



**Wyższa Szkoła Technologii  
Informatycznych w Katowicach**

# **System Zarządzania Budżetem Domowym**

Kierunek	Informatyka
Rok akademicki	2025/2026
Przedmiot	Projekt Systemu Informatycznego
Semestr	Piąty
Grupa	5ION1

<b>Dane wykonującego</b>	
Imię i nazwisko	Jakub Reczko
Nr albumu	09224
<b>Dane prowadzącego</b>	
	mgr Jacek Żywczyk
Data oddania	
Data zaliczenia	

# Spis treści

<b>Spis treści.....</b>	<b>1</b>
<b>Wprowadzenie do tematyki projektu.....</b>	<b>3</b>
<b>Zamierzony cel projektu.....</b>	<b>4</b>
Cel główny.....	4
Cele szczegółowe.....	4
<b>Wstępne założenia i uwarunkowania.....</b>	<b>5</b>
<b>Założone ograniczenia i możliwość ewaluacji projektu.....</b>	<b>6</b>
Ograniczenia techniczne.....	6
Ograniczenia organizacyjne i czasowe.....	7
<b>Chronologiczny plan pracy.....</b>	<b>8</b>
<b>Wymagania funkcjonalne.....</b>	<b>9</b>
Obsługa transakcji.....	9
Obsługa kategorii.....	10
Podstawowa analiza finansowa.....	11
Filtrowanie danych.....	12
<b>Przechowywanie danych.....</b>	<b>12</b>
Interfejs użytkownika.....	12
<b>Wymagania niefunkcjonalne.....</b>	<b>13</b>
Wymagania sprzętowe.....	13
Wymagania systemowe.....	13
<b>Wymagania organizacyjne.....</b>	<b>14</b>
<b>Wymagania dotyczące danych.....</b>	<b>15</b>
Tabela: Category.....	15
Tabela: Transaction.....	15
Tabela: Settings.....	16
Relacje między tabelami.....	17
Uzasadnienie wyboru struktur danych.....	17
Diagram bazy danych.....	18
<b>Metody pracy, narzędzia i techniki.....</b>	<b>19</b>
Technologie.....	19
Diagram Przypadków Użycia.....	20
Szkice projektowe.....	21
Opis Struktur.....	23
balance (Dashboard).....	23
transaction (Transakcja).....	24
category (Kategoria).....	25
settings (Ustawienia).....	26
Spis Metod.....	27
Warstwa serwisowa - services/transactions.ts.....	27
Warstwa akcji serwerowych - app/transactions/actions.ts.....	28
Warstwa serwisowa - services/categories.ts.....	29
Warstwa akcji serwerowych - app/categories/actions.ts.....	29

Warstwa serwisowa - services/settings.ts.....	30
Warstwa akcji serwerowych - app/settings/actions.ts.....	30
Warstwa akcji serwerowych - app/actions.ts.....	30
Funkcje pomocnicze - lib/format.ts.....	31
Funkcje komponentu - components/shared>SelectPeriod.tsx.....	32
<b>Wykaz Algorytmów.....</b>	<b>32</b>
Agregacja danych finansowych (Summary).....	32
<b>Użytkownie.....</b>	<b>34</b>
Proces uruchomieniowy.....	34
Proces testowy.....	36
Zrzuty ekranów.....	37
<b>Podsumowanie.....</b>	<b>39</b>
<b>Bibliografia.....</b>	<b>40</b>
<b>Spis Rysunków.....</b>	<b>41</b>
<b>Spis Tabel.....</b>	<b>42</b>

# **Wprowadzenie do tematyki projektu**

Zarządzanie budżetem domowym jest istotnym elementem codziennego życia.

Wielu użytkowników poszukuje prostych narzędzi, które pozwalają kontrolować wydatki, analizować wpływy oraz planować oszczędności w sposób intuicyjny i przejrzysty.

Dostępne na rynku aplikacje do planowania budżetu często oferują rozbudowane funkcje, które mogą przytłaczać użytkowników szukających prostoty. Wiele z nich wymaga również połączenia z Internetem, tworzenia kont lub udostępniania danych finansowych firmom trzecim, co budzi obawy o prywatność i bezpieczeństwo. Z tego powodu pojawia się potrzeba stworzenia lekkiego, lokalnego rozwiązania, które pozwoli użytkownikowi w pełni kontrolować własne dane finansowe.

Celem projektu jest stworzenie prostego programu komputerowego, który umożliwi:

- rejestrowanie transakcji finansowych,
- klasyfikowanie ich według kategorii,
- analizowanie przepływów finansowych w wybranych okresach,
- prezentację danych w formie zestawień i wykresów.

System ma na celu nie tylko ułatwienie zarządzania codziennymi wydatkami, lecz także kształcenie dobrych nawyków finansowych przez prowadzenie domowego budżetu. Wyciąganie wniosków z operacji zwiększa świadomość na temat struktury jego finansów.

System będzie aplikacją desktopową lub webową, z prostym interfejsem graficznym, dostępną dla jednego użytkownika lokalnie.

# Zamierzony cel projektu

## Cel główny

Głównym celem projektu jest opracowanie prostego, intuicyjnego oprogramowania wspierającego użytkownika w efektywnym zarządzaniu budżetem domowym.

System ma umożliwić wizualizację przepływów finansowych, planowanie wydatków oraz analizę stanu finansowego w różnych okresach czasowych.

Aplikacja ma stanowić praktyczne narzędzie, które pozwoli użytkownikowi zrozumieć strukturę własnych przychodów i kosztów oraz wprowadzić bardziej świadome decyzje finansowe w codziennym życiu.

## Cele szczegółowe

Aby zrealizować cel główny, projekt zakłada osiągnięcie następujących celów szczegółowych:

1. **Rejestracja transakcji finansowych** – stworzenie funkcji umożliwiającej wprowadzanie przychodów i wydatków z okrešeniem ich daty, kwoty, opisu oraz kategorii.
2. **Tworzenie i zarządzanie kategoriami budżetowymi** – użytkownik będzie mógł zdefiniować własne kategorie wydatków (np. jedzenie, mieszkanie, transport, rozrywka) oraz przypisywać do nich transakcje.
3. **Analiza stanu finansowego** – system będzie umożliwiał obliczanie bilansu w danym okresie (np. tygodniowym, miesięcznym, rocznym) oraz prezentację zestawień pokazujących strukturę wydatków i przychodów.
4. **Wizualizacja danych finansowych** – projekt przewiduje generowanie prostych wykresów słupkowych i kołowych, które ułatwią analizę nawyków finansowych i identyfikację obszarów, w których można wprowadzić oszczędności.
5. **Utrzymanie prostoty i czytelności interfejsu** – aplikacja ma być zrozumiała nawet dla osób bez doświadczenia technicznego, a jej obsługa nie powinna wymagać znajomości terminologii finansowej.
6. **Bezpieczeństwo i prywatność danych** – dane użytkownika będą przechowywane lokalnie, w bazie SQLite<sup>1</sup>, co eliminuje ryzyko ich udostępnienia podmiotom zewnętrznym.

---

<sup>1</sup> <https://sqlite.org/>

# Wstępne założenia i uwarunkowania

Projekt zakłada stworzenie prostego systemu do zarządzania budżetem domowym w warunkach akademickich, przy ograniczonym czasie realizacji i zasobach. Aplikacja będzie tworzona przez jedną osobę, w celu praktycznego zastosowania wiedzy zdobytej podczas studiów oraz doskonalenia umiejętności w zakresie programowania i projektowania systemów informatycznych.

1. Językiem programowania wykorzystanym do realizacji projektu będzie [Next.js](#)<sup>2</sup> (JavaScript/TypeScript) + [Drizzle ORM](#)<sup>3</sup> + [HeroUI](#)<sup>4</sup>
2. Aplikacja będzie działać lokalnie, z wykorzystaniem lekkiej bazy danych SQLite<sup>5</sup>.
3. System zostanie zaprojektowany zgodnie z zasadami programowania obiektowego.
4. Zastosowany zostanie prosty model projektowy – **model kaskadowy**<sup>6</sup> (Waterfall).
5. Diagramy zostaną wykonane za pomocą programu [Draw.io](#)<sup>7</sup>.
6. Dokumentacja zostanie wyeksportowana do formatu PDF.
7. Projekt będzie rozwijany etapowo, z raportowaniem postępów na każdych zajęciach.

---

<sup>2</sup> <https://nextjs.org/>

<sup>3</sup> <https://orm.drizzle.team/>

<sup>4</sup> <https://www.heroui.com/>

<sup>5</sup> <https://sqlite.org/>

<sup>6</sup> [https://pl.wikipedia.org/wiki/Model\\_kaskadowy](https://pl.wikipedia.org/wiki/Model_kaskadowy)

<sup>7</sup> <https://draw.io>

# Założone ograniczenia i możliwość ewaluacji projektu

Projekt realizowany w ramach zajęć akademickich posiada określone ograniczenia wynikające z warunków technicznych, organizacyjnych oraz czasowych. Głównym założeniem jest utrzymanie projektu w zakresie możliwym do wykonania przez jedną osobę, przy zachowaniu jednaczecznym zachowaniem terminów.

Podstawą jest czytelna struktura kodu, dostarczenie prostych funkcjonalności i kompletnej dokumentacji.

