jest to zaawansowany, obiektowo-relacyjny system bazodanowy (ORDBMS), który oferuje:

- Pełną zgodność ze standardem ACID (Atomicity, Consistency, Isolation, Durability).
- Obsługę zaawansowanych typów danych.
- Wysoką wydajność przy złożonych zapytaniach analitycznych (agregacje, grupowanie).

### 3.3 Kluczowe biblioteki (NuGet Packages)

1. **Entity Framework Core (Npgsql)**: Zastosowano podejście *Code-First*, co pozwoliło na zarządzanie schematem bazy danych z poziomu kodu C# poprzez mechanizm Migracji.
2. **Hangfire**: Biblioteka do przetwarzania zadań w tle. Wykorzystuje ona trwałe magazyn danych (PostgreSQL) do kolejkowania zadań, co gwarantuje ich wykonanie nawet w przypadku restartu aplikacji.
3. **FluentValidation**: Biblioteka umożliwiająca separację reguł walidacji od modeli danych, co jest zgodne z zasadą Single Responsibility Principle (SRP).
4. **MailKit / MimeKit**: Nowoczesna, asynchroniczna biblioteka do obsługi protokołu SMTP, wykorzystywana do wysyłki powiadomień.
5. **BCrypt.Net-Next**: Implementacja funkcji skrótu haseł, zapewniająca odporność na ataki brute-force dzięki konfigurowalnemu parametrowi *Work Factor* i soleniu.

## Rozdział 4

# Architektura Aplikacji i Wzorce Projektowe

System został zaprojektowany zgodnie z paradygmatem **Clean Architecture** (Czysta Architektura), co zapewnia niezależność logiki biznesowej od frameworków, interfejsu użytkownika oraz bazy danych.

### 4.1 Struktura warstwowa

Rozwiązanie zostało podzielone na cztery główne projekty (assemblies):

#### 4.1.1 Warstwa Domeny (CashFlow.Domain)

Stanowi jądro systemu. Zawiera encje (Entities), które reprezentują obiekty biznesowe i stanowią odzwierciedlenie struktury bazy danych. Warstwa ta nie posiada żadnych zależności zewnętrznych.

#### 4.1.2 Warstwa Aplikacji (CashFlow.Application)

Zawiera logikę biznesową systemu. Zdefiniowane są tu interfejsy repozytoriów i serwisów, modele DTO (Data Transfer Objects), walidatory oraz implementacje serwisów (Services). To tutaj zapadają decyzje biznesowe (np. czy użytkownik ma wystarczające środki na koncie).

#### 4.1.3 Warstwa Infrastruktury (CashFlow.Infrastructure)

Odpowiada za szczegóły techniczne, takie jak dostęp do bazy danych, komunikacja z zewnętrznymi API (NBP) czy wysyłka e-maili. Warstwa ta implementuje interfejsy zdefiniowane w warstwie Aplikacji.

#### 4.1.4 Warstwa API (CashFlow.Api)

Punkt wejścia do systemu. Zawiera kontrolery REST, konfigurację kontenera Dependency Injection oraz Middleware.

## 4.2 Zastosowane wzorce projektowe

### 4.2.1 Repository Pattern

Wzorzec ten posłużył do abstrakcji warstwy dostępu do danych. Dzięki niemu, serwisy biznesowe nie operują bezpośrednio na kontekście Entity Framework ('DbContext'), lecz na interfejsach repozytoriów (np. 'IAccountRepository'). Ułatwia to testowanie jednostkowe.

### 4.2.2 Unit of Work

Wzorzec Jednostki Pracy zapewnia spójność transakcyjną. Pozwala na grupowanie wielu operacji na repozytoriach w jedną logiczną transakcję biznesową. Metoda 'SaveChangesAsync' wywoływana jest tylko raz, na końcu procesu biznesowego.

### 4.2.3 Dependency Injection (DI)

Wstrzykiwanie zależności jest realizowane natywnie przez framework ASP.NET Core. Wszystkie serwisy i repozytoria są rejestrowane w kontenerze IoC z odpowiednim cyklem życia (zazwyczaj `Scoped`, co oznacza jedną instancję na żądanie HTTP).

# Rozdział 5

## Mechanizmy Bezpieczeństwa

Bezpieczeństwo danych było priorytetem podczas projektowania systemu. Zastosowano wielowarstwową strategię ochrony.

### 5.1 Uwierzytelnianie i Autoryzacja (JWT)

System wykorzystuje bezstanowy mechanizm uwierzytelniania oparty na tokenach JWT (JSON Web Token).

- Po poprawnym zalogowaniu, serwer generuje podpisany cyfrowo token zawierający roszczenia (Claims), takie jak 'UserId', 'Email' czy 'Role'.
- Token jest przesyłany w nagłówku HTTP 'Authorization: Bearer <token>' przy każdym żądaniu do chronionych zasobów.
- Middleware weryfikuje podpis tokena oraz jego czas ważności (Expiration Time).

### 5.2 Ochrona haseł

Hasła użytkowników nigdy nie są przechowywane w formie otwartego tekstu. Podczas rejestracji hasło jest poddawane procesowi haszowania algorytmem BCrypt z użyciem losowej soli. Uniemożliwia to odtworzenie hasła nawet w przypadku wycieku bazy danych.

### 5.3 Walidacja danych wejściowych

Wszystkie dane przychodzące do API są rygorystycznie walidowane przy użyciu biblioteki FluentValidation. Zapobiega to atakom typu SQL Injection (dzięki EF Core, który parametryzuje zapytania) oraz XSS (Cross-Site Scripting).

## Rozdział 6

# Specyfikacja Bazy Danych i Diagram ERD

Struktura bazy danych została zaprojektowana w sposób znormalizowany, zgodnie z trzecią postacią normalną (3NF). Dzięki podejściu Code-First, definicje tabel powstają bezpośrednio z klas C# w warstwie domeny.

### 6.1 Diagram Związków Encji (ERD)

Poniższy schemat obrazuje relacje między tabelami w bazie danych PostgreSQL.



Rysunek 6.1: Schemat ERD bazy danych CashFlow

## 6.2 Szczegółowa specyfikacja tabel

### 6.2.1 Tabela users

Centralna tabela przechowująca tożsamości użytkowników.

- `user_id` (PK, int): Unikalny identyfikator.
- `email` (varchar, unique): Adres e-mail, login.
- `nickname` (varchar, unique): Nazwa wyświetlana.
- `password_hash` (varchar): Zaszyfrowane hasło.
- `is_active` (bool): Flaga aktywności (Soft Delete).
- `is_verified` (bool): Czy email został potwierdzony.
- `created_at`, `updated_at`: Znaczniki czasowe.

### 6.2.2 Tabela accounts

Przechowuje portfele finansowe użytkownika. Relacja 1:N z tabelą Users.

- `account_id` (PK, int).
- `balance` (decimal(22,2)): Aktualne saldo konta.
- `currency_code` (varchar, FK): Kod waluty (np. PLN).
- `photo_url` (varchar): Link do ikony konta.

### 6.2.3 Tabela transactions

Rejestr operacji finansowych. Relacja 1:N z Accounts, Users, Categories.

- `transaction_id` (PK, int).
- `amount` (decimal): Kwota transakcji.
- `type` (varchar): 'income' lub 'expense'.
- `description` (varchar): Opis operacji.
- `date` (timestamp): Data wykonania.

### 6.2.4 Tabela rec\_transactions

Szablony transakcji cyklicznych. Nie są to transakcje wykonane, lecz definicje dla schedulera.

- `rec_transaction_id` (PK, int).
- `frequency` (varchar): Częstotliwość (Daily, Weekly...).
- `next_payment_date` (timestamp): Data kolejnego uruchomienia.
- `is_true` (bool): Tryb automatyczny (true) lub przypomnienie (false).

### 6.2.5 Tabela limits

Budżety dla kategorii.

- `limit_id` (PK, int).
- `value` (decimal): Kwota limitu.
- `start_date, end_date`: Przedział czasowy.

### 6.2.6 Tabela key\_words

Słownik do autokategoryzacji.

- `word_id` (PK, int).
- `word` (varchar): Fraza kluczowa (np. "Netflix").
- `category_id` (FK): Kategoria docelowa.

### 6.2.7 Tabela currencies

Słownik walut aktualizowany z NBP.

- `currency_code` (PK, varchar).
- `rate_to_base` (decimal): Kurs wymiany.

# Rozdział 7

## Implementacja Kluczowych Komponentów

W niniejszym rozdziale zaprezentowano obszerne fragmenty kodu źródłowego, które stanowią rdzeń logiki biznesowej aplikacji. Kod został zaimplementowany w C# 13 (.NET 9).

