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 - Frontend

Projekt zaliczeniowy przedmiotu Bazy Danych

Prowadzący:
mgr inż. Aleksander Wojtowicz

Rzeszów, 2026 r.

Spis treści

1 Wprowadzenie

1.1 Cel Projektu

Projekt `Cashflow` to nowoczesna aplikacja typu SPA (Single Page Application), służąca do kompleksowego zarządzania finansami osobistymi. System umożliwia użytkownikom monitorowanie wydatków, planowanie budżetu, definiowanie celów oszczędnościowych oraz analizę kondycji finansowej poprzez interaktywne wykresy.

Aplikacja została zaprojektowana z myślą o responsywności (RWD) oraz wysokiej wydajności, wykorzystując najnowsze standardy ekosystemu React.

1.2 Stos Technologiczny

Analiza pliku `package.json` wykazuje użycie nowoczesnego stacku technologicznego (stan na rok 2026).

Technologia	Opis i Wersja
React 19.2	Najnowsza wersja biblioteki UI, wspierająca współbieżny model renderowania (Concurrent Mode).
TypeScript 5.9	Nadzbiór JavaScript zapewniający statyczne typowanie, co zwiększa stabilność kodu i ułatwia refaktoryzację.
Vite 7.2	Narzędzie budujące (Build Tool) nowej generacji, wykorzystujące plugin SWC do błyskawicznej kompilacji.
Bootstrap 5.3	Framework CSS odpowiadający za system siatki (Grid) i bazowe style responsywne.
Axios 1.13	Klient HTTP oparty na Promise, służący do komunikacji z API backendowym.
MUI X Charts 8.23	Biblioteka do wizualizacji danych (wykresy kołowe, słupkowe).
GSAP 3.14	Zaawansowana biblioteka do animacji interfejsu (GreenSock Animation Platform).

1.3 Uruchomienie Środowiska

Aplikacja wykorzystuje menedżer pakietów `npm`. Podstawowe skrypty zdefiniowane w konfiguracji:

```
# Instalacja zaleznosci
npm install

# Uruchomienie serwera deweloperskiego (z HMR)
npm run dev

# Budowanie wersji produkcyjnej (TypeScript check + Build)
npm run build
```

Listing 1: Skrypty startowe

2 Struktura Projektu

Poniżej przedstawiono drzewo katalogów projektu, obrazujące podział na warstwę logiczną, prezentacyjną oraz konfigurację.

```
src/
├── api/ ..... Konfiguracja klienta HTTP (Axios)
├── assets/ ..... Zasoby statyczne (grafiki, dzwieki)
├── components/ ..... Komponenty wspoldzielone
│   ├── Auth/ ..... Logika autoryzacji (ProtectedRoute)
│   ├── Layout/ ..... Uklad strony (Header, Sidebar)
│   └── UI/ ..... Biblioteka kontrolek (Input, Select)
├── contexts/ ..... Globalny stan aplikacji
├── hooks/ ..... Wlasne hooki (useLoading, useTransactions)
├── pages/ ..... Glowne widoki (Pages)
│   ├── DashboardPage/ ..... Panel uzytkownika po zalogowaniu
│   ├── LoginPage/ ..... Ekran logowania
│   └── HomePage/ ..... Landing Page
├── App.tsx ..... Glowny router aplikacji
└── main.tsx ..... Punkt wejscia (Entry Point)
```

3 Architektura Systemu

3.1 Routing i Bezpieczeństwo

Nawigacja w aplikacji oparta jest na bibliotece `react-router-dom` (v7). W pliku `App.tsx` zdefiniowano wyraźny podział na strefy dostępu:

1. **Strefa Publiczna:** Dostępna dla niezalogowanych użytkowników (Landing Page, Logowanie, Rejestracja).
2. **Strefa Chroniona (Dashboard):** Dostępna wyłącznie po autoryzacji.