## Ograniczenia techniczne

W ograniczeniach technicznych wykazałem jakie funkcjonalności nie zostaną zrealizowane. Oto kilka potencjalnych funkcjonalności, które można dodać do systemu, gdyby zainwestować więcej czasu:

### Filtrowanie

- Możliwości filtrowania szczegółowego transakcji wprowadzonych do systemu
- Konfigurowanie szczegółowe filtrów dla list elementów wprowadzonych do systemu

### Notyfikacje

- Integrowanie dostawców usługi wysyłek SMS i Email
- Notyfikacje sms, email o przekroczeniu budżetów czy innych istotnych zdarzeniach w systemie

### Wizualizacja danych

- Zaawansowane wykresy liniowe obrazujące wydatki rozłożone w czasie
- Wykresy kołowe obrazujące proporcje kategorii w ramach budżetów

### Użytkownicy

- Jednostanowiskowość systemu oznacza, że aplikacja będzie przeznaczona dla jednego użytkownika i będzie działać lokalnie na jego maszynie.
- Brak funkcji logowania zmniejsza bezpieczeństwo aplikacji i podnosi ryzyko wypłynięcia danych.

### Bezpieczeństwo

- Integracja z zewnętrznymi API bankowymi oznacza, że wszelkie transakcje muszą zostać wprowadzone ręcznie.
- Zabezpieczenia kryptograficzne danych przechowywanych w systemie.
- Automatyczne generowanie kopii zapasowych.

## Inne

- Możliwość eksportu danych do pliku np PDF i XLS.
- Interfejs użytkownika będzie uproszczony (brak zaawansowanych animacji, responsywności).

## Ograniczenia organizacyjne i czasowe

- Prace nad projektem są ograniczone do kilku spotkań w semestrze (okres od października 2025 do stycznia 2026). Zakres funkcji musi być dostosowany do tego harmonogramu.
- Wszystkie elementy projektu (kod, dokumentacja, diagramy) muszą być wykonane samodzielnie, co ogranicza możliwość równoległej pracy nad większą ilością modułów.
- Projekt będzie realizowany z wykorzystaniem ogólnodostępnych narzędzi i bibliotek open source.
- Projekt ma charakter dydaktyczny, dlatego jego celem nie jest stworzenie komercyjnego produktu, lecz praktyczne zastosowanie wiedzy z zakresu inżynierii oprogramowania.

# Chronologiczny plan pracy

Praca na projektem została zaplanowana według terminów oddania poszczególnych elementów dokumentacji i projektu. Praca została podzielona na etapy które w przyrostowy sposób doprowadzą do dokończenia projektu w sztywnym terminie.

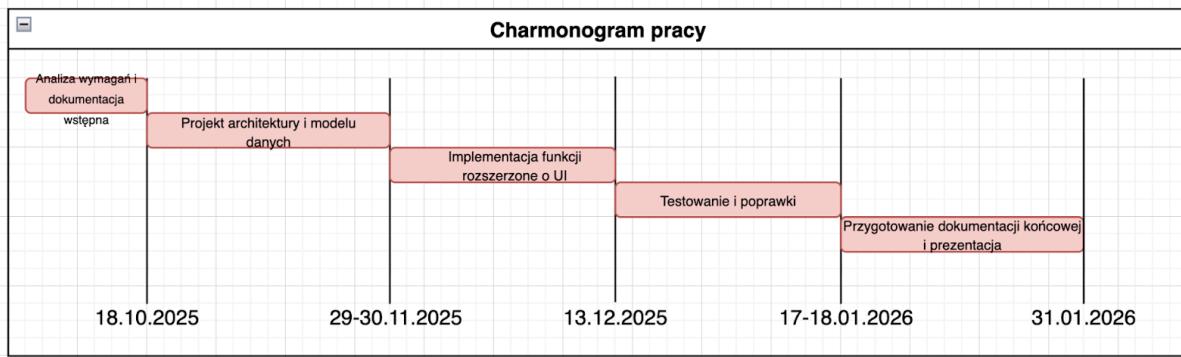


Tabela 1. przedstawiająca chronologię wykonywania projektu w stosunku do terminów ćwiczeń

Etap	Termin	Zakres prac
Analiza wymagań i dokumentacja wstępna	18.10.2025	Opracowanie założeń, planu pracy, opis projektu
Projekt architektury i modelu danych	29-30.11.2025	Diagramy klas, tabele bazy danych, relacje
Implementacja funkcji rozszerzone o UI	13.12.2025	Moduł dodawania przychodów i wydatków, Kategorie, filtry, widok sum miesięcznych
Testowanie i poprawki	17-18.01.2026	Testy funkcjonalne i niefunkcjonalne
Przygotowanie dokumentacji końcowej i prezentacja	31.01.2026	Podsumowanie, wnioski, podręcznik użytkownika

# Wymagania funkcjonalne

System Zarządzania Budżetem Domowym powinien realizować następujące funkcjonalności

## Obsługa transakcji

Tabela 2. Przedstawiająca metodę funkcjonalną

<b>Nazwa funkcji</b>	<code>addTransaction</code>
<b>Opis</b>	Dodawanie nowych transakcji (przychody i wydatki)
<b>Dane wejściowe</b>	Obiekt nowej transakcji
<b>Dane wyjściowe</b>	Obiekt utworzonej transakcji

Tabela 3. Przedstawiająca metodę funkcjonalną

<b>Nazwa funkcji</b>	<code>editTransaction</code>
<b>Opis</b>	Edytowanie istniejących transakcji.
<b>Dane wejściowe</b>	Obiekt istniejącej aktualizowanej transakcji
<b>Dane wyjściowe</b>	Obiekt istniejącej zaktualizowanej transakcji

Tabela 4. Przedstawiająca metodę funkcjonalną

<b>Nazwa funkcji</b>	<code>deleteTransaction</code>
<b>Opis</b>	Usuwanie transakcji z systemu.
<b>Dane wejściowe</b>	Identyfikator transakcji
<b>Dane wyjściowe</b>	Brak

Tabela 5. Przedstawiająca metodę funkcjonalną

<b>Nazwa funkcji</b>	<code>listTransactions</code>
----------------------	-------------------------------

*Tabela 5. Przedstawiająca metodę funkcjonalną*

<b>Opis</b>	Wyświetlanie listy wszystkich transakcji wraz z podstawowymi informacjami (data, kwota, opis, kategoria, typ).
<b>Dane wejściowe</b>	brak
<b>Dane wyjściowe</b>	Tablica dodanych transakcji

## Obsługa kategorii

*Tabela 6. Przedstawiająca metodę funkcjonalną*

<b>Nazwa funkcji</b>	<code>addCategory</code>
<b>Opis</b>	Dodawanie własnych kategorii budżetowych.
<b>Dane wejściowe</b>	Obiekt nowej kategorii
<b>Dane wyjściowe</b>	Obiekt utworzonej kategorii

*Tabela 7. Przedstawiająca metodę funkcjonalną*

<b>Nazwa funkcji</b>	<code>editCategory</code>
<b>Opis</b>	Edytowanie nazw istniejących kategorii.
<b>Dane wejściowe</b>	Obiekt istniejącej aktualizowanej transakcji
<b>Dane wyjściowe</b>	Obiekt istniejącej zaktualizowanej transakcji

*Tabela 8. Przedstawiająca metodę funkcjonalną*

<b>Nazwa funkcji</b>	<code>deleteCategory</code>
<b>Opis</b>	Usuwanie kategorii (jeśli nie narusza to integralności danych).
<b>Dane wejściowe</b>	Identyfikator kategorii
<b>Dane wyjściowe</b>	brak

Tabela 9. Przedstawiająca metodę funkcjonalną

<b>Nazwa funkcji</b>	assignCategory
<b>Opis</b>	Przypisywanie transakcji do wybranych kategorii. (data, kwota, opis, kategoria, typ).
<b>Dane wejściowe</b>	Identyfikator transakcji i Identyfikator kategorii
<b>Dane wyjściowe</b>	Obiekt Przypisanej Transakcji

## Podstawowa analiza finansowa

Tabela 10. Przedstawiająca metodę funkcjonalną

<b>Nazwa funkcji</b>	sumExpenses, sumIncomes
<b>Opis</b>	Obliczanie sum przychodów i wydatków w wybranym okresie.
<b>Dane wejściowe</b>	Data początkowa, Data końcowa
<b>Dane wyjściowe</b>	Zestawienie obliczeń

Tabela 11. Przedstawiająca metodę funkcjonalną

<b>Nazwa funkcji</b>	getBalance
<b>Opis</b>	Obliczanie bilansu (różnicy pomiędzy przychodami a wydatkami).
<b>Dane wejściowe</b>	Data początkowa, Data końcowa
<b>Dane wyjściowe</b>	Zestaw obliczeń

Tabela 12. Przedstawiająca metodę funkcjonalną

<b>Nazwa funkcji</b>	getSummary
<b>Opis</b>	Wyświetlanie zestawienia podsumowującego dla

Tabela 12. Przedstawiająca metodę funkcjonalną

	wybranego zakresu dat.
<b>Dane wejściowe</b>	Data początkowa, Data końcowa
<b>Dane wyjściowe</b>	Zestaw obliczeń

## Filtrowanie danych

Tabela 13. Przedstawiająca metodę funkcjonalną

<b>Nazwa funkcji</b>	<code>filterTransactions</code>
<b>Opis</b>	Filtrowanie transakcji na podstawie danych wejściowych
<b>Dane wejściowe</b>	Data początkowa, Data końcowa, Kategoria, Typ transakcji
<b>Dane wyjściowe</b>	Filtrowana Lista Transakcji

## Przechowywanie danych

- Zapisywanie wszystkich danych w lokalnej bazie SQLite.
- Automatyczne wczytywanie danych podczas uruchamiania aplikacji.
- Trwałe przechowywanie kategorii i transakcji na dysku użytkownika.