### 7.1 Wzorzec Unit of Work

Implementacja ‘UnitOfWork’ zarządza transakcjami bazodanowymi, gwarantując atomowość operacji wieloetapowych.

```
1 using CashFlow.Application.Interfaces;
2 using CashFlow.Infrastructure.Data;
3 using Microsoft.EntityFrameworkCore.Storage;
4 using System.Threading.Tasks;
5
6 namespace CashFlow.Infrastructure.Data
7 {
8     public class UnitOfWork : IUnitOfWork
9     {
10         private readonly CashFlowDbContext _context;
11
12         public UnitOfWork(CashFlowDbContext context)
13         {
14             _context = context;
15         }
16
17         public async Task<IDbContextTransaction> BeginTransactionAsync()
18         {
19             return await _context.Database.BeginTransactionAsync();
20         }
21
22         public async Task<int> SaveChangesAsync()
23         {
24             return await _context.SaveChangesAsync();
25         }
26
27         public void Dispose()
28         {
29             _context.Dispose();
30         }
31     }
32 }
```

Listing 7.1: CashFlow.Infrastructure.Data.UnitOfWork.cs

## 7.2 Złożona Logika Transakcyjna (TransactionService)

Serwis ten realizuje najważniejszy proces biznesowy: dodawanie transakcji. Metoda ‘CreateNewTransactionAsync’ zawiera logikę autokategoryzacji (sprawdzanie słów kluczowych), walidację salda oraz system ostrzegania o przekroczeniu limitów budżetowych.

```
1 using CashFlow.Application.Interfaces;
2 using CashFlow.Application.Repositories;
3 using CashFlow.Domain.Models;
4 using CashFlow.Application.DTO.Requests;
5 using CashFlow.Application.DTO.Responses;
6
7 namespace CashFlow.Application.Services
8 {
9     public class TransactionService : ITransactionService
10    {
11        private readonly ITransactionRepository _transactionRepository;
12        private readonly IAccountRepository _accountRepository;
13        private readonly IKeywordRepository _keywordRepository;
14        private readonly IUnitOfWork _unitOfWork;
15        private readonly ILimitRepository _limitRepository;
16        private readonly INotificationService _notificationService;
17        private readonly ICategoryRepository _categoryRepository;
18        private readonly ICurrencyService _currencyService;
19
20        public TransactionService(ITransactionRepository transactionRepository,
21            IAccountRepository accountRepository, IKeywordRepository keywordRepository,
22            IUnitOfWork unitOfWork, ILimitRepository limitRepository, INotificationService
23            notificationService, ICategoryRepository categoryRepository, ICurrencyService
24            currencyService)
25        {
26            _transactionRepository = transactionRepository;
27            _accountRepository = accountRepository;
28            _keywordRepository = keywordRepository;
29            _unitOfWork = unitOfWork;
30            _limitRepository = limitRepository;
31            _notificationService = notificationService;
32            _categoryRepository = categoryRepository;
33            _currencyService = currencyService;
34        }
35
36        public async Task CreateNewTransactionAsync(int userId, NewTransactionRequest
37            request, bool useTransaction = true)
38        {
39            using var dbTransaction = useTransaction ? await _unitOfWork.
40                BeginTransactionAsync() : null;
41            try
42            {
43                if (request.CategoryId == null)
44                {
45                    if (request.Description == null)
46                    {
47                        request.Description = string.Empty;
48                    }
49                    request.CategoryId = await _keywordRepository.
50                        GetCategoryIdByDescriptionAsync(userId, request.Description);
51                }
52
53                if (request.Amount <= 0 || request.Amount == null)
54                {
55                    throw new Exception("Transaction must be greater than 0");
56                }
57                if (request.AccountId == null)
58                {
59                    throw new Exception("AccountId is required to create a transaction");
60                }
61                if (request.CategoryId == null)
62                {
63                    throw new Exception("CategoryId is required to create a transaction")
64                }
65            }
66
67            ;
68
69            var account = await _accountRepository.GetAccountByIdAsync(userId, (int)
70                request.AccountId!);
71
72            if (account == null)
73            {
74                throw new Exception("Account not found or access denied.");
75            }
76        }
77    }
78 }
```

```

65     var transactionType = request.Type!.ToLower();
66
67     if (transactionType == "expense")
68     {
69         if (account.Balance < request.Amount)
70         {
71             throw new Exception("You can not have less than zero money");
72         }
73         account.Balance -= (decimal)request.Amount!;
74     }
75     if (transactionType == "income")
76     {
77         account.Balance += (decimal)request.Amount!;
78     }
79
80     var newTransaction = new Transaction
81     {
82         UserId = userId!,
83         AccountId = (int)request.AccountId!,
84         CategoryId = request.CategoryId.Value,
85         Amount = (decimal)request.Amount!,
86         Description = request.Description!,
87         Type = request.Type!
88     };
89
90     await _transactionRepository.AddAsync(newTransaction);
91     await _accountRepository.UpdateAsync(account);
92
93     if (transactionType == "expense")
94     {
95         var categoryId = newTransaction.CategoryId;
96         var limits = await _limitRepository.GetLimitsForCategoryAsync(
categoryId);
97
98         foreach (var limit in limits)
99         {
100             if (DateTime.UtcNow >= limit.StartDate && DateTime.UtcNow <=
limit.EndDate)
101             {
102                 var category = await _categoryRepository.
GetCategoryInfoByIdWithDetailsAsync(userId, limit.CategoryId);
103
104                 if (category == null)
105                 {
106                     continue;
107                 }
108
109                 var spentAmount = await _transactionRepository.
GetCategorySpendingAsync(userId, limit.CategoryId, limit.AccountId, limit.StartDate,
limit.EndDate);
110
111                 if (spentAmount > limit.Value)
112                 {
113                     await _notificationService.SendNotificationAsync(userId,
$"Spending limit exceeded for limit: {limit.Name}", $"You reached your limit for
category: {category.Name!} on account: {account.Name!} You reached {spentAmount}/{
limit.Value}", "info");
114                 }
115                 else if (spentAmount > (limit.Value * 0.8m))
116                 {
117                     await _notificationService.SendNotificationAsync(userId,
$"You are close to exceeding the limit for the limit: {limit.Name}", $"You nearly
reached your limit for category: {category.Name!} on account: {account.Name!} You
reached {spentAmount}/{limit.Value}", "info");
118                 }
119             }
120         }
121     }
122     if (dbTransaction != null)
123     {
124         await dbTransaction.CommitAsync();
125     }
126 }
127 catch
128 {
129     if (dbTransaction != null)
130     {
131         await dbTransaction.RollbackAsync()!;
132     }
133     throw;
134 }
135 }
136 }
137

```

Listing 7.2: CashFlow.Application.Services.TransactionService.cs

## 7.3 Automatyzacja Cykliczna (RecTransactionService)

Serwis odpowiedzialny za logikę "serca" systemu automatyzacji. Współpracuje z Hangfire, wykonując transakcje w wyznaczonych terminach.

```

1 using CashFlow.Application.Interfaces;
2 using CashFlow.Application.Repositories;
3 using CashFlow.Domain.Models;
4 using CashFlow.Application.DTO.Requests;
5 using CashFlow.Application.DTO.Responses;
6
7 namespace CashFlow.Application.Services
8 {
9     public class RecTransactionService : IRecTransactionService
10    {
11        private readonly IRecTransactionRepository _recTransactionRepository;
12        private readonly ITransactionRepository _transactionRepository;
13        private readonly IAccountRepository _accountRepository;
14        private readonly IEmailService _emailService;
15        private readonly IUnitOfWork _unitOfWork;
16        private readonly IKeywordRepository _keywordRepository;
17        private readonly ITransactionService _transactionService;
18        private readonly INotificationService _notificationService;
19
20        public RecTransactionService(IRecTransactionRepository recTransactionRepository,
21        ITransactionRepository transactionRepository, IAccountRepository accountRepository,
22        IEmailService emailService, IUnitOfWork unitOfWork, IKeywordRepository
23        keywordRepository, ITransactionService transactionService, INotificationService
24        notificationService)
25        {
26            _recTransactionRepository = recTransactionRepository;
27            _transactionRepository = transactionRepository;
28            _accountRepository = accountRepository;
29            _emailService = emailService;
30            _unitOfWork = unitOfWork;
31            _keywordRepository = keywordRepository;
32            _transactionService = transactionService;
33            _notificationService = notificationService;
34        }
35
36        private DateTime CalculateNextDate(DateTime nextPaymentDate, string frequency)
37        {
38            var cleanedFrequency = frequency.ToLower().Trim();
39
40            switch (cleanedFrequency)
41            {
42                case "daily":
43                    nextPaymentDate = nextPaymentDate.AddDays(1);
44                    break;
45                case "weekly":
46                    nextPaymentDate = nextPaymentDate.AddDays(7);
47                    break;
48                case "monthly":
49                    nextPaymentDate = nextPaymentDate.AddMonths(1);
50                    break;
51                case "yearly":
52                    nextPaymentDate = nextPaymentDate.AddYears(1);
53                    break;
54                default:
55                    break;
56            }
57            return nextPaymentDate;
58        }
59
60        public async Task ProcessPendingTransactionsAsync()
61        {
62            var recTransactionsPending = await _recTransactionRepository.
63            GetPendingRecurringTransactionsWithDetailsAsync(DateTime.UtcNow);
64
65            var userSummaries = new Dictionary<int, List<string>>();
66
67            foreach (var recTransactionPending in recTransactionsPending)
68            {