Mechanizm ochrony realizowany jest przez komponent `ProtectedRoute`. Sprawdza on flagę `isAuthenticated` w globalnym kontekście. Jeśli użytkownik nie posiada aktywnej sesji, następuje automatyczne przekierowanie do widoku logowania.

```
return (
    isLoading ? <Spinner /> :
    requireAuth ? (
        isAuthenticated ? children : <Navigate to='/login' />
    ) : (
        isAuthenticated ? <Navigate to='/dashboard' /> : children
    )
);
```

Listing 2: Logika ProtectedRoute.tsx

3.2 Warstwa Danych (API)

Komunikacja z backendem odbywa się za pośrednictwem biblioteki **Axios**. Instancja klienta została skonfigurowana w pliku `api/api.ts` i zawiera:

- **Base URL:** `http://localhost:5205/api/`

- **Request Interceptor:** Automatycznie pobiera token JWT z `localStorage` i dołącza go do nagłówka `Authorization` każdego zapytania.
- **Response Interceptor:** Globalna obsługa błędów. W przypadku otrzymania statusu `401 Unauthorized` (wygaśnięcie tokena), aplikacja automatycznie wylogowuje użytkownika.

3.3 Zarządzanie Stanem (Context API)

Aplikacja wykorzystuje wbudowany w React mechanizm kontekstów do zarządzania danymi globalnymi, unikając zewnętrznych bibliotek typu Redux dla zachowania lekkości.

- **AuthContext:** Przechowuje obiekt zalogowanego użytkownika oraz metody `login/logout`.
- **AccountContext:** Kluczowy dla logiki finansowej. Przechowuje informacje o aktualnie wybranym portfelu (konto bankowe, gotówka). Zmiana konta w tym kontekście powoduje odświeżenie wykresów w całej aplikacji.
- **ToastContext:** System powiadomień *pop-up*, umożliwiający wywołanie komunikatu z dowolnego miejsca w kodzie.

4 Strona Główna (Landing Page)

Widok `HomePage` stanowi punkt wejścia dla nowych użytkowników. Został zaprojektowany w modelu *One-Page Layout* z płynnym przewijaniem (`scroll-behavior: smooth`).

4.1 Struktura Modułowa

Strona składa się z niezależnych sekcji komponentowych:

- **Hero Section:** Główny baner powitalny z animowanym tekstem i grafikami (biblioteka ReactBits).
- **Features:** Prezentacja kluczowych funkcji (Planowanie, Analiza, Cele) w układzie Flex Grid.
- **Testimonials:** Sekcja opinii klientów z dynamicznym ładowaniem zasobów.

4.2 Dynamiczne Zarządzanie Zasobami

W komponencie `Testimonials.tsx` zastosowano zaawansowany mechanizm Vite do importowania grafik. Zamiast ręcznie importować każdy awatar, użyto funkcji `import.meta.glob`, co pozwala na automatyczne załadowanie wszystkich plików z katalogu asetów.

```
const avatars = import.meta.glob(
    '../../../../assets/Testimonial/*',
    { eager: true, import: 'default', query: '?url' }
);
```

Listing 3: Dynamiczny import w Testimonials.tsx

5 Biblioteka Komponentów UI

W celu zapewnienia spójności wizualnej oraz obsługi trybu ciemnego (Dark Mode), zrezygnowano z domyślnych kontrolek przeglądarki na rzecz autorskich komponentów React.

5.1 Animacje (AnimatedContent)

Komponent `AnimatedContent` to wrapper (HOC - Higher Order Component) integrujący bibliotekę **ReactBits**.

- **Działanie:** Elementy owinięte tym komponentem są domyślnie ukryte (`opacity: 0`). Animacja wejścia (np. wjazd od dołu, fade-in) uruchamia się dopiero w momencie przewinięcia strony do danego elementu.
- **Parametryzacja:** Komponent przyjmuje propsy takie jak `direction` (kierunek), `delay` (opóźnienie) czy `scale`, co pozwala na tworzenie różnorodnych efektów bez duplikowania kodu.