## Interfejs użytkownika

- Prosty i intuicyjny interfejs umożliwiający szybkie przeglądanie i dodawanie danych.
- Formularze do wprowadzania transakcji i kategorii.
- Widok zbiorczy prezentujący stan bieżącego budżetu.

# **Wymagania niefunkcjonalne**

## **Wymagania sprzętowe**

System Zarządzania Budżetem Domowym jest aplikacją lekką i nie wymaga specjalistycznego sprzętu. Do poprawnego działania wystarczy standardowy komputer osobisty o następujących parametrach minimalnych:

- Procesor: dwurdzeniowy (np. Intel i3 lub odpowiednik AMD).
- Pamięć RAM: minimum 4 GB.
- Miejsce na dysku: min. 200 MB wolnej przestrzeni (na aplikację, bazę danych oraz pliki pomocnicze).
- Monitor o rozdzielcości co najmniej 1280x720 px.
- Klawiatura i urządzenie wskazujące (mysz lub touchpad).

### **Wariant alternatywny:**

System może również działać na komputerach o wyższej wydajności, jednak nie wpływa to na działanie aplikacji, ponieważ obciążenie sprzętowe jest niewielkie.

## **Wymagania systemowe**

Aplikacja po zbudowaniu będzie działać lokalnie i nie wymaga połączenia z Internetem.

### **Do jej poprawnego uruchomienia wymagane są:**

System operacyjny:

- Windows 10 / 11, lub
- macOS, lub
- Linux (np. Ubuntu, Fedora).

Środowisko uruchomieniowe:

- Node.js (w wersji 22 LTS lub nowszej).
- Wsparcie dla TypeScript (kompilator tsc).

Biblioteki i narzędzia wymagane przez projekt:

- SQLite – lokalna baza danych.
- NPM – zarządzanie pakietami aplikacji.

Dodatkowe oprogramowanie potrzebne podczas tworzenia i testowania projektu (wymagane jest połączenie z internetem):

- Edytor kodu (np. Visual Studio Code).
- Przeglądarka internetowa (Chrome / Firefox / Edge) do obsługi interfejsu webowego.

## **Wymagania organizacyjne**

System przeznaczony jest do użytku przez jednego użytkownika na pojedynczej maszynie.

Organizacja pracy z systemem oraz środowisko użycia zakłada:

**Jednostenowiskowość** – brak możliwości korzystania przez wielu użytkowników jednocześnie.

**Brak integracji sieciowych** – system działa bez połączenia z Internetem; wszelkie dane wprowadzane są ręcznie.

**Warunki poprawnej pracy** – aplikacja wymaga jedynie uruchomionej przeglądarki oraz działającego środowiska Node.js.

**Wydajność** – system nie jest przeznaczony do obsługi dużych zbiorów danych; jest zoptymalizowany pod kątem danych użytkownika indywidualnego.

**Bezpieczeństwo danych** – dane finansowe są przechowywane w lokalnej bazie SQLite, bez wysyłania ich na zewnętrzne serwery.

**Organizacja pracy projektowej** – tworzenie i rozwój aplikacji przebiega zgodnie z modelem kaskadowym; kolejne etapy są realizowane i dokumentowane zgodnie z harmonogramem zajęć.

# Wymagania dotyczące danych

System korzysta z lokalnej bazy danych SQLite, przechowującej informacje o transakcjach, kategoriach oraz podstawowych ustawieniach aplikacji.

## Tabela: Category

Przechowuje listę kategorii, do których użytkownik przypisuje transakcje.

<i>Tabela 14. Przedstawiająca schemat tabeli Category</i>			
Pole	Typ danych	Opis	Uzasadnienie
<b>id</b>	<code>INTEGER PRIMARY KEY AUTOINCREMENT</code>	Unikalny identyfikator kategorii	Umożliwia jednoznaczne powiązanie z transakcjami
<b>name</b>	<code>TEXT NOT NULL</code>	Nazwa kategorii (np. Jedzenie, Transport)	Tekstowy opis danej grupy wydatków/przychodów
<b>type</b>	<code>TEXT CHECK(type IN ('income', 'expense')) NOT NULL</code>	Typ kategorii	Pomaga oddzielić kategorie przychodów od kategorii wydatków

Uwaga: Pole type upraszcza logikę filtrowania i generowania wykresów.

## Tabela: Transaction

Przechowuje wszystkie zapisane przez użytkownika operacje finansowe.

<i>Tabela 15. Przedstawiająca schemat tabeli Transaction</i>			
Pole	Typ danych	Opis	Uzasadnienie

<i>Tabela 15. Przedstawiająca schemat tabeli Transaction</i>			
<b>id</b>	<b>INTEGER PRIMARY KEY AUTOINCREMENT</b>	Unikalny identyfikator transakcji	Niezbędny do zarządzania rekordami
<b>amount</b>	<b>REAL NOT NULL</b>	Kwota transakcji (dodatnia lub ujemna)	REAL pozwala na zapis wartości dziesiętnych
<b>date</b>	<b>TEXT NOT NULL (format YYYY-MM-DD)</b>	Data wykonania transakcji	Przechowywana jako tekst dla prostoty (SQLite nie ma typu DATE)
<b>description</b>	<b>TEXT</b>	Krótki opis transakcji	Ułatwia identyfikację wpisów
<b>category_id</b>	<b>INTEGER NOT NULL</b>	Powiązanie z kategorią	Relacja do tabeli Category
<b>type</b>	<b>TEXT CHECK(type IN ('income', 'expense')) NOT NULL</b>	Typ transakcji	Podstawowa klasyfikacja wpisu
<b>created_at</b>	<b>TEXT NOT NULL</b>	Data dodania transakcji	Umożliwia sortowanie chronologiczne operacji

Relacja: `category_id → Category(id)` (klucz obcy)

## Tabela: Settings

Przechowuje podstawowe ustawienia aplikacji.

<i>Tabela 16. Przedstawiająca tabeli settings</i>			
Pole	Typ danych	Opis	Uzasadnienie
<b>currency</b>	TEXT DEFAULT 'PLN'	Domyślna waluta	Użytkownik może zmienić walutę wyświetlaną

## Relacje między tabelami

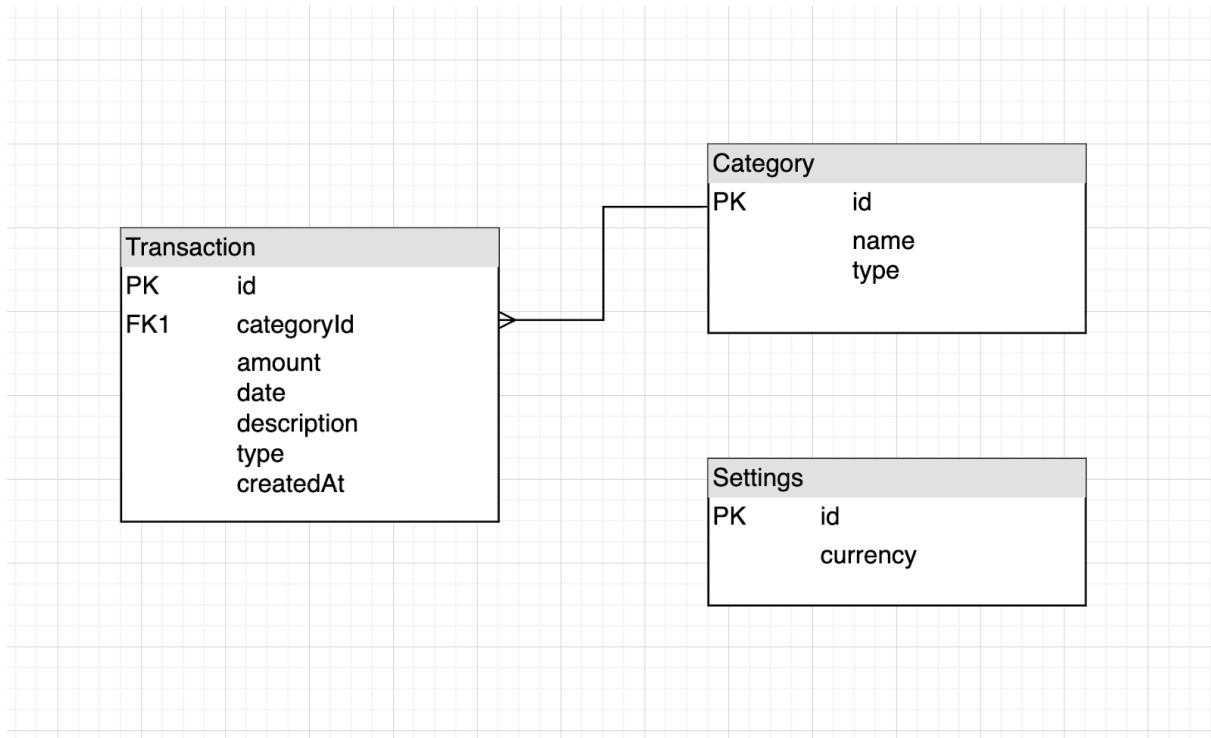
- **Category (1) → Transaction (N)**  
Każda transakcja należy do jednej kategorii.  
Jedna kategoria może mieć wiele transakcji.
- **Settings** nie posiada relacji – zawiera dane globalne.