```

```

64         using var dbTransaction = await _unitOfWork.BeginTransactionAsync();
65         try
66         {
67             if (recTransactionPending.User == null || recTransactionPending.
Account == null) continue;
68
69             if (!userSummaries.ContainsKey(recTransactionPending.UserId))
70             {
71                 userSummaries[recTransactionPending.UserId] = new List<string>();
72             }
73
74             if (recTransactionPending.IsTrue == true)
75             {
76                 var transactionRequest = new NewTransactionRequest()
77                 {
78                     AccountId = recTransactionPending.AccountId,
79                     CategoryId = recTransactionPending.CategoryId,
80                     Amount = recTransactionPending.Amount,
81                     Description = "(Automat) " + recTransactionPending.
Description,
82                     Type = recTransactionPending.Type
83                 };
84
85                 await _transactionService.CreateNewTransactionAsync(
recTransactionPending.UserId, transactionRequest, false);
86
87                 userSummaries[recTransactionPending.UserId].Add($"Added: {
recTransactionPending.Name} ({recTransactionPending.Amount})");
88             }
89             else if (recTransactionPending.IsTrue == false)
90             {
91                 userSummaries[recTransactionPending.UserId].Add($"To add: {
recTransactionPending.Name} ({recTransactionPending.Amount})");
92             }
93
94             recTransactionPending.NextPaymentDate = CalculateNextDate(
recTransactionPending.NextPaymentDate, recTransactionPending.Frequency);
95             recTransactionPending.UpdatedAt = DateTime.UtcNow;
96
97             await _recTransactionRepository.UpdateAsync(recTransactionPending);
98             await dbTransaction.CommitAsync();
99         }
100         catch (Exception ex)
101         {
102             await dbTransaction.RollbackAsync();
103             Console.WriteLine(ex);
104         }
105     }
106
107     foreach (var summary in userSummaries)
108     {
109         var userId = summary.Key;
110         var messages = summary.Value;
111
112         string title = $"Today recurring transaction report for: ({messages.Count
}) transaction/s";
113         string body = string.Join("\n", messages);
114
115         await _notificationService.SendNotificationAsync(userId, title, $"There
is list for recurring transactions on your account" + "\n" + body, "info");
116     }
117
118     await ProcessUpcomingRemindersAsync();
119 }
120
121 public async Task ProcessUpcomingRemindersAsync()
122 {
123     var recTransactionsUpcoming = await _recTransactionRepository.
GetUpcomingRecurringTransactionsWithDetailsAsync(DateTime.UtcNow.AddDays(1));
124
125     var userSummaries = new Dictionary<int, List<string>>();
126
127     foreach (var recTransactionUpcoming in recTransactionsUpcoming)
128     {
129         if (recTransactionUpcoming.User == null || recTransactionUpcoming.Account
== null) continue;
130
131         if (!userSummaries.ContainsKey(recTransactionUpcoming.UserId))
132         {
133             userSummaries[recTransactionUpcoming.UserId] = new List<string>();
134         }
135
136         userSummaries[recTransactionUpcoming.UserId].Add($"Tomorrow: {
recTransactionUpcoming.Name} ({recTransactionUpcoming.Amount})");

```

```

137     }
138
139     foreach (var summary in userSummaries)
140     {
141         var userId = summary.Key;
142         var messages = summary.Value;
143
144         string title = $"Upcoming recurring transaction report for: ({messages.
Count}) transaction/s";
145         string body = string.Join("\n", messages);
146
147         await _notificationService.SendNotificationAsync(userId, title, "There is
list for upcoming recurring transactions on your account" + "\n" + body, "info");
148     }
149 }
150 }
151 }

```

Listing 7.3: CashFlow.Application.Services.RecTransactionService.cs

## 7.4 Serwis Użytkownika i Seeding Danych (UserService)

Obsługuje rejestrację i inicjalizację konta (seeding kategorii i słów kluczowych).

```

1 using CashFlow.Application.Interfaces;
2 using CashFlow.Application.Repositories;
3 using CashFlow.Domain.Models;
4 using CashFlow.Application.DTO.Requests;
5 using CashFlow.Application.DTO.Responses;
6 using BCrypt.Net;
7 using Microsoft.Extensions.Configuration;
8
9 namespace CashFlow.Application.Services
10 {
11     public class UserService : IUserService
12     {
13         private readonly IUserRepository _userRepository;
14         private readonly IJWTService _jwtService;
15         private readonly IEmailService _emailService;
16         private readonly ICategoryRepository _categoryRepository;
17         private readonly IKeywordRepository _keywordRepository;
18         private readonly IAccountRepository _accountRepository;
19         private readonly IUnitOfWork _unitOfWork;
20         private readonly IConfiguration _configuration;
21
22         public UserService(IUserRepository userRepository, IJWTService jwtService,
IEmailService emailService, ICategoryRepository categoryRepository,
IKeywordRepository keywordRepository, IAccountRepository accountRepository,
IUnitOfWork unitOfWork, IConfiguration configuration)
23         {
24             _userRepository = userRepository;
25             _jwtService = jwtService;
26             _emailService = emailService;
27             _categoryRepository = categoryRepository;
28             _keywordRepository = keywordRepository;
29             _accountRepository = accountRepository;
30             _unitOfWork = unitOfWork;
31             _configuration = configuration;
32         }
33
34         public async Task RegisterAsync(RegisterRequest request)
35         {
36             var isEmailTaken = await _userRepository.IsEmailTakenAsync(request.Email!);
37             var isNicknameTaken = await _userRepository.IsNicknameTakenAsync(request.
Nickname!);
38
39             if (isEmailTaken == true || isNicknameTaken == true)
40             {
41                 throw new Exception($"Given email or nickname is taken!");
42             }
43
44             string passwordHash = BCrypt.Net.BCrypt.HashPassword(request.Password);
45             var userGUID = Guid.NewGuid().ToString();
46
47             var newUser = new User
48             {
49                 FirstName = request.FirstName!,
50                 LastName = request.LastName!,

```

```

51         Email = request.Email!,
52         Nickname = request.Nickname!,
53         PasswordHash = passwordHash,
54         IsAdmin = false,
55         IsActive = true,
56         IsVerified = false,
57         VerificationToken = userGUID,
58         VerifiedAt = null,
59     };
60
61     await _userRepository.AddAsync(newUser);
62     var backendUrl = _configuration["AppUrls:BackendUrl"];
63     await _emailService.SendEmailAsync(request.Email!, "Welcome to CashFlow!", $"
    <h1>Welcome to CashFlow {request.FirstName} {request.LastName}</h1><br>Please click
    the link below to activate your account!<br><a href=\"{backendUrl}/api/verify?
    verificationToken={userGUID}\">VERIFY HERE</a>");
64     await SeedUserDataAsync(newUser.UserId);
65 }
66
67 public async Task SeedUserDataAsync(int userId)
68 {
69     var defaultCategories = new List<(string Name, string Icon, string Color,
    string Type, string[] Keywords)>
70     {
71         ("Jedzenie", "fastfood", "#FF5733", "expense", new[] { "Biedronka", "Lidl
    ", "KFC", "McDonalds", "Żabka" }),
72         ("Transport", "directions_car", "#33B5FF", "expense", new[] { "Shell", "
    Orlen", "Uber", "Bolt", "ZTM", "PKP" }),
73         ("Wynagrodzenie", "payments", "#28B463", "income", new[] { "Przelew", "
    Salary", "Premia", "Wynagrodzenie" }),
74         ("Rozrywka", "theater_comedy", "#AF7AC5", "expense", new[] { "Netflix", "
    Spotify", "Cinema", "Kino", "Pub" }),
75         ("Zdrowie", "medical_services", "#E74C3C", "expense", new[] { "Apteka", "
    Dentysta", "Lekarz", "Luxmed" })
76     };
77
78     var defaultAccounts = new List<(string Name, decimal Balance, string
    CurrencyCode, bool IsActive, string PhotoUrl)>
79     {
80         ("Główne Konto", 0.00m, "PLN", true, "default_account_url")
81     };
82
83     foreach (var accountData in defaultAccounts)
84     {
85         var newAccount = new Account
86         {
87             UserId = userId,
88             Name = accountData.Name,
89             Balance = accountData.Balance,
90             CurrencyCode = accountData.CurrencyCode,
91             IsActive = accountData.IsActive,
92             PhotoUrl = accountData.PhotoUrl,
93         };
94         await _accountRepository.AddAsync(newAccount);
95     }
96
97     foreach (var item in defaultCategories)
98     {
99         var category = new Category
100         {
101             UserId = userId,
102             Name = item.Name,
103             Icon = item.Icon,
104             Color = item.Color,
105             Type = item.Type
106         };
107         await _categoryRepository.AddAsync(category);
108
109         foreach (var word in item.Keywords)
110         {
111             await _keyWordRepository.AddAsync(new KeyWord
112             {
113                 UserId = userId,
114                 CategoryId = category.CategoryId,
115                 Word = word
116             });
117         }
118     }
119 }
120 }
121 }