5.2 Inteligentne Pola (Input)

Komponent `Input.tsx` to uniwersalna nakładka na tag `<input>`.

- **Obsługa haseł:** Jeśli typ pola to `password`, komponent automatycznie renderuje przycisk "oczka" (bi-eye), pozwalający użytkownikowi na podgląd wpisanego hasła. Stan widoczności (`showPassword`) jest zarządzany lokalnie wewnątrz komponentu.
- **Walidacja:** Komponent przyjmuje prop `error`. Jeśli wystąpi błąd walidacji, pole otrzymuje czerwoną ramkę (klasa `is-invalid`), a komunikat błędu wyświetlany jest pod spodem.

5.3 Selecty (CurrencySelect / CustomSelect)

Ze względu na ograniczone możliwości stylowania natywnego tagu `<select>`, stworzono własne rozwiązanie typu Dropdown.

- **Click Outside:** Zastosowano hook `useEffect` do wykrywania kliknięć poza obszar listy, co powoduje jej automatyczne zamknięcie.
- **Dostępność:** Komponenty obsługują nawigację klawiaturą oraz atrybuty ARIA, zachowując standardy dostępności (a11y).

6 Układ Aplikacji (Layout)

Panel użytkownika (`Dashboard`) wykorzystuje responsywny układ typu *Admin Dashboard*, składający się z paska bocznego (Sidebar) oraz nagłówka (Header).

6.1 Inteligentny Nagłówek (Header)

Komponent `Header.tsx` pełni podwójną rolę: nawigacji na stronie głównej oraz paska narzędzi w panelu użytkownika.

- **Animowany Slider:** W menu nawigacyjnym zastosowano efekt "pływającego tła". Pozycja i szerokość aktywnego elementu są obliczane dynamicznie przy użyciu `getBoundingClientRect`
- **Efekt Sticky:** Podczas przewijania strony, nagłówek zmienia swoje tło na półprzezroczyste (`backdrop-filter: blur`), co poprawia czytelność treści pod spodem.

```
if (activeElement && sliderRef.current) {
    const rect = activeElement.getBoundingClientRect();
    const containerRect = activeElement.parentElement?.getBoundingClientRect
    ();
    // Ustawienie nowej pozycji i szerokosci
    sliderRef.current.style.width = `${Math.min(rect.width + 26,
    containerRect.width)}px`;
    sliderRef.current.style.left = `${newLeft - 12}px`;
}
```

Listing 4: Obliczanie pozycji slidera w Header.tsx

6.2 Responsywny Pasek Boczny (Sidebar)

Komponent `Sidebar` dostosowuje się do szerokości okna przeglądarki (wykorzystując hook `useWindowWidth`):

- **Desktop (>768px):** Pasek może być zwinięty (tylko ikony, 60px) lub rozwinięty (ikony + tekst, 270px). Zmiana szerokości jest animowana płynnym przejściem CSS.
- **Mobile (<768px):** Pasek jest domyślnie ukryty. Po aktywacji wysuwa się jako warstwa wierzchnia (Overlay), przesłaniając treść aplikacji.

7 Moduł Uwierzytelniania (Authentication)

Aplikacja posiada rozbudowany system zarządzania tożsamością, obejmujący logowanie, rejestrację oraz odzyskiwanie dostępu.

7.1 Logowanie (LoginPage)

Formularz logowania obsługuje walidację po stronie klienta i komunikację z API.

- **Blokada UI:** Podczas wysyłania żądania, hook `useLoading` blokuje interfejs i wyświetla spinner, zapobiegając wielokrotnemu kliknięciu.
- **Obsługa Błędów:** Nieudane próby logowania skutkują wyświetleniem komunikatu w komponencie `alert-danger`.