## Uzasadnienie wyboru struktur danych

- **Prostota** – projekt akademicki nie wymaga rozbudowanego modelu wielotabelowego.
- **Łatwość implementacji** – SQLite w pełni wspiera operacje CRUD dla takich tabel.
- **Czytelność dokumentacji** – model jest łatwy do opisania i przedstawienia na diagramach.
- **Możliwości rozszerzeń** – w przyszłości można dodać tabele: Budżety, Użytkownicy, Limity, Tagowanie transakcji.

## Diagram bazy danych

Schemat prezentuje strukturę bazy danych spełniającej podstawowe potrzeby systemu



Rysunek 5 - Diagram bazy danych Systemu zarządzania budżetem

# Metody pracy, narzędzia i techniki

Opis głównych klas, metod, obiektów, struktur i algorytmów zastosowanych w projekcie (uwzględniając stosowanie gotowych narzędzi obcego autorstwa, w tym open source).

## Technologie

Do uruchomienia kręgosłupa aplikacji skorzystałem z open source frameworka [Next.js](#) napisanego w języku JavaScript.

<https://nextjs.org/docs/pages/api-reference/cli/create-next-app>

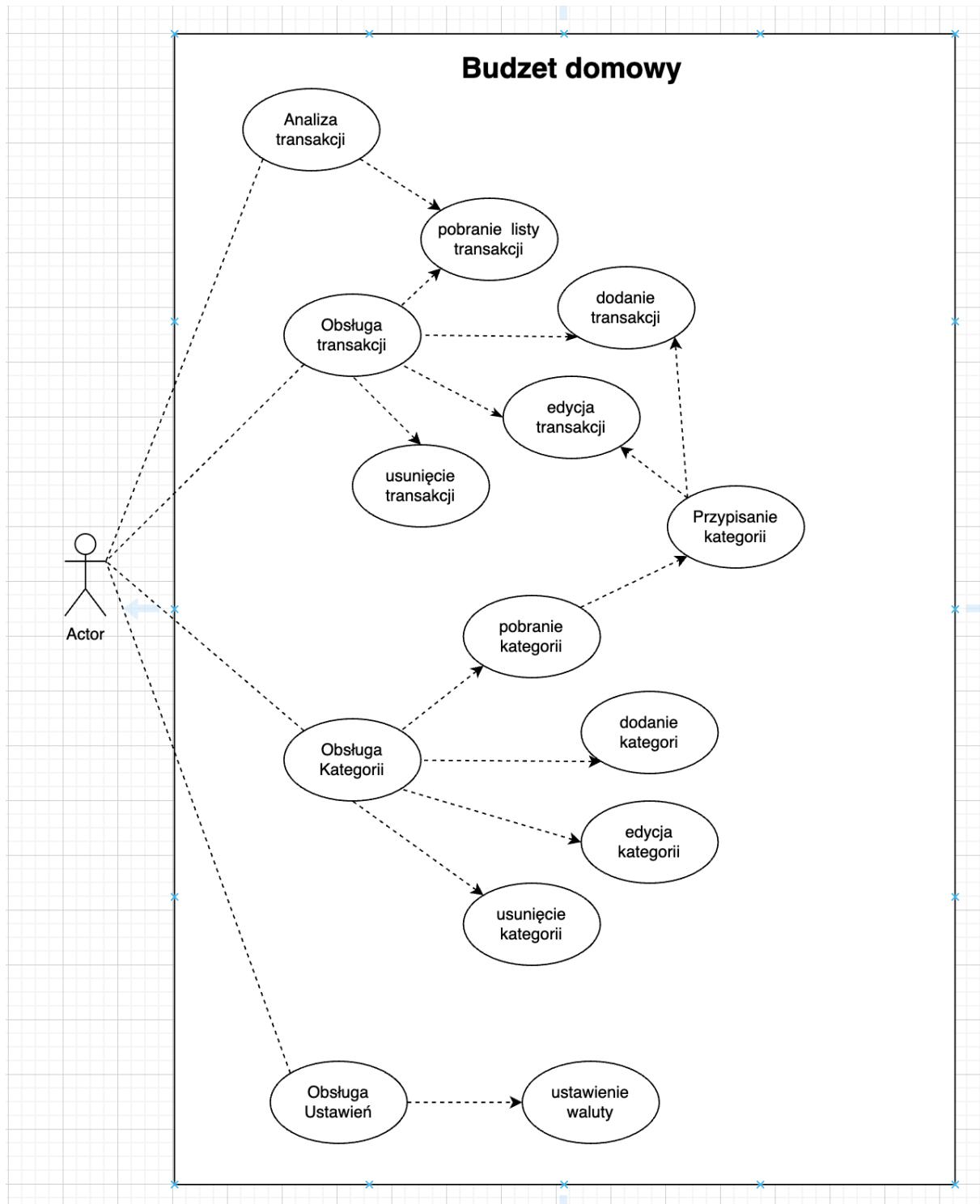
Baza danych obsługiwana będzie za pomocą Drizzle ORM

<https://orm.drizzle.team/docs/get-started-sqlite>

Elementy interfejsu dostarczone są przez bibliotekę Open Source HeroUI

<https://www.heroui.com/docs/guide/introduction>

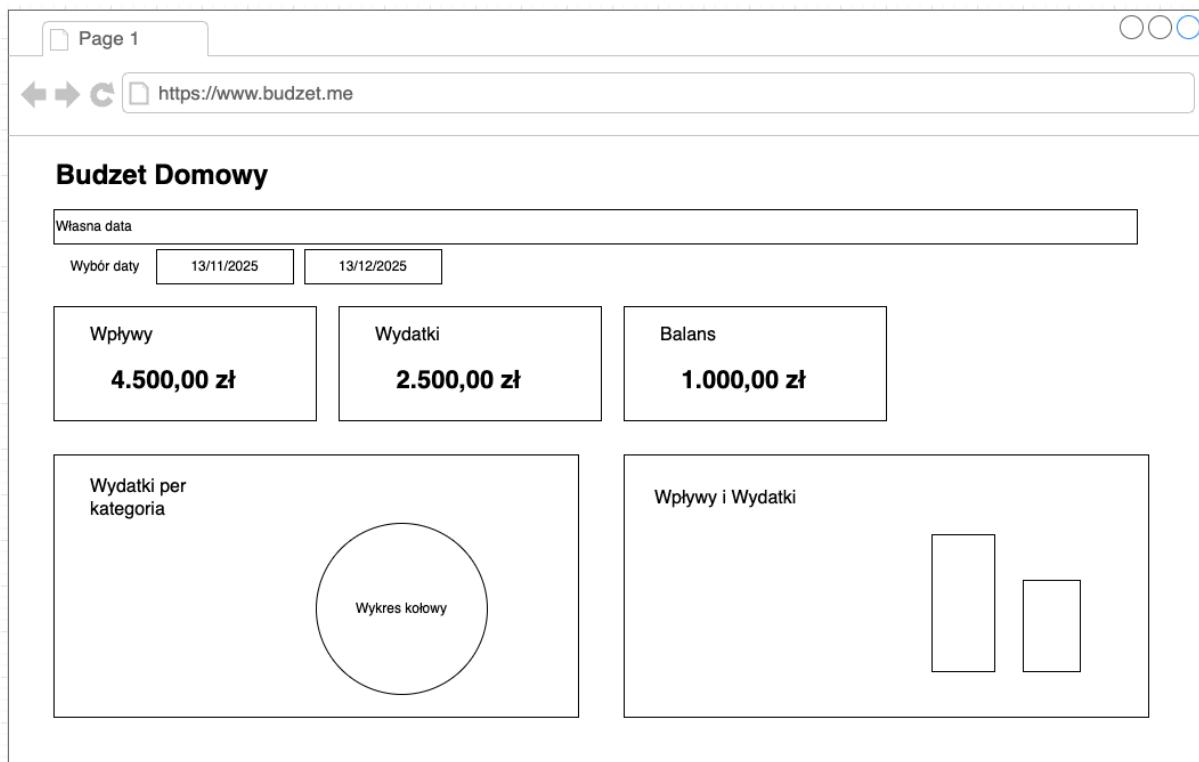
## Diagram Przypadków Użycia



Rysunek 2 - Diagram przypadków użycia

## Szkice projektowe

Strona główna prezentuje podsumowanie wpływów i wydatków dla wybranego okresu. Prezentuje również balans transakcji w danym okresie



Rysunek 4 - Szkic przedstawiający Stronę Główną

Strona Transakcje pozwala na dodawanie, listowanie, edycję i usunięcie transakcji

The wireframe shows a web browser interface with the URL <https://www.budzet.me>. The main content area is titled "Transakcje". On the left, there is a form titled "Dodaj Transakcję" with fields for Kwota (Amount), Typ (Type), Data (Date), Kategoria (Category), and Opis (Opcjonalny) (Description). A "Dodaj Transakcję" button is at the bottom. On the right, there is a "Lista Transakcji" section displaying three transaction entries:

Przychód / Typ	Datum	Kategoria	Opis opcjonalny	Wartość
Przychód	13.12.2025			1.200,00 zł
		Kategoria Opis opcjonalny		
Typ	13.12.2025			1.200,00 zł
		Kategoria Opis opcjonalny		
Typ	13.12.2025			1.200,00 zł
		Kategoria Opis opcjonalny		