```

Listing 7.4: CashFlow.Application.Services.UserService.cs

## 7.5 Integracja z Zewnętrznym API (CurrencyFetcher)

Klient HTTP odpowiedzialny za pobieranie tabel kursowych z NBP.

```
1 using CashFlow.Application.Interfaces;
2 using CashFlow.Application.DTO.Externals;
3 using System.Net.Http.Json;
4
5 namespace CashFlow.Infrastructure.ExternalServices
6 {
7     public class NbpTableDto
8     {
9         [System.Text.Json.Serialization.JsonPropertyName("rates")]
10        public List<CurrencyRate> Rates { get; set; } = new();
11    }
12
13    public class CurrencyFetcher : ICurrencyFetcher
14    {
15        private readonly HttpClient _httpClient;
16
17        public CurrencyFetcher(HttpClient httpClient)
18        {
19            _httpClient = httpClient;
20        }
21
22        public async Task<List<CurrencyRate>> FetchRatesAsync()
23        {
24            var result = await _httpClient.GetFromJsonAsync<List<NbpTableDto>>("http://
25            api.nbp.pl/api/exchangerates/tables/A/?format=json");
26
27            if (result == null || result.Count == 0)
28            {
29                return new List<CurrencyRate>();
30            }
31
32            return result[0].Rates;
33        }
34    }
```

Listing 7.5: CashFlow.Infrastructure.ExternalServices.CurrencyFetcher.cs

## 7.6 Zaawansowane Mechanizmy i Algorytmika

W tej sekcji przedstawiono implementację bardziej złożonych problemów inżynierskich, takich jak normalizacja danych wielowalutowych, kryptograficzne zabezpieczanie sesji oraz algorytmy dopasowania wzorców tekstowych.

### 7.6.1 Agregacja Danych i Normalizacja Walutowa

Metoda ‘GetCategoryAnalyticsAsync’ w serwisie transakcyjnym demonstruje wykorzystanie zaawansowanych zapytań LINQ (Language Integrated Query) do grupowania i sumowania danych. Kluczowym aspektem jest dynamiczna normalizacja kwot transakcji do waluty bazowej (PLN) "w locie", co pozwala na spójną analizę portfela składającego się z wielu walut obcych.

```
1 public async Task<List<CategoryAnalyticsResponse>> GetCategoryAnalyticsAsync(int userId,
2     DateTime startDate, DateTime endDate, string type)
3 {
4     var transactions = await _transactionRepository.GetTransactionsForAnalyticsAsync(
5         userId, startDate, endDate, type);
6
7     if (!transactions.Any()) return new List<CategoryAnalyticsResponse>();
8
9     var uniqueCurrencies = transactions.Select(t => t.Account.CurrencyCode).Distinct();
```

```

8   var rates = new Dictionary<string, decimal>();
9   foreach (var code in uniqueCurrencies)
10  {
11      rates[code] = await _currencyService.GetExchangeRateAsync(code, "PLN");
12  }
13
14  var normalizedTransactions = transactions.Select(t => new
15  {
16      t.Category,
17      AmountInPLN = t.Amount * (rates.ContainsKey(t.Account.CurrencyCode) ? rates[t.
18      Account.CurrencyCode] : 1.0m)
19  }).ToList();
20
21  decimal totalAmount = normalizedTransactions.Sum(t => t.AmountInPLN);
22
23  var analytics = normalizedTransactions
24  .GroupBy(t => t.Category)
25  .Select(g => new CategoryAnalyticsResponse
26  {
27      CategoryId = g.Key.CategoryId,
28      CategoryName = g.Key.Name,
29      Color = g.Key.Color ?? "#808080",
30      TotalValue = g.Sum(t => t.AmountInPLN),
31      Percentage = totalAmount == 0 ? 0 : Math.Round((g.Sum(t => t.AmountInPLN) /
32      totalAmount) * 100, 2)
33  })
34  .OrderByDescending(x => x.TotalValue)
35  .ToList();
36
37  return analytics;
38 }

```

Listing 7.6: Fragment TransactionService.cs - Analitka Wielowalutowa

## 7.6.2 Generowanie Tokenów Bezpieczeństwa (JWT)

Za bezpieczeństwo sesji odpowiada ‘JWTService’. Poniższy kod obrazuje proces tworzenia cyfrowo podpisanego tokena. W tokenie zaszywane są roszczenia (Claims) identyfikujące użytkownika oraz jego rolę, co eliminuje konieczność odpytywania bazy danych przy każdym żądaniu autoryzowanym.

```

1  using CashFlow.Application.Interfaces;
2  using CashFlow.Domain.Models;
3  using Microsoft.Extensions.Configuration;
4  using Microsoft.IdentityModel.Tokens;
5  using System.IdentityModel.Tokens.Jwt;
6  using System.Security.Claims;
7  using System.Text;
8
9  namespace CashFlow.Application.Services
10 {
11     public class JWTService : IJWTService
12     {
13         private readonly IConfiguration _configuration;
14         public JWTService(IConfiguration configuration)
15         {
16             _configuration = configuration;
17         }
18
19         public string GenerateTokenAsync(User user)
20         {
21             var claims = new List<Claim>
22             {
23                 new Claim(ClaimTypes.NameIdentifier, user.UserId.ToString()),
24                 new Claim(ClaimTypes.Email, user.Email!),
25                 new Claim(ClaimTypes.Name, user.Nickname!),
26                 new Claim(ClaimTypes.Role, user.IsAdmin ? "Admin" : "User")
27             };
28
29             var secretKey = _configuration["JwtSettings:Secret"];
30             if (string.IsNullOrEmpty(secretKey))
31             {
32                 throw new InvalidOperationException("JWT Secret key is not configured in
33                 appsettings.json.");
34             }
35         }
36     }
37 }

```

```

35     var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));
36     var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
37
38     var tokenDescriptor = new SecurityTokenDescriptor
39     {
40         Subject = new ClaimsIdentity(claims),
41         Expires = DateTime.UtcNow.AddMinutes(double.Parse(_configuration["
JwtSettings:ExpiryMinutes"]!)),
42         SigningCredentials = credentials,
43         Issuer = _configuration["JwtSettings:Issuer"],
44         Audience = _configuration["JwtSettings:Audience"]
45     };
46
47     var tokenHandler = new JwtSecurityTokenHandler();
48     var token = tokenHandler.CreateToken(tokenDescriptor);
49
50     return tokenHandler.WriteToken(token);
51 }
52 }
53 }

```

Listing 7.7: Implementacja JWTService.cs

### 7.6.3 Algorytm Dopasowania Słów Kluczowych (Repozytorium)

Silnik reguł opiera się na wydajnym przeszukiwaniu słownika słów kluczowych. Metoda ‘GetCategoryIdByDescriptionAsync’ w repozytorium ‘KeyWordRepository’ implementuje logikę, która dopasowuje najdłuższe możliwe słowo kluczowe do opisu transakcji, minimalizując ryzyko błędnej kategoryzacji (np. odróżnienie "Apple" jako sklepu od "Apple" jako owocu, w zależności od kontekstu bazy słów).

```

1 public async Task<int?> GetCategoryIdByDescriptionAsync(int userId, string description)
2 {
3     description = description.ToLower();
4
5     var allKeyWords = await _context.KeyWords
6         .Where(k => k.Category.UserId == userId && k.DeletedAt == null)
7         .ToListAsync();
8
9     var matchedKeyWord = allKeyWords
10        .OrderByDescending(k => k.Word.Length)
11        .FirstOrDefault(k => description.Contains(k.Word.ToLower()));
12
13     if (matchedKeyWord != null)
14     {
15         return matchedKeyWord.CategoryId;
16     }
17     return null;
18 }

```

Listing 7.8: Logika wyszukiwania w KeyWordRepository.cs

## 7.7 Procedury Bezpieczeństwa i Zarządzanie Kontem

System implementuje rygorystyczne procedury dotyczące zmiany danych wrażliwych. Poniżej przedstawiono implementację metod serwisowych obsługujących zmianę hasła, procedurę "Zapomniałem hasła" oraz zmianę adresu e-mail z wymuszoną ponowną weryfikacją.