7.2 Rejestracja (RegisterPage)

Rejestracja wymaga podania szczegółowych danych (Imie, Nazwisko, Nick). Zastosowano rygorystyczną walidację Regex:

- **Imię/Nazwisko:** Dozwolone tylko litery (w tym polskie znaki). Cyfry i znaki specjalne są blokowane.
- **Hasło:** Wymagane min. 8 znaków, duża litera oraz cyfra.

```
if (/[^a-zA-Z\s]\/.test(formData.firstName)) {
    err.firstName = 'Imie nie moze zawierac cyfr ani znakow specjalnych';
}
```

Listing 5: Walidacja imienia (Regex)

7.3 Resetowanie Hasła

Proces ten jest dwuetapowy:

1. **Żądanie (RequestPasswordReset):** Użytkownik podaje email, na który wysyłany jest link.
2. **Potwierdzenie (ConfirmPasswordReset):** Użytkownik wchodzi w link z tokenem. Token jest pobierany z parametrów URL (`searchParams.get('token')`) i wysyłany wraz z nowym hasłem do API.

8 Panel Użytkownika (Dashboard)

Widok `DashboardPage` stanowi główny kontener aplikacji po zalogowaniu. Jego kluczową rolą jest nie tylko wyświetlanie interfejsu, ale również inicjalizacja kontekstu finansowego.

8.1 Kontekst Konta (AccountContext)

Ponieważ użytkownik może posiadać wiele portfeli (np. "Portfel", "Bank", "Oszczędności"), aplikacja musi wiedzieć, którego konta dotyczą wyświetlane dane. Za tę logikę odpowiada `AccountContext`.

- **Persystencja Danych:** Wybrane konto jest zapisywane w `localStorage` pod kluczem `selectedAccount`. Dzięki temu, po odświeżeniu strony, użytkownik nie traci kontekstu swojej pracy.
- **Globalne Odświeżanie:** Zmiana konta w pasku nagłówka automatycznie triggeruje pobranie nowych danych dla wykresów i list transakcji w całej aplikacji (mechanizm reaktywny Reacta).

```
const [account, setAccountState] = useState<AccountContextProps | null>(()
    => {
    const savedAccount = localStorage.getItem('selectedAccount');
    return savedAccount ? JSON.parse(savedAccount) : null
});
```

Listing 6: Inicjalizacja stanu w AccountContext.tsx

9 Zarządzanie Kontami

Moduł `Accounts` umożliwia tworzenie, edycję oraz przeglądanie dostępnych środków.

9.1 Responsywny Interfejs (Mobile First)

Szczególną uwagę poświęcono obsłudze urządzeń mobilnych. Lista kont na desktopie wyświetlana jest w siatce, natomiast na telefonach edycja konta odbywa się za pomocą panelu wysuwanego z dołu ekranu (*Bottom Sheet*).

W pliku `Accounts.module.css` zdefiniowano animację `slideUp`, która nadaje aplikacji odczucie natywności ("app-like feel").

```
.bottomSheet {
    border-top-left-radius: 1.5rem;
    border-top-right-radius: 1.5rem;
    transform: translateY(100%);
    animation: slideUp 0.3s forwards cubic-bezier(0.16, 1, 0.3, 1);
    position: relative;
    max-height: 80vh;
}
```

Listing 7: Animacja panelu mobilnego (CSS)

9.2 Tworzenie Konta (AccountCreator)

Formularz tworzenia konta wykorzystuje autorski komponent wyboru ikon. Użytkownik wybiera symbol graficzny (np. `bi-piggy-bank`) z siatki przycisków. Waluta jest wybierana przez komponent `CurrencySelect`, który w przypadku błędu API przechodzi w tryb offline (korzystając z danych "zaszytych" w kodzie).

10 Moduł Analityczny (Charts)

Sekcja `Charts` służy do wizualizacji kondycji finansowej. Wykorzystano bibliotekę `@mui/x-charts` do generowania wykresów liniowych i kołowych.

10.1 Dostosowanie do Trybu Ciemnego

Domyślne style biblioteki MUI zostały nadpisane, aby pasowały do ciemnego motywu aplikacji (*Dark Mode*). Linie siatki oraz osie wykresów otrzymały kolory zgodne z paletą `-border-emphasis` (#525252).