Rysunek 5 - Wireframe przedstawiający Stronę Transakcji

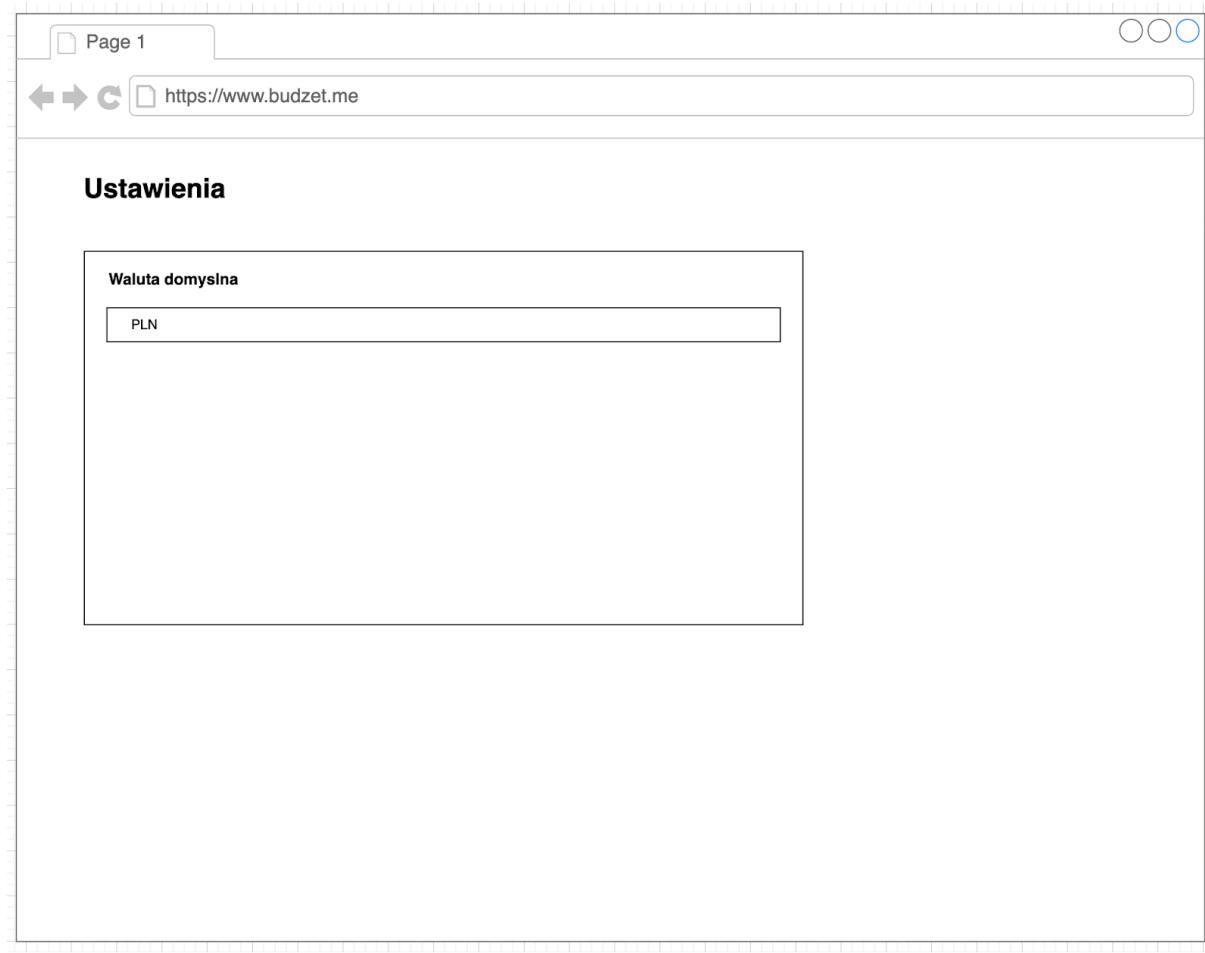
Strona Kategorie pozwala na dodawanie, listowanie, edycję i usunięcie kategorii

The wireframe shows a web browser interface with the URL <https://www.budzet.me>. The main content area is titled "Kategorie". On the left, there is a form titled "Dodaj Kategorię" with a field for Nazwa (Name) and a "Dodaj Kategorię" button. On the right, there is a "Lista Kategorii" section displaying a list of categories with edit and delete options:

Kategoria	Edytuj	Usuń
Jedzenie		
Edukacja		
Inne		
Mieszkanie		
Premia		
Jedzenie		

Rysunek 6 - Wireframe przedstawiający Stronę Kategorii

Strona Ustawienia pozwala na zmianę ustawień globalnych



Rysunek 7 - Wireframe przedstawiający Ustawienia

## Opis Struktur

### balance (Dashboard)

Tabela 18. Przedstawia opis struktury strony głównej

Pole	Typ	Opis
<b>totalIncome</b>	number	Suma wszystkich wpływów w wybranym okresie
<b>totalExpense</b>	number	Suma wszystkich wydatków w wybranym okresie
<b>balance</b>	number	Bilans finansowy (totalIncome - totalExpense)

*Tabela 18. Przedstawia opis struktury strony głównej*

<b>expensesByCategory</b>	<b>CategoryExpense[ ]</b>	Tablica wydatków pogrupowanych według kategorii
<b>monthlyData</b>	<b>MonthlyData[ ]</b>	Dane porównawcze wpływów i wydatków w podziale na miesiące

**Struktura CategoryExpense:**

- categoryId: string | null - Identyfikator kategorii
- categoryName: string - Nazwa kategorii
- total: number - Suma wydatków dla danej kategorii

**Struktura MonthlyData:**

- month: string - Miesiąc w formacie "YYYY-MM"
- income: number - Suma wpływów w danym miesiącu
- expense: number - Suma wydatków w danym miesiącu

**transaction (Transakcja)***Tabela 19. Przedstawia opis struktury transaction*

Pole	Typ	Wymagane	Opis
<b>id</b>	<b>string</b>	Tak	Unikalny identyfikator transakcji (UUID)
<b>amount</b>	<b>number</b>	Tak	Kwota transakcji (musi być > 0)
<b>type</b>	<b>"INCOME"   "EXPENSE"</b>	Tak	Typ transakcji: przychód lub wydatek
<b>date</b>	<b>Date</b>	Tak	Data transakcji (nie może być z przyszłości)
<b>description</b>	<b>string   null</b>	Nie	Opcjonalny opis transakcji (max 500 znaków)
<b>categoryId</b>	<b>string   null</b>	Nie	Identyfikator kategorii (referencja do <a href="#">categories.id</a> )
<b>createdAt</b>	<b>Date</b>	Tak	Data utworzenia rekordu

*Tabela 19. Przedstawia opis struktury transaction*

<b>updatedAt</b>	<b>Date</b>	Tak	Data ostatniej modyfikacji
------------------	-------------	-----	----------------------------

**Relacje:**

- category: Relacja many-to-one z tabelą categories (onDelete: "set null")

**Walidacja:**

- amount: > 0
- date: <= dzisiaj
- description: max 500 znaków
- type: tylko "INCOME" lub "EXPENSE"

## category (Kategoria)

*Tabela 20. Przedstawia opis struktury category*

Pole	Typ	Wymagane	Opis
<b>id</b>	<b>string</b>	Tak	Unikalny identyfikator kategorii (UUID)
<b>name</b>	<b>string</b>	Tak	Nazwa kategorii (unikalna, 1-100 znaków)
<b>createdAt</b>	<b>Date</b>	Tak	Data utworzenia rekordu
<b>updatedAt</b>	<b>Date</b>	Tak	Data ostatniej modyfikacji

**Relacje:**

- transactions: Relacja one-to-many z tabelą transactions

**Walidacja:**

- name: wymagane, 1-100 znaków, unikalne

**Przykładowe kategorie:**

- Jedzenie
- Transport
- Rozrywka
- Zdrowie

- Mieszkanie
- Edukacja
- Premia
- Inne

#### **Uwagi:**

- Przy usunięciu kategorii, powiązane transakcje otrzymują `categoryId = null`
- Kategorie są współdzielone między przychodami i wydatkami

### **settings (Ustawienia)**

*Tabela 21. Przedstawia opis struktury settings*

Pole	Typ	Wymagane	Wartość domyślna	Opis
<b>id</b>	<code>string</code>	Tak	"singleton"	Identyfikator (zawsze "singleton" - tylko jeden rekord)
<b>currency</b>	<code>"PLN"   "EUR"   "USD"</code>	Tak	"PLN"	Waluta wyświetlnia

#### **Typ Singleton:**

- Tabela zawiera dokładnie jeden rekord z `id = "singleton"`
- Służy do przechowywania globalnych ustawień aplikacji

#### **Dostępne waluty:**

- PLN - Polski złoty
- EUR - Euro
- USD - Dolar amerykański

#### **Formatowanie:**

- Waluta wpływa na formatowanie wszystkich kwot w aplikacji
- Wykorzystuje `Intl.NumberFormat` z locale "pl-PL"
- Wyświetla zawsze 2 miejsca po przecinku

## Spis Metod

### Warstwa serwisowa - `services/transactions.ts`

Tabela 22. Metody obsługi transakcji (Services)

Metoda	Parametry	Zwraca	Opis
createTransaction	<code>input: Omit&lt;NewTransaction, "id"   "createdAt"   "updatedAt"&gt;</code>	<code>Promise&lt; Transaction&gt;</code>	Tworzy nową transakcję w bazie danych. Automatycznie generuje ID (UUID) i ustawia timestamp.
updateTransaction	<code>id: string input: Partial&lt;Omit&lt;NewTransaction, ...&gt;&gt;</code>	<code>Promise&lt; Transaction&gt;</code>	Aktualizuje istniejącą transakcję. Automatycznie aktualizuje pole <code>updatedAt</code> .
deleteTransaction	<code>id: string</code>	<code>Promise&lt; void&gt;</code>	Usuwa transakcję z bazy danych.
getTransaction	<code>id: string</code>	<code>Promise&lt; Transaction   undefined&gt;</code>	Pobiera pojedynczą transakcję po ID. Zwraca <code>undefined</code> jeśli nie znaleziono.
getTransactions	<code>filters?: TransactionFilters</code>	<code>Promise&lt; Array&lt;Transaction &amp; { category }&gt;&gt;</code>	Pobiera listę transakcji z opcjonalnym filtrowaniem. Sortuje po dacie malejąco. Zawiera zagnieżdżone dane kategorii.
getSummary	<code>filters?: TransactionFilters</code>	<code>Promise&lt; SummaryData&gt;</code>	Oblicza podsumowanie finansowe: suma wpływów, wydatków i bilans dla wybranego okresu.