### 7.7.1 Zmiana Hasła (ModifyPassword)

Metoda weryfikuje poprawność starego hasła przed nadpisaniem go nowym skrótem.

```

1 public async Task ModifyPasswordAsync(int userId, ModifyPasswordRequest request)
2 {
3     var user = await _userRepository.GetUserByIdWithDetailsAsync(userId);
4
5     if (user == null)
6     {
7         throw new Exception("User not found or access denied.");
8     }
9
10    if (!BCrypt.Net.BCrypt.Verify(request.OldPassword, user.PasswordHash))
11    {
12        throw new Exception("Current password is invalid.");
13    }
14
15    if (request.OldPassword == request.NewPassword)
16    {
17        throw new Exception("New password cannot be same as the old password.");
18    }
19
20    user.PasswordHash = BCrypt.Net.BCrypt.HashPassword(request.NewPassword);
21    user.UpdatedAt = DateTime.UtcNow;
22
23    await _userRepository.UpdateAsync(user);
24    await _emailService.SendEmailAsync(user.Email, "Reset Password - CashFlow", "<h1>Your
    password has been changed, thank you for supporting our project</h1>");
25 }

```

Listing 7.9: UserService.cs - Zmiana hasła

## 7.7.2 Procedura Resetowania Hasła

Proces składa się z dwóch etapów: żądania resetu (generowanie tokena) oraz potwierdzenia (ustawienie nowego hasła).

```

1 public async Task RequestPasswordResetAsync(string email)
2 {
3     var user = await _userRepository.GetUserByEmailOrNicknameAsync(email);
4     if (user == null)
5     {
6         return;
7     }
8
9     var resetGUID = Guid.NewGuid().ToString();
10    user.PasswordResetToken = resetGUID;
11    user.ResetTokenExpiresAt = DateTime.UtcNow.AddHours(1);
12
13    await _userRepository.UpdateAsync(user);
14    var frontendUrl = _configuration["AppUrls:FrontendUrl"];
15    await _emailService.SendEmailAsync(user.Email, "Reset Password - CashFlow", $"<h1>
    Welcome {user.FirstName} {user.LastName}</h1><br>To reset, click the link below:<br>
    <a href='{frontendUrl}/api/confirm-password-reset?token={resetGUID}'>RESET PASSWORD
    </a>");
16 }
17
18 public async Task ResetPasswordConfirmAsync(ResetPasswordRequest request)
19 {
20     var user = await _userRepository.GetUserByPasswordResetTokenAsync(request.Token);
21
22     if (user == null || user.ResetTokenExpiresAt < DateTime.UtcNow)
23     {
24         throw new Exception("Invalid or expired token.");
25     }
26
27     user.PasswordHash = BCrypt.Net.BCrypt.HashPassword(request.NewPassword);
28     user.PasswordResetToken = null;
29     user.ResetTokenExpiresAt = null;
30
31     await _userRepository.UpdateAsync(user);
32     await _emailService.SendEmailAsync(user.Email, "Reset Password - CashFlow", "<h1>Your
    password has been changed, thank you for supporting our project</h1>");
33 }

```

Listing 7.10: UserService.cs - Logika resetowania hasła

### 7.7.3 Zmiana Adresu E-mail

Zmiana adresu e-mail wymaga potwierdzenia własności nowej skrzynki pocztowej poprzez token weryfikacyjny.

```
1 public async Task RequestEmailChangeAsync(int userId, string newEmail)
2 {
3     var user = await _userRepository.GetUserByIdWithDetailsAsync(userId);
4
5     if (user == null)
6     {
7         throw new Exception("User not found or access denied.");
8     }
9
10    if (await _userRepository.IsEmailTakenAsync(newEmail))
11    {
12        throw new Exception("This email is already taken.");
13    }
14
15    var token = Guid.NewGuid().ToString();
16
17    user.NewEmailPending = newEmail;
18    user.EmailChangeToken = token;
19    user.EmailChangeTokenExpiresAt = DateTime.UtcNow.AddHours(1);
20
21    await _userRepository.UpdateAsync(user);
22    var frontendUrl = _configuration["AppUrls:FrontendUrl"];
23    await _emailService.SendEmailAsync(newEmail, "Confirm your new email - CashFlow", $"<
24    h1>Welcome {user.FirstName} {user.LastName}</h1><br>To confirm your new Email, click
25    the link below:<br> <a href=\"{frontendUrl}/api/confirm-email-change?token={token}\">
26    CHANGE EMAIL</a>");
27 }
28
29 public async Task EmailChangeConfirmAsync(string token)
30 {
31     var user = await _userRepository.GetUserByEmailChangeTokenAsync(token);
32
33     if (user == null || user.EmailChangeTokenExpiresAt < DateTime.UtcNow)
34     {
35         throw new Exception("Invalid or expired email change token.");
36     }
37
38     user.Email = user.NewEmailPending!;
39
40     user.NewEmailPending = null;
41     user.EmailChangeToken = null;
42     user.EmailChangeTokenExpiresAt = null;
43     user.UpdatedAt = DateTime.UtcNow;
44
45     await _emailService.SendEmailAsync(user.Email, "Email Changed - CashFlow", "<h1>Your
46     Email has been changed, thank you for supporting our project</h1>");
47     await _userRepository.UpdateAsync(user);
48 }
```

Listing 7.11: UserService.cs - Dwuetapowa zmiana e-maila

# Rozdział 8

## Warstwa Infrastruktury i Walidacji

W celu zapewnienia spójności danych oraz integracji z usługami zewnętrznymi, zaimplementowano dedykowane komponenty infrastrukturalne.

### 8.1 Konfiguracja Relacji Obiektowo-Relacyjnych (ORM)

Klasa kontekstu bazy danych 'CashFlowDbContext' zawiera metodę 'OnModelCreating', która definiuje precyzyjne mapowanie obiektów na tabele relacyjne. Skonfigurowano tu klucze główne, relacje jeden-do-wielu (One-to-Many) oraz indeksy unikalne.

```
1 using CashFlow.Domain.Models;
2 using Microsoft.EntityFrameworkCore;
3 using System.Collections.Generic;
4 using System.Security.Principal;
5
6 namespace CashFlow.Infrastructure.Data
7 {
8     public class CashFlowDbContext : DbContext
9     {
10         public CashFlowDbContext(DbContextOptions<CashFlowDbContext> options)
11             : base(options)
12         {
13
14
15             public DbSet<User> Users { get; set; }
16             public DbSet<Currency> Currencies { get; set; }
17             public DbSet<Account> Accounts { get; set; }
18             public DbSet<Category> Categories { get; set; }
19             public DbSet<Limit> Limits { get; set; }
20             public DbSet<Transaction> Transactions { get; set; }
21             public DbSet<RecTransaction> RecTransactions { get; set; }
22             public DbSet<Keyword> Keywords { get; set; }
23             public DbSet<Notification> Notifications { get; set; }
24
25             protected override void OnModelCreating(ModelBuilder modelBuilder)
26             {
27                 modelBuilder.Entity<User>().ToTable("users");
28                 modelBuilder.Entity<User>().HasIndex(u => u.Email).IsUnique();
29
30                 modelBuilder.Entity<Currency>().ToTable("currencies");
31
32                 modelBuilder.Entity<Account>().ToTable("accounts");
33
34                 modelBuilder.Entity<Category>().ToTable("categories");
35
36                 modelBuilder.Entity<Limit>().ToTable("limits");
37
38                 modelBuilder.Entity<Transaction>().ToTable("transactions");
39
40                 modelBuilder.Entity<RecTransaction>().ToTable("rec_transactions");
41
42                 modelBuilder.Entity<Keyword>().ToTable("key_words");
```

```

43     modelBuilder.Entity<KeyWord>()
44     .HasIndex(k => new { k.UserId, k.Word })
45     .IsUnique();
46
47     modelBuilder.Entity<Notification>().ToTable("notifications");
48
49     modelBuilder.Entity<Category>()
50     .HasOne(c => c.User)
51     .WithMany(u => u.Categories)
52     .HasForeignKey(c => c.UserId)
53     .IsRequired();
54
55     modelBuilder.Entity<KeyWord>()
56     .HasOne(k => k.Category)
57     .WithMany(c => c.KeyWords)
58     .HasForeignKey(k => k.CategoryId)
59     .IsRequired();
60
61     modelBuilder.Entity<Account>()
62     .HasOne(a => a.User)
63     .WithMany(u => u.Accounts)
64     .HasForeignKey(a => a.UserId)
65     .IsRequired();
66
67     modelBuilder.Entity<Account>()
68     .HasOne(a => a.Currency)
69     .WithMany(c => c.Accounts)
70     .HasForeignKey(a => a.CurrencyCode)
71     .IsRequired();
72
73     modelBuilder.Entity<Transaction>()
74     .HasOne(t => t.Category)
75     .WithMany(c => c.Transactions)
76     .HasForeignKey(t => t.CategoryId)
77     .IsRequired();
78
79     modelBuilder.Entity<RecTransaction>()
80     .HasOne(rt => rt.Category)
81     .WithMany(c => c.RecTransactions)
82     .HasForeignKey(rt => rt.CategoryId)
83     .IsRequired();
84
85     modelBuilder.Entity<Notification>()
86     .HasOne(n => n.User)
87     .WithMany(u => u.Notifications)
88     .HasForeignKey(n => n.UserId)
89     .IsRequired();
90
91     modelBuilder.Entity<Transaction>()
92     .HasOne(t => t.User)
93     .WithMany(u => u.Transactions)
94     .HasForeignKey(t => t.UserId)
95     .IsRequired();
96
97     modelBuilder.Entity<Transaction>()
98     .HasOne(t => t.Account)
99     .WithMany(a => a.Transactions)
100    .HasForeignKey(t => t.AccountId)
101    .IsRequired();
102
103    modelBuilder.Entity<RecTransaction>()
104    .HasOne(rt => rt.User)
105    .WithMany(u => u.RecTransactions)
106    .HasForeignKey(rt => rt.UserId)
107    .IsRequired();
108
109    modelBuilder.Entity<RecTransaction>()
110    .HasOne(rt => rt.Account)
111    .WithMany(a => a.RecTransactions)
112    .HasForeignKey(rt => rt.AccountId)
113    .IsRequired();
114
115    modelBuilder.Entity<Limit>()
116    .HasOne(l => l.Category)
117    .WithMany(c => c.Limits)
118    .HasForeignKey(l => l.CategoryId)
119    .IsRequired();
120
121    modelBuilder.Entity<Limit>()
122    .HasOne(l => l.Account)
123    .WithMany(a => a.Limits)
124    .HasForeignKey(l => l.AccountId)
125    .IsRequired();
126