10.2 Obsługa Pustych Stanów (Empty States)

W przypadku braku transakcji w danym okresie, aplikacja nie wyświetla pustego wykresu (co mogłoby wyglądać jak błąd), lecz dedykowany komponent informacyjny.

```
{dailyData.length > 0 ? (
    <LineChart
        series={[ { data: dailyData.map((d) => d.balance), ... } ]}
        // ... konfiguracja wykresu
    />
) : (
    <div className="opacity-50 text-center">
        <i className="bi bi-inbox fs-1 mb-3"></i>
        <p>Brak danych dziennych</p>
    </div>
)}
```

Listing 8: Warunkowe renderowanie w Charts.tsx

11 Logika Biznesowa i Finanse

11.1 Sterowanie i Filtrowanie (SidebarControl)

Komponent `SidebarControl` pełni funkcję centrum dowodzenia w Dashboardzie. Umożliwia on użytkownikowi definiowanie ram czasowych dla wyświetlanych danych.

- **Zakres Dat:** Wykorzystano natywne pola `<input type="date">`. Aby zachować spójność z ciemnym motywem ("Dark Mode"), domyślna ikona kalendarza przeglądarki została nadpisana przy użyciu obrazka SVG zakodowanego w Base64 (Data URI) w pliku CSS.
- **Podsumowanie:** Komponent dynamicznie oblicza i wyświetla saldo oraz sumę wydatków dla wybranego okresu.

```
.date[type="date"]::-webkit-calendar-picker-indicator {
    background: url("data:image/svg+xml,... fill='%23DEDEDE' ...");
    cursor: pointer;
}
```

Listing 9: Stylowanie kalendarza w SidebarControl.module.css

11.2 Obsługa Transakcji (useTransactions)

Logika pobierania operacji finansowych została wydzielona do dedykowanego hooka `useTransactions`. Zapewnia to separację warstwy prezentacji od warstwy danych.

Kluczowe mechanizmy:

1. **Pobieranie Danych:** Hook pobiera wszystkie transakcje dla danego konta (`/transactions-info`) przy każdej zmianie ID konta.
2. **Optymalizacja (Memoizacja):** Filtrowanie transakcji po dacie oraz typie (przychód/wydatek) odbywa się po stronie klienta z wykorzystaniem `useMemo`. Dzięki temu zmiana zakresu dat w `SidebarControl` nie powoduje zbędnych zapytań do API, a jedynie przeliczenie widoku w pamięci przeglądarki.

11.3 Kategorie i Budżetowanie

Moduł `Categories` to nie tylko słownik nazw, ale zaawansowane narzędzie do kontroli wydatków.

11.3.1 Limity Budżetowe

Użytkownik może zdefiniować limit kwotowy dla każdej kategorii w ujęciu miesięcznym, kwartalnym lub rocznym. Aplikacja wizualizuje stopień wykorzystania budżetu za pomocą pasków postępu (Progress Bars):

- **Stan Normalny:** Pasek wypełnia się proporcjonalnie do wydatków, wyświetlając kwotę "Pozostało do wydania".
- **Przekroczenie:** Jeśli wydatki przekroczą 100% limitu, pasek zmienia kolor na czerwony (`bg-danger`), a użytkownik otrzymuje ostrzeżenie o kwocie przekroczenia.

```
// Obliczanie procentu wykorzystania
const percentage = Math.min((limit.currentAmount / limit.value) * 100, 100);

// Warunkowe renderowanie koloru
<div className={`progress-bar ${
    limit.currentAmount > limit.value ? 'bg-danger' : 'bg-success'
}`}
style={{ width: `${percentage}%` }}>
</div>
```

Listing 10: Logika wizualizacji limitu

11.3.2 Kreator Kategorii (Paginacja)

Ze względu na dużą liczbę dostępnych ikon (biblioteka Bootstrap Icons), w kreatorze kategorii (`CategoriesCreator`) zaimplementowano mechanizm paginacji. Użytkownik przegląda ikony stronami, co zapobiega przeciążeniu interfejsu (renderowanie setek elementów DOM jednocześnie).