*Tabela 22. Metody obsługi transakcji (Services)*

<code>getExpensesByCategory</code>	<code>filters?: TransactionFilters</code>	<code>Promise&lt;CategoryExpense[]&gt;</code>	Grupuje wydatki według kategorii. Zwraca sumę dla każdej kategorii.
<code>getMonthlyComparison</code>	<code>filters?: TransactionFilters</code>	<code>Promise&lt;MonthlyData[]&gt;</code>	Porównuje wpływy i wydatki w podziale na miesiące. Sortuje chronologicznie.

**Warstwa akcji serwerowych - app/transactions/actions.ts***Tabela 23. Metody obsługi transakcji (Server Actions)*

Metoda	Parametry	Zwraca	Opis
<code>addTransaction</code>	<code>input: Omit&lt;NewTransaction, "id"   "createdAt"   "updatedAt"&gt;</code>	<code>Promise&lt;ActionResult&gt;</code>	Dodaje nową transakcję z walidacją. Odświeża cache stron / i /transactions.
<code>editTransaction</code>	<code>id: string input: Partial&lt;Omit&lt;NewTransaction, ...&gt;&gt;</code>	<code>Promise&lt;ActionResult&gt;</code>	Edytuje transakcję z walidacją. Odświeża cache.
<code>removeTransaction</code>	<code>id: string</code>	<code>Promise&lt;ActionResult&gt;</code>	Usuwa transakcję. Odświeża cache.
<code>validateTransaction</code>	<code>input: { amount?, date?, description? }</code>	<code>string   null</code>	(Wewnętrzna) Waliduje dane transakcji. Zwraca błąd lub null.

## Warstwa serwisowa - `services/categories.ts`

<i>Tabela 24. Metody obsługi kategorii (Services)</i>			
<b>Metoda</b>	<b>Parametry</b>	<b>Zwraca</b>	<b>Opis</b>
getCategories	brak	Promise<Category[]>	Pobiera wszystkie kategorie posortowane alfabetycznie po nazwie.
getCategory	id: string	Promise<Category   undefined>	Pobiera pojedynczą kategorię po ID.
createCategory	input: Omit<NewCategory, "id"   "createdAt"   "updatedAt">	Promise<Category>	Tworzy nową kategorię. Automatycznie generuje UUID.
updateCategory	id: string input: Partial<Omit<NewCategory, ...>>	Promise<Category>	Aktualizuje kategorię. Aktualizuje updatedAt.
deleteCategory	id: string	Promise<void>	Usuwa kategorię. Transakcje z tą kategorią otrzymają categoryId = null.

## Warstwa akcji serwerowych - `app/categories/actions.ts`

<i>Tabela 25. Metody obsługi kategorii (Server Actions)</i>			
<b>Metoda</b>	<b>Parametry</b>	<b>Zwraca</b>	<b>Opis</b>
addCategory	input: Omit<NewCategory, "id"   "createdAt"   "updatedAt">	Promise<ActionResult>	Dodaje kategorię z walidacją nazwy. Odświeża cache /categories.
editCategory	id: string input: Partial<Omit<NewCategory, ...>>	Promise<ActionResult>	Edytuje kategorię. Odświeża cache /categories, /transactions, /.

Tabela 25. Metody obsługi kategorii (Server Actions)

removeCategory	<code>id: string</code>	<code>Promise&lt;ActionResult&gt;</code>	Usuwa kategorię. Odświeża cache wszystkich powiązanych stron.
----------------	-------------------------	--	---

### Warstwa serwisowa - `services/settings.ts`

Tabela 26. Metody obsługi ustawień (Services)

Metoda	Parametry	Zwraca	Opis
getSettings	<code>brak</code>	<code>Promise&lt;Settings&gt;</code>	Pobiera ustawienia aplikacji. Tworzy domyślne (PLN) jeśli nie istnieją. Pattern: Singleton.
updateSettings	<code>currency: Currency</code>	<code>Promise&lt;Settings&gt;</code>	Aktualizuje walutę wyświetlania. Zawsze operuje na rekordzie id="singleton".

### Warstwa akcji serwerowych - `app/settings/actions.ts`

Tabela 27. Metody obsługi ustawień (Server Actions)

Metoda	Parametry	Zwraca	Opis
changeCurrency	<code>currency: Currency</code>	<code>Promise&lt;ActionResult&gt;</code>	Zmienia walutę. Odświeża cache /settings, /transactions, /.

### Warstwa akcji serwerowych - `app/actions.ts`

Tabela 28. Metody pomocnicze dashboardu

Metoda	Parametry	Zwraca	Opis
--------	-----------	--------	------

*Tabela 28. Metody pomocnicze dashboardu*

<code>getDashboardData</code>	<code>filters?: TransactionFilters</code>	<code>Promise&lt;DashboardData&gt;</code>	Pobiera wszystkie dane dla dashboardu (summary, expensesByCategory, monthlyData) w jednym zapytaniu. Optymalizacja: Promise.all.
-------------------------------	---	---	--

**Funkcje pomocnicze - lib/format.ts***Tabela 29. Metody formatowania*

Metoda	Parametry	Zwraca	Opis
<code>formatCurrency</code>	<code>amount: number</code> <code>currency: Currency = "PLN"</code>	<code>string</code>	Formatuje kwotę według wybranej waluty. Używa Intl.NumberFormat z locale "pl-PL".
<code>formatDate</code>	<code>date: Date</code>	<code>string</code>	Formatuje datę w formacie DD.MM.YYYY (polski).
<code>formatDateInput</code>	<code>date: Date</code>	<code>string</code>	Formatuje datę dla inputu HTML (YYYY-MM-DD).

## Funkcje komponentu - `components/shared>SelectPeriod.tsx`

Tabela 30. Metody pomocnicze `SelectPeriod`

Metoda	Parametry	Zwraca	Opis
<code>getDateRangeFromPreset</code>	<code>preset: PeriodPreset</code> <code>customFrom?: string</code> <code>customTo?: string</code>	<code>{     dateFrom: Date,     dateTo: Date }</code>	Konwertuje preset lub custom range na obiekt z datami początku i końca.

## Wykaz Algorytmów

### Agregacja danych finansowych (Summary)

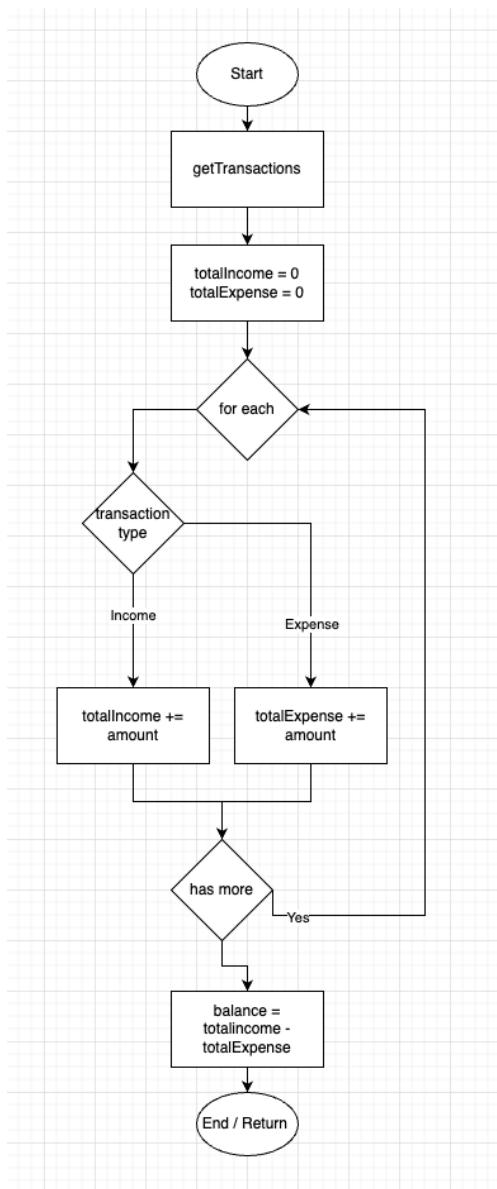
#### Opis

Algorytm oblicza sumę wpływów, wydatków i bilans finansowy poprzez filtrowanie i redukcję tablicy transakcji. Wykorzystuje metodę reduce do akumulacji wartości.