```

```

127         base.OnModelCreating(modelBuilder);
128     }
129 }
130

```

Listing 8.1: Konfiguracja modelu w CashFlowDbContext.cs

## 8.2 Walidacja Danych Wejściowych

System wykorzystuje bibliotekę ‘FluentValidation’ do weryfikacji poprawności danych przesyłanych przez API. Poniższy walidator rejestracji wymusza silną politykę haseł (długość, znaki specjalne, cyfry) oraz poprawność formatu adresu e-mail.

```

1 using FluentValidation;
2 using CashFlow.Application.DTO.Requests;
3
4 namespace CashFlow.Application.Validators
5 {
6     public class RegisterRequestValidator : AbstractValidator<RegisterRequest>
7     {
8         public RegisterRequestValidator()
9         {
10             RuleFor(request => request.Password).NotEmpty();
11             RuleFor(request => request.Password).MinimumLength(8);
12             RuleFor(request => request.Password)
13                 .Matches("(?=.*\\d)")
14                 .WithMessage("Password must contain at least one number.");
15             RuleFor(request => request.Password)
16                 .Matches("(?=.*[a-z]")
17                 .WithMessage("Password must contain at least one small letter.");
18             RuleFor(request => request.Password)
19                 .Matches("(?=.*[A-Z]")
20                 .WithMessage("Password must contain at least one big letter.");
21             RuleFor(request => request.Password)
22                 .Matches(@"(?=.*[!@#%~&*_-])")
23                 .WithMessage("Password must contain at least one special character.");
24
25             RuleFor(request => request.Email).NotEmpty().EmailAddress();
26
27             RuleFor(request => request.FirstName).NotEmpty();
28             RuleFor(request => request.LastName).NotEmpty();
29             RuleFor(request => request.Nickname).NotEmpty();
30         }
31     }
32 }

```

Listing 8.2: Walidator rejestracji - RegisterRequestValidator.cs

## 8.3 Komunikacja SMTP (E-mail)

Serwis ‘EmailService’ odpowiada za wysyłkę wiadomości transakcyjnych (weryfikacja konta, reset hasła, powiadomienia). Wykorzystuje bibliotekę ‘MailKit’ do bezpiecznej komunikacji z serwerem pocztowym.

```

1 using CashFlow.Application.Interfaces;
2 using MailKit.Net.Smtp;
3 using MailKit.Security;
4 using Microsoft.Extensions.Configuration;
5 using MimeKit;
6 using MimeKit.Text;
7
8 namespace CashFlow.Application.Services
9 {
10     public class EmailService : IEmailService
11     {
12         private readonly IConfiguration _configuration;
13

```

```

14     public EmailService(IConfiguration configuration)
15     {
16         _configuration = configuration;
17     }
18
19     public async Task SendEmailAsync(string to, string subject, string body)
20     {
21         var email = new MimeMessage();
22
23         email.From.Add(new MailboxAddress(_configuration["EmailSettings:FromName"],
24             _configuration["EmailSettings:FromAddress"]));
25
26         email.To.Add(MailboxAddress.Parse(to));
27
28         email.Subject = subject;
29         email.Body = new TextPart(TextFormat.Html) { Text = body };
30
31         using var smtp = new SmtpClient();
32
33         try
34         {
35             await smtp.ConnectAsync(
36                 _configuration["EmailSettings:Host"],
37                 int.Parse(_configuration["EmailSettings:Port"]!),
38                 SecureSocketOptions.StartTls
39             );
40             await smtp.AuthenticateAsync(
41                 _configuration["EmailSettings:UserName"],
42                 _configuration["EmailSettings:Password"]
43             );
44
45             await smtp.SendAsync(email);
46         }
47         catch (Exception ex)
48         {
49             throw new Exception($"Cannot send email: {ex.Message}");
50         }
51         finally
52         {
53             await smtp.DisconnectAsync(true);
54         }
55     }
56 }

```

Listing 8.3: Implementacja EmailService.cs

# Rozdział 9

## Konfiguracja i Harmonogramowanie

System wymaga precyzyjnej konfiguracji startowej, w tym rejestracji serwisów w kontenerze DI oraz definicji zadań cyklicznych.

### 9.1 Konfiguracja Hangfire

Poniższy kod konfiguracyjny z warstwy API definiuje zadania *Recurring Jobs*. Zastosowanie wyrażeń CRON umożliwia elastyczne zarządzanie czasem wykonywania procesów w tle.

```
1 using Hangfire;
2 using CashFlow.Application.Interfaces;
3
4 namespace CashFlow.Api.Configuration
5 {
6     public static class HangfireConfiguration
7     {
8         public static void ConfigureRecurringJobs(this IApplicationBuilder app)
9         {
10             RecurringJob.AddOrUpdate<IRecTransactionService>(
11                 "process-recurring-transactions",
12                 service => service.ProcessPendingTransactionsAsync(),
13                 Cron.Daily(5, 0)
14             );
15
16             RecurringJob.AddOrUpdate<ICurrencyService>(
17                 "fetch-nbp-rates",
18                 service => service.SyncRatesAsync(),
19                 Cron.Daily(12, 0)
20             );
21         }
22     }
23 }
```

Listing 9.1: CashFlow.Api.Configuration.HangfireConfiguration.cs

### 9.2 Inicjalizacja Aplikacji (Program.cs)

Punkt wejścia aplikacji, w którym następuje rejestracja wszystkich zależności, konfiguracja Entity Framework (Npgsql), konfiguracja JWT Bearer Auth oraz Hangfire Server. Jest to kluczowy element architektury, spinający wszystkie warstwy systemu.

```
1 var builder = WebApplication.CreateBuilder(args);
2 var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
```

```

3
4 AppContext.SetSwitch("Npgsql.EnableLegacyTimestampBehavior", true);
5
6 builder.Services.AddDbContext<CashFlowDbContext>(options =>
7 {
8     options.UseNpgsql(connectionString);
9 });
10
11 builder.Services.AddScoped<IUserRepository, UserRepository>();
12 builder.Services.AddScoped<IAccountRepository, AccountRepository>();
13 builder.Services.AddScoped<ICategoryRepository, CategoryRepository>();
14 builder.Services.AddScoped<ITransactionRepository, TransactionRepository>();
15
16 builder.Services.AddScoped<IUserService, UserService>();
17 builder.Services.AddScoped<IJWTService, JWTService>();
18 builder.Services.AddScoped<IAccountService, AccountService>();
19 builder.Services.AddScoped<ITransactionService, TransactionService>();
20
21 builder.Services.AddHttpClient<ICurrencyFetcher, CurrencyFetcher>();
22
23 builder.Services.AddHangfire(config => config
24     .SetDataCompatibilityLevel(CompatibilityLevel.Version_180)
25     .UseSimpleAssemblyNameTypeSerializer()
26     .UseRecommendedSerializerSettings()
27     .UsePostgreSqlStorage(options => options.UseNpgsqlConnection(connectionString)));
28
29 builder.Services.AddHangfireServer();
30
31 var app = builder.Build();
32
33 app.UseHangfireDashboard();
34 app.ConfigureRecurringJobs();
35 app.MapControllers();
36 app.Run();

```

Listing 9.2: Fragment konfiguracji Program.cs

# Rozdział 10

## Interfejs Programistyczny (API)

Warstwa API (Application Programming Interface) stanowi punkt styku pomiędzy logiką biznesową backendu a aplikacją kliencką. Została zaimplementowana w oparciu o architekturę REST (Representational State Transfer).