12 Centrum Powiadomień (Notifications)

Moduł powiadomień służy do komunikacji systemu z użytkownikiem (np. alerty bezpieczeństwa, potwierdzenia operacji).

12.1 Statusy Wiadomości

Każde powiadomienie posiada atrybut `status`, który może przyjmować wartość `'read'` lub `'unread'`.

- **Wizualizacja:** Nieprzeczytane wiadomości są wyróżnione grubszą, kolorową krawędzią z lewej strony (`border-left: 3px solid primary`).
- **Interakcja:** Kliknięcie przycisku "odfajkowania" wysyła żądanie `PATCH` do API, zmieniając status wiadomości w bazie danych.

```
.unread {
    border-left: 3px solid var(--bs-primary) !important;
}
.notificationCard:hover {
    transform: translateX(-4px); /* Mikro-interakcja */
}
```

Listing 11: Wyróżnienie nieprzeczytanej wiadomości (CSS)

13 Automatyzacja (Recurring Transactions)

Aplikacja wspiera definiowanie operacji powtarzalnych (np. subskrypcje, czynsz). Zaimplementowano dwa tryby działania, rozróżniane wizualnie za pomocą kolorów:

1. **Tryb Automatyczny (Zielony):** System sam tworzy transakcję w bazie danych, gdy nadejdzie termin płatności. Użytkownik nie musi podejmować żadnej akcji.
2. **Tryb Przypomnienia (Niebieski):** System jedynie wysyła powiadomienie o nadchodzącej płatności, nie ingerując w saldo konta.

14 Konfiguracja i Bezpieczeństwo (Settings)

Komponent `Settings.tsx` pełni rolę "routera modalnego". Zamiast tworzyć osobne podstrony dla każdego ustawienia, zastosowano jeden kontener, który dynamicznie renderuje odpowiedni formularz wewnątrz okna modalnego.

Dzięki stanowi `display`, aplikacja wie, który formularz wyświetlić:

- `'accountDelete'` → Formularz usuwania konta.
- `'accountEditPassword'` → Zmiana hasła.
- `'emailChange'` → Procedura zmiany adresu e-mail.

```
<div className="modal-content">
    {display === 'accountDelete' &&
        <DeleteUserForm deleteFunction={handleDeleteAccount} />
    }
    {display === 'accountEditPassword' &&
        <ChangePasswordForm editFunction={handleEditPassword} />
    }
    {/* Pozostałe formularze... */}
</div>
```

Listing 12: Routing wewnątrz modala Settings

15 Podsumowanie Projektu

Projekt **CashFlow** to kompletna, produkcyjna aplikacja finansowa, zrealizowana zgodnie z nowoczesnymi standardami inżynierii oprogramowania (rok 2025/2026).

15.1 Kluczowe Osiągnięcia

- **Architektura:** Modularny podział kodu, wykorzystanie Custom Hooks do separacji logiki od widoku oraz Context API do zarządzania stanem globalnym.
- **Technologia:** Użycie najnowszego React 19 oraz TypeScript 5.9 zapewnia wysoką wydajność, bezpieczeństwo typów i łatwość utrzymania.
- **User Experience:** Aplikacja jest w pełni responsywna (Mobile First), obsługuje tryb ciemny (Dark Mode) i posiada zaawansowane animacje (ReactBits), co przekłada się na płynność działania ("App-like feel").
- **Bezpieczeństwo:** Zaimplementowano pełną ścieżkę autoryzacji (JWT), walidację danych po stronie klienta (Regex) oraz mechanizmy ochrony przed błędami (Error Boundaries, Interceptory Axios).

Dokumentacja obejmuje stan projektu na dzień 22 stycznia 2026.