#### Lokalizacja

`src/services/transactions.ts` - funkcja `getSummary()`

## Wizualizacja



Rysunek 8 - Algorytm sumowania balansu

## Pseudokod

```
FUNKCJA getSummary(filters):  
    transactions = getTransactions(filters)  
    totalIncome = 0  
    totalExpense = 0  
    DLA KAŻDEJ transakcji W transactions:  
        JEŚLI transakcja.type == "INCOME":  
            totalIncome = totalIncome + transakcja.amount  
        W PRZECIWNYM RAZIE:
```

```
    totalExpense = totalExpense + transakcja.amount  
    balance = totalIncome - totalExpense  
    ZWRÓĆ {  
        totalIncome,  
        totalExpense,  
        balance  
    }
```

## Złożoność

Czasowa: O(n) gdzie n = liczba transakcji

Pamięciowa: O(1) - stała ilość zmiennych

## Optymalizacja

Możliwa optymalizacja przez agregację SQL (SUM z GROUP BY)

Obecna implementacja: O(n) w pamięci (akceptowalne dla małych zbiorów)

## Przykład

```
// Dla transakcji: [100, -50, 200, -30]  
// Wynik: { totalIncome: 300, totalExpense: 80, balance: 220 }
```

# Użytkowanie

## Proces uruchomieniowy

### Wymagania środowiskowe

- Node.js 20 lub nowszy
- pnpm (lub npm/yarn)

### Instalacja

W konsoli CMD lub Bash wykonaj następujące komendy:

1. Instalacja programu

```
pnpm install
```

2. Utworzenie pliku konfiguracyjnego

```
echo "DB_FILE_NAME=file:./local.db" > .env.local
```

3. Inicjalizacja bazy danych

```
pnpm drizzle-kit push
```

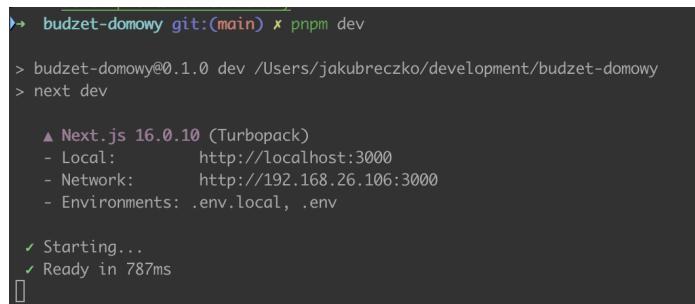
4. Załadowanie danych testowych

```
pnpm seed
```

### Uruchomienie serwera deweloperskiego

W konsoli CMD lub Bash wykonaj następującą komendę:

```
pnpm dev
```



```
▶ budzet-domowy git:(main) ✘ pnpm dev
> budzet-domowy@0.1.0 dev /Users/jakubreczko/development/budzet-domowy
> next dev

  ▲ Next.js 16.0.10 (Turbopack)
  - Local:      http://localhost:3000
  - Network:    http://192.168.26.106:3000
  - Environments: .env.local, .env

  ✓ Starting...
  ✓ Ready in 787ms
  └
```

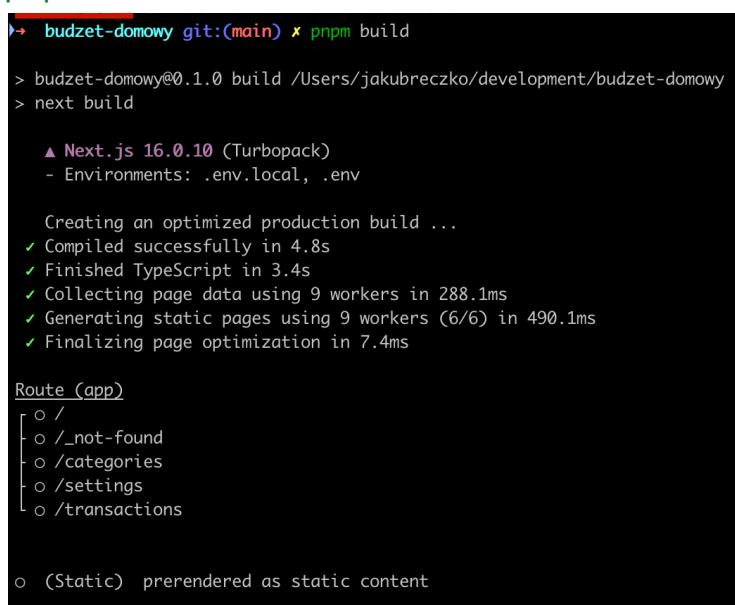
Rysunek 9 - Uruchomienie komendy `dev`

Aplikacja będzie dostępna pod adresem <http://localhost:3000>

### Uruchomienie produkcyjne

1. Budowa aplikacji

```
pnpm build
```



```
▶ budzet-domowy git:(main) ✘ pnpm build
> budzet-domowy@0.1.0 build /Users/jakubreczko/development/budzet-domowy
> next build

  ▲ Next.js 16.0.10 (Turbopack)
  - Environments: .env.local, .env

  Creating an optimized production build ...
  ✓ Compiled successfully in 4.8s
  ✓ Finished TypeScript in 3.4s
  ✓ Collecting page data using 9 workers in 288.1ms
  ✓ Generating static pages using 9 workers (6/6) in 490.1ms
  ✓ Finalizing page optimization in 7.4ms

  Route (app)
  └
    o /
    └ _not-found
    o /categories
    └ /settings
      o /transactions

    o (Static) prerendered as static content
```

Rysunek 10 - Uruchomienie komendy `build`

2. Uruchomienie serwera produkcyjnego

```
pnpm start
```

```
↳ budżet-domowy git:(main) ✘ pnpm start

> budżet-domowy@0.1.0 start /Users/jakubreczko/development/budżet-domowy
> next start

  ▲ Next.js 16.0.10
    - Local:      http://localhost:3000
    - Network:    http://192.168.26.106:3000

    ✓ Starting...
    ✓ Ready in 453ms
```

Rysunek 11 - Uruchomienie komendy `start`

## Dostępne strony

- `/` - Dashboard z podsumowaniem i wykresami
- `/transactions` - Zarządzanie transakcjami
- `/categories` - Zarządzanie kategoriami
- `/settings` - Ustawienia waluty

## Proces testowy

### Testy jednostkowe i integracyjne

- Uruchomienie testów w trybie watch

```
pnpm test
  ✓ src/app/transactions/__tests__/actions.test.ts (17 tests) 0ms

  Test Files 14 passed (14)
  Tests 128 passed (128)
  Start at 09:31:32
  Duration 4.95s (transform 588ms, setup 2.22s, import 8.66s, tests 5.59s, environment 5.93s)

  PASS Waiting for file changes...
  press h to show help, press q to quit
```

Rysunek 12 - Uruchomienie komendy `test`

- Uruchomienie testów jednorazowo

```
pnpm test:run
```

```
Test Files 14 passed (14)
  Tests 128 passed (128)
  Start at 09:39:16
  Duration 5.60s (transform 916ms, setup 3.71s, import 13.34s, tests 6.11s, environment 9.31s)
```

Rysunek 13 - Uruchomienie komendy `test:run`

## Struktura testów

Testy w projekcie podzielone są proporcjonalnie:

- **70%** - Testy jednostkowe (funkcje, serwisy, utility)

- 20% - Testy integracyjne (komponenty, akcje serwerowe)

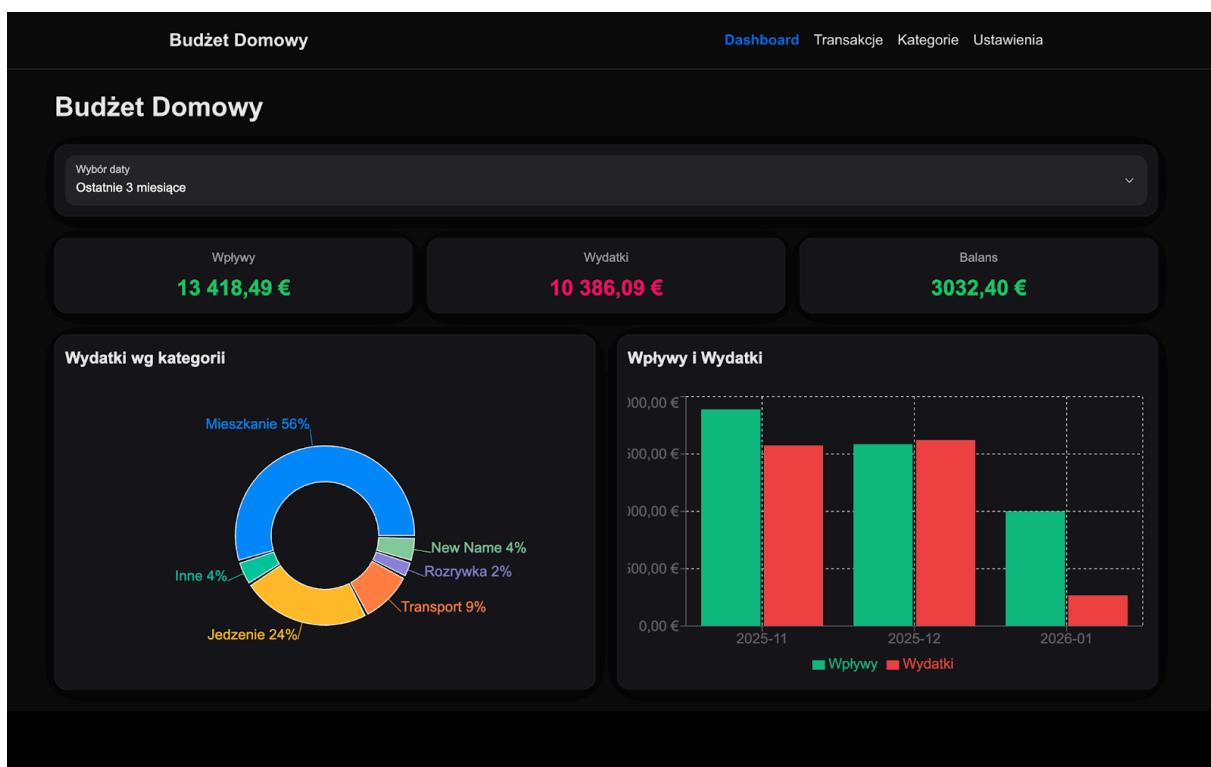
### Lokalizacja testów

- Testy jednostkowe: `src/**/*.test.ts` lub `src/**/_tests_/*.test.ts`
- Testy komponentów: `src/components/**/*.test.tsx`

### Narzędzia testowe

- **Vitest** - framework testowy
- **React Testing Library** - testowanie komponentów React
- **In-memory SQLite** - izolowana baza danych dla testów

### Zrzuty ekranów



Rysunek 14 - Widok strony Dashboard aplikacji

The screenshot shows the 'Transakcje' (Transactions) section of the application. On the left, there's a form for adding a new transaction with fields for amount, type, date, category, and description. On the right, a list of existing transactions is displayed, showing details like date, type, category, amount, and edit/delete buttons.