### 10.1 Implementacja Kontrolerów

Kontrolery w systemie "CashFlow" pełnią rolę orkiestratorów. Nie zawierają logiki biznesowej, lecz delegują zadania do odpowiednich serwisów, dbając jednocześnie o poprawność kodów odpowiedzi HTTP oraz walidację wstępną.

#### 10.1.1 Kontroler Transakcji (TransactionsController)

Poniższy kod prezentuje implementację kontrolera obsługującego operacje finansowe. Warto zwrócić uwagę na wykorzystanie atrybutów `[Authorize]` (wymagane logowanie) oraz wstrzykiwanie zależności przez konstruktor.

```
1 using CashFlow.Application.DTO.Requests;
2 using CashFlow.Application.DTO.Responses;
3 using CashFlow.Application.Interfaces;
4 using Microsoft.AspNetCore.Authorization;
5 using Microsoft.AspNetCore.Mvc;
6 using System.Security.Claims;
7
8 namespace CashFlow.Api.Controllers
9 {
10     [Authorize]
11     [Route("api/")]
12     [ApiController]
13     public class TransactionsController : ControllerBase
14     {
15         private readonly ITransactionService _transactionService;
16
17         private int CurrentUserId => int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)
18             ?.Value!);
19
20         public TransactionsController(ITransactionService transactionService)
21         {
22             _transactionService = transactionService;
23         }
24
25         [HttpGet]
26         [Route("transactions-info")]
27         public async Task<ActionResult<TransactionResponse>> GetAccountTransactions(int
28             accountId)
```

```

27     {
28         var transactionsDto = await _transactionService.GetAccountTransactionsAsync(
29             CurrentUserId, accountId);
30         return Ok(transactionsDto);
31     }
32     [HttpPost]
33     [Route("create-new-transaction")]
34     public async Task<ActionResult> CreateNewTransaction([FromBody]
35         NewTransactionRequest request)
36     {
37         try
38         {
39             await _transactionService.CreateNewTransactionAsync(CurrentUserId,
40                 request);
41             return Created();
42         }
43         catch (Exception ex)
44         {
45             if (ex.Message.Contains("must be greater than 0"))
46             {
47                 return Conflict(new { message = ex.Message });
48             }
49             if (ex.Message.Contains("is required to"))
50             {
51                 return Conflict(new { message = ex.Message });
52             }
53             return StatusCode(500, new { message = "An internal server error occurred"
54         });
55     }
56     [HttpGet]
57     [Route("category-analytics")]
58     public async Task<ActionResult<List<CategoryAnalyticsResponse>>>
59         GetCategoryAnalytics([FromQuery] DateTime startDate, [FromQuery] DateTime endDate, [
60             FromQuery] string type)
61     {
62         var categoryAnalyticsDto = await _transactionService.
63             GetCategoryAnalyticsAsync(CurrentUserId, startDate, endDate, type);
64         return Ok(categoryAnalyticsDto);
65     }
66     [HttpGet]
67     [Route("balance-analytics")]
68     public async Task<ActionResult<List<MonthlyAnalyticsResponse>>>
69         GetBalanceAnalytics([FromQuery] DateTime startDate, [FromQuery] DateTime endDate)
70     {
71         var balanceAnalyticsDto = await _transactionService.GetMonthlyAnalyticsAsync(
72             CurrentUserId, startDate, endDate);
73         return Ok(balanceAnalyticsDto);
74     }
75 }

```

Listing 10.1: CashFlow.Api.Controllers.TransactionsController.cs

### 10.1.2 Kontroler Użytkowników (UsersController)

Odpowiada za procesy uwierzytelniania. Metoda 'LoginUser' zwraca token JWT, który jest następnie wykorzystywany przez klienta do autoryzacji kolejnych żądań.

```

1 [HttpPost]
2 [Route("login")]
3 [AllowAnonymous]
4 public async Task<ActionResult<LoginResponse>> LoginUser([FromBody] LoginRequest request)
5 {
6     try
7     {
8         var token = await _userService.LoginAsync(request);
9         return Ok(token);
10    }
11    catch (Exception ex)
12    {
13        return BadRequest(new { message = ex.Message });
14    }
15 }
16

```

```

17 [HttpGet]
18 [Route("login-info")]
19 public async Task<ActionResult<UserResponse>> GetUser()
20 {
21     try
22     {
23         var userDto = await _userService.GetUserByIdAsync(CurrentUserId);
24         return Ok(userDto);
25     }
26     catch (Exception ex)
27     {
28         if (ex.Message.Contains("User was not found"))
29         {
30             return NotFound();
31         }
32         throw;
33     }
34 }

```

Listing 10.2: Fragment UsersController.cs

## 10.2 Kontrakty Danych (DTO)

W systemie zastosowano separację modelu domenowego od modelu prezentacji. Obiekty DTO (Data Transfer Objects) definiują strukturę danych przesyłanych w formacie JSON.

### 10.2.1 Modele Żądań (Requests)

Przykład modelu służącego do utworzenia nowej transakcji.

```

1 namespace CashFlow.Application.DTO.Requests
2 {
3     public class NewTransactionRequest
4     {
5         public int? AccountId { get; set; }
6         public int? CategoryId { get; set; }
7         public decimal? Amount { get; set; }
8         public string? Description { get; set; }
9         public string? Type { get; set; }
10    }
11 }

```

Listing 10.3: CashFlow.Application.DTO.Requests.NewTransactionRequest.cs

### 10.2.2 Modele Odpowiedzi (Responses)

Przykład modelu zwracanego do klienta, zawierającego zagnieżdżony obiekt kategorii.

```

1 using System;
2
3 namespace CashFlow.Application.DTO.Responses
4 {
5     public class TransactionResponse
6     {
7         public int TransactionId { get; set; }
8         public decimal Amount { get; set; }
9         public string? Description { get; set; }
10        public DateTime Date { get; set; }
11        public string? Type { get; set; }
12
13        public CategoryResponse? Category { get; set; }
14    }
15 }

```

Listing 10.4: CashFlow.Application.DTO.Responses.TransactionResponse.cs

# Rozdział 11

## Interfejs Programistyczny (API)

System udostępnia pełne RESTful API, które służy jako warstwa komunikacyjna dla aplikacji klienckich. Poniżej przedstawiono kompletny wykaz punktów końcowych (end-points) pogrupowanych według kontrolerów i odpowiedzialności biznesowej.

### 11.1 Moduł Użytkownika (UserController)

Zarządzanie tożsamością, autoryzacją i profilem użytkownika.

- POST /api/register - Rejestracja nowego konta w systemie.
- POST /api/login - Uwierzytelnienie użytkownika i generacja tokena JWT.
- GET /api/login-info - Pobranie szczegółowych danych zalogowanego użytkownika.
- GET /api/verify - Weryfikacja adresu e-mail (endpoint publiczny).
- DELETE /api/delete-user - Usunięcie konta użytkownika (Soft Delete).
- PATCH /api/update-user - Aktualizacja danych profilowych (Imię, Nazwisko, Nick, Avatar).
- PATCH /api/modify-password - Zmiana hasła dla zalogowanego użytkownika.
- POST /api/request-password-reset - Inicjacja procedury odzyskiwania hasła.
- POST /api/confirm-password-reset - Ustawienie nowego hasła przy użyciu tokena resetującego.
- POST /api/request-email-change - Żądanie zmiany adresu e-mail.
- POST /api/confirm-email-change - Zatwierdzenie nowego adresu e-mail.

## 11.2 Moduł Rachunków (AccountsController)

Zarządzanie portfelami finansowymi.