**Dodaj transakcję**

- Kwota: 0.00
- Typ: Wybierz typ
- Kwota musi być większa od 0
- Data: 18/01/2026
- Kategoria: Wybierz kateg...
- Data nie może być z przyszłości
- Opis: Opcjonalny opis transakcji
- 0/500 znaków

**Dodaj transakcję**

**Lista transakcji (24)**

Typ transakcji	Data	Kwota	Akcje
Przychód	25.12.2025	+4753,35 zł	Edytuj Usuń
Przychód	25.11.2025	+5134,88 zł	Edytuj Usuń
Przychód	25.10.2025	+4594,84 zł	Edytuj Usuń
Przychód	25.09.2025	+4741,20 zł	Edytuj Usuń
Przychód	25.08.2025	+5421,45 zł	Edytuj Usuń

Rysunek 15 - Widok strony Transakcje aplikacji

The screenshot shows the 'Kategorie' (Categories) page of the application. At the top, there is a header bar with the title 'Budżet Domowy' and navigation links: 'Dashboard', 'Transakcje', 'Kategorie' (which is highlighted in blue), and 'Ustawienia'. Below the header, the main content area has a title 'Kategorie'. On the left, there is a form titled 'Dodaj Kategorię' (Add Category) with a single input field labeled 'Nazwa\*' (Name\*) containing 'Nazwa kategorii' (Category name). Below the input field is a note '0/100 znaków' (0/100 characters). A blue button labeled 'Dodaj Kategorię' (Add Category) is at the bottom of the form. To the right, there is a table titled 'Lista Kategorii' (List of Categories) showing a list of categories with their details and actions:

Kategoria	Edytuj	Usuń
Test Category	Edytuj	Usuń
Inne	Edytuj	Usuń
Jedzenie	Edytuj	Usuń
Mieszkanie	Edytuj	Usuń
New Name	Edytuj	Usuń
Premia	Edytuj	Usuń
Rozrywka	Edytuj	Usuń
Salary	Edytuj	Usuń
Test Category	Edytuj	Usuń
Transport	Edytuj	Usuń
Updated Name	Edytuj	Usuń

Rysunek 16 - Widok strony Kategorie aplikacji

The screenshot shows the 'Ustawienia' (Settings) page of the application. At the top, there is a header bar with the title 'Budżet Domowy' and navigation links: 'Dashboard', 'Transakcje', 'Kategorie', and 'Ustawienia' (which is highlighted in blue). Below the header, the main content area has a title 'Ustawienia'. On the left, there is a form titled 'Waluta domyślna' (Default Currency) with a dropdown menu showing 'Waluta domyślna' and 'EUR - Euro'. Below the dropdown is a note: 'Wybrana waluta będzie używana do formatowania wszystkich kwot w aplikacji' (The selected currency will be used to format all amounts in the application). A blue button labeled 'Zapisz' (Save) is at the bottom of the form.

Rysunek 17 - Widok strony Ustawienia aplikacji

## Podsumowanie

Zakres założeń wstępnych został w pełni zaimplementowany i przetestowany manualnie na zajęciach. Wykryto kilka błędów w trakcie testów, które zostały poprawione w trakcie zajęć.

Zidentyfikowano obszary do usprawnień takie jak paginacja listy transakcji, lub zabezpieczenie przed przypadkowym usunięciem kategorii. Wskazane usprawnienia mogą zostać dodane w kolejnych iteracjach na etapie rozwoju produktu.

Poza podstawowym zakresem założeń i ograniczeń projektu zaimplementowano dodatkowe funkcjonalności, które rozszerzają możliwości aplikacji. Wśród najważniejszych należy wymienić kompleksowy system testów. Obejmują testy jednostkowe, integracyjne. Dodano filtrowanie transakcji po okresie na dashboardzie, z niestandardowym zakresem dat oraz przygotowanymi presetami.

Obecny stan aplikacji jest kompletny i gotowy do dalszego rozwoju. Użyte narzędzia i wzorce umożliwiają rozszerzenie programu o kolejne usprawnienia.

# Bibliografia

**Vercel Inc.** Next.js Documentation. [Next.js](#)<sup>8</sup> by Vercel.

Oficjalna dokumentacja frameworka Next.js 16 wykorzystanego jako podstawa aplikacji. Zawiera informacje o App Router, Server Components, Server Actions oraz optymalizacji wydajności.

**Meta Platforms Inc.** React Documentation. [React](#)<sup>9</sup>.

Dokumentacja biblioteki React 19, opisująca komponenty, hooks, Server Components oraz najlepsze praktyki w budowaniu interfejsów użytkownika.

**Microsoft Corporation.** TypeScript Documentation. [TypeScript](#)<sup>10</sup>.

Oficjalna dokumentacja języka TypeScript wykorzystanego do zapewnienia bezpieczeństwa typów w całej aplikacji. Zawiera informacje o systemie typów, interfejsach oraz zaawansowanych funkcjach.

**Drizzle Team.** Drizzle ORM Documentation. [Drizzle ORM](#)<sup>11</sup>.

Dokumentacja ORM Drizzle wykorzystanego do zarządzania bazą danych SQLite. Zawiera informacje o schemacie, relacjach, zapytaniach oraz migracjach.

**MDN Web Docs Contributors.** Intl.NumberFormat - JavaScript | MDN. [Mozilla Developer Network](#)<sup>12</sup>.

Dokumentacja API Intl.NumberFormat wykorzystanego do formatowania walut zgodnie z lokalizacją polską. Zawiera informacje o opcjach formatowania oraz obsłudze różnych walut.

---

<sup>8</sup> <https://nextjs.org/docs>

<sup>9</sup> <https://react.dev>

<sup>10</sup> <https://www.typescriptlang.org/docs>

<sup>11</sup> <https://orm.drizzle.team/docs/overview>

<sup>12</sup>

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Intl/NumberFormat](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Intl/NumberFormat)

# Spis Rysunków

**Rysunek 1.** - Diagram bazy danych Systemu zarządzania budżetem

**Rysunek 2.** - Diagram przypadków użycia

**Rysunek 3.** - Diagram klas

**Rysunek 4.** - Szkic przedstawiający Stronę Główną

**Rysunek 5.** - Szkic przedstawiający Stronę Transakcji

**Rysunek 6.** - Szkic przedstawiający Stronę Kategorii

**Rysunek 7.** - Szkic przedstawiający Ustawienia

**Rysunek 8.** - Algorytm Sumowania Balansu

**Rysunek 9.** - Uruchomienie komendy `dev`

**Rysunek 10.** - Uruchomienie komendy `build`

**Rysunek 11.** - Uruchomienie komendy `start`

**Rysunek 12.** - Uruchomienie komendy `test`

**Rysunek 13.** - Uruchomienie komendy `test:run`

**Rysunek 14.** - Widok strony Dashboard aplikacji

**Rysunek 15.** - Widok strony Transakcje aplikacji

**Rysunek 16.** - Widok strony Kategorie aplikacji

**Rysunek 17.** - Widok strony Ustawienia aplikacji

# Spis Tabel

**Tabela 1** - przedstawiająca chronologię wykonania projektu w środowisku do terminu ćwiczeń

**Tabela od 2. do 13.** - Przedstawiająca metodę funkcjonalną

**Tabela 14.** - Przedstawiająca schemat tabeli category

**Tabela 15.** - Przedstawiająca schemat tabeli transaction

**Tabela 17.** - Przedstawiająca schemat tabeli settings

**Tabela 18.** - Przedstawia opis struktury strony głównej

**Tabela 19.** - Przedstawia opis struktury transaction

**Tabela 20.** - Przedstawia opis struktury category

**Tabela 21.** - Przedstawia opis struktury settings

**Tabela 22.** - Metody obsługi transakcji (Services)

**Tabela 23.** - Metody obsługi transakcji (Server Actions)

**Tabela 24.** - Metody obsługi kategorii (Services)

**Tabela 25.** - Metody obsługi kategorii (Server Actions)

**Tabela 26.** - Metody obsługi ustawień (Services)

**Tabela 27.** - Metody obsługi ustawień (Server Actions)

**Tabela 29.** - Metody formatowania

**Tabela 30.** - Metody pomocnicze `SelectPeriod`