- GET /api/accounts-info - Pobranie listy wszystkich kont użytkownika wraz z saldami.
- POST /api/create-new-account - Utworzenie nowego rachunku (np. w innej walucie).
- DELETE /api/delete-account - Usunięcie (dezaktywacja) rachunku.
- PATCH /api/update-account - Edycja nazwy lub ikony rachunku.
- GET /api/total-balance - Zwraca sumaryczne saldo wszystkich kont przeliczone na zadaną walutę (domyślnie PLN).

## 11.3 Moduł Transakcji (TransactionsController)

Obsługa operacji finansowych i analityki.

- GET /api/transactions-info - Pobranie historii transakcji dla wskazanego konta.
- POST /api/create-new-transaction - Rejestracja nowego przychodu lub wydatku.
- DELETE /api/delete-transaction - Anulowanie transakcji (z przywróceniem salda).
- PATCH /api/update-transaction - Edycja parametrów istniejącej transakcji.
- GET /api/category-analytics - Dane do wykresu kołowego (udział kategorii w wydatkach).
- GET /api/balance-analytics - Dane miesięczne (przychody vs wydatki vs balans).
- GET /api/daily-analytics - Szczegółowe dane dzienne do wykresów liniowych.

## 11.4 Moduł Transakcji Cyklicznych (RecTransactionsController)

Zarządzanie automatyzacją płatności.

- GET /api/rec-transaction-info - Lista zdefiniowanych płatności cyklicznych.
- POST /api/create-new-rec-transaction - Definicja nowej płatności cyklicznej.
- DELETE /api/delete-rec-transaction - Usunięcie definicji płatności.

- PATCH /api/update-rec-transaction - Aktualizacja parametrów (np. zmiana kwoty lub częstotliwości).

## 11.5 Moduł Kategorii (CategoryController)

Organizacja wydatków.

- GET /api/categories-info - Pobranie drzewa kategorii użytkownika.
- POST /api/create-new-category - Utworzenie nowej kategorii własnej.
- DELETE /api/delete-category - Usunięcie kategorii.
- PATCH /api/update-category - Edycja nazwy, koloru lub ikony kategorii.

## 11.6 Moduł Słów Kluczowych (KeywordController)

Konfiguracja silnika autokategoryzacji.

- GET /api/key-words-info - Pobranie słów kluczowych przypisanych do kategorii.
- POST /api/create-new-key-word - Dodanie nowego słowa kluczowego (reguły).
- DELETE /api/delete-key-word - Usunięcie słowa kluczowego.
- PATCH /api/update-key-word - Edycja słowa lub przypisania do kategorii.

## 11.7 Moduł Limitów i Budżetowania (LimitController)

Kontrola wydatków.

- GET /api/limits-info - Pobranie aktywnych limitów budżetowych.
- POST /api/create-new-limit - Ustalenie nowego limitu na kategorię.
- DELETE /api/delete-limit - Usunięcie limitu.
- PATCH /api/update-limit - Zmiana kwoty lub okresu obowiązywania limitu.

## 11.8 Moduł Powiadomień (NotificationController)

System alertów i komunikatów.

- GET /api/notifications-info - Pobranie listy powiadomień użytkownika.
- PATCH /api/set-notification-status-read - Oznaczenie pojedynczego powiadomienia jako przeczytane.
- PATCH /api/set-notification-status-read-for-all-unread - Oznaczenie wszystkich jako przeczytane ("Mark all as read").
- DELETE /api/delete-notification - Trwałe usunięcie powiadomienia.

## 11.9 Moduł Walutowy (CurrencyController)

Dane referencyjne NBP.

- GET /api/currencies-info - Pobranie aktualnej tabeli kursów walut dostępnych w systemie.

# Rozdział 12

## Instrukcja Instalacji i Konfiguracji Środowiska

Niniejszy rozdział zawiera szczegółowe wytyczne dotyczące przygotowania środowiska deweloperskiego oraz uruchomienia aplikacji serwerowej "CashFlow".

### 12.1 Wymagania wstępne

Do poprawnego działania systemu wymagane jest zainstalowanie następujących komponentów:

- **.NET SDK:** Wersja 9.0 lub nowsza.
- **PostgreSQL:** Wersja 16.0 (lokalnie lub w kontenerze Docker).
- **IDE:** Visual Studio 2022 / JetBrains Rider / VS Code.
- **Klient API:** Postman lub Insomnia (opcjonalnie, do testów manualnych).

### 12.2 Konfiguracja bazy danych

Przed pierwszym uruchomieniem aplikacji należy skonfigurować połączenie z bazą danych.

1. Utwórz nową bazę danych w PostgreSQL o nazwie `CashFlow`.
2. Otwórz plik konfiguracyjny `appsettings.json` w projekcie `CashFlow.Api`.
3. Zmodyfikuj sekcję `ConnectionStrings`:

```

1 "ConnectionStrings": {
2   "DefaultConnection": "Host=localhost;Port=5432;Database=CashFlowDb;Username=postgres;
   Password=TwojeHaslo"
3 },
4 "JwtSettings": {
5   "Secret": "BardzoDlugiISkomplikowanySekretnyKluczDoPodpisuTokenow123!",
6   "Issuer": "CashFlowApi",
7   "Audience": "CashFlowClient",
8   "ExpiryMinutes": 60
9 }

```

Listing 12.1: Konfiguracja appsettings.json

## 12.3 Inicjalizacja schematu (Migracje)

System wykorzystuje Entity Framework Core Migrations do zarządzania strukturą bazy danych. Aby utworzyć tabele, należy wykonać następujące polecenie w terminalu (w katalogu projektu API):

```

1 dotnet ef database update

```

Listing 12.2: Aplikowanie migracji

Powyższa komenda automatycznie utworzy tabele: `Users`, `Accounts`, `Transactions`, `RecTransactions`, `Limits`, `Categories`, `KeyWords` oraz tabele systemowe biblioteki Hangfire.

**Rozwiązywanie problemów konfiguracyjnych:** W przypadku wystąpienia błędu wskazującego na brak rozpoznania polecenia `dotnet ef`, należy zainstalować globalne narzędzia Entity Framework Core CLI. W tym celu należy wykonać polecenie:

```

1 dotnet tool install --global dotnet-ef

```

Listing 12.3: Instalacja narzędzi EF Core CLI

## 12.4 Uruchomienie serwera

Aby uruchomić aplikację w trybie deweloperskim, należy użyć polecenia:

```

1 dotnet run --project CashFlow.Api

```

Listing 12.4: Start aplikacji

Po poprawnym uruchomieniu, dokumentacja Swagger UI będzie dostępna pod adresem:

<http://localhost:5205/swagger/index.html>

Panel administracyjny Hangfire (do monitorowania zadań w tle) dostępny jest pod adresem:

<http://localhost:5205/hangfire>

# Rozdział 13

## Wnioski

Zrealizowany projekt systemu "CashFlow" w warstwie backendowej spełnia wszystkie zdefiniowane wymagania funkcjonalne i нефункционалне. Architektura oparta na serwisach, repozytoriach i asynchronicznym przetwarzaniu zadań (Hangfire) stanowi solidny fundament pod dalszy rozwój. Kod źródłowy charakteryzuje się wysoką jakością i zgodnością z nowoczesnymi standardami .NET.

# Dodatek A

## Słownik pojęć i skrótów

<b>ACID</b>	(Atomicity, Consistency, Isolation, Durability) – zbiór właściwości gwarantujących poprawne przetwarzanie transakcji w bazach danych.
<b>API</b>	(Application Programming Interface) – interfejs programistyczny aplikacji pozwalający na komunikację między różnymi systemami.
<b>Cron</b>	Harmonogram zadań w systemach uniksowych, wykorzystywany w projekcie do planowania zadań cyklicznych Hangfire.
<b>DI</b>	(Dependency Injection) – wzorzec projektowy polegający na wstrzykiwaniu zależności do obiektów, co zwiększa modularność i testowalność kodu.
<b>DTO</b>	(Data Transfer Object) – prosty obiekt służący do przesyłania danych pomiędzy podsystemami oprogramowania.
<b>JWT</b>	(JSON Web Token) – standard przesyłania danych w formacie JSON, wykorzystywany do bezpiecznego uwierzytelniania użytkowników.
<b>ORM</b>	(Object-Relational Mapping) – technologia mapowania obiektowo-relacyjnego, pozwalająca na dostęp do relacyjnej bazy danych przy użyciu języka obiektowego.
<b>Soft Delete</b>	Technika usuwania danych polegająca na oznaczeniu rekordu jako nieaktywny (np. poprzez flagę <code>IsActive</code> lub datę usunięcia), zamiast fizycznego usunięcia go z bazy.
<b>Swagger</b>	Narzędzie do automatycznego generowania dokumentacji API oraz interaktywnego testowania endpointów.