



Wyższa Szkoła Technologii
Informatycznych w Katowicach

System Zarządzania Budżetem Domowym

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

Dane wykonującego	
Imię i nazwisko	Jakub Reczko
Nr albumu	09224
Dane prowadzącego	
	mgr Jacek Żywczok
Data oddania	
Data zaliczenia	

Spis treści

Spis treści.....	1
Wprowadzenie do tematyki projektu.....	3
Zamierzony cel projektu.....	4
Cel główny.....	4
Cele szczegółowe.....	4
Wstępne założenia i uwarunkowania.....	5
Założone ograniczenia i możliwość ewaluacji projektu.....	6
Ograniczenia techniczne.....	6
Ograniczenia organizacyjne i czasowe.....	7
Chronologiczny plan pracy.....	8
Wymagania funkcjonalne.....	9
Obsługa transakcji.....	9
Obsługa kategorii.....	10
Podstawowa analiza finansowa.....	11
Filtrowanie danych.....	12
Przechowywanie danych.....	12
Interfejs użytkownika.....	12
Wymagania нефункционалне.....	13
Wymagania sprzętowe.....	13
Wymagania systemowe.....	13
Wymagania organizacyjne.....	14
Wymagania dotyczące danych.....	15
Tabela: Category.....	15
Tabela: Transaction.....	15
Tabela: Settings.....	16
Relacje między tabelami.....	17
Uzasadnienie wyboru struktur danych.....	17
Diagram bazy danych.....	18
Metody pracy, narzędzia i techniki.....	19
Technologie.....	19
Diagram Przypadków Użycia.....	20
Diagram Klas.....	21
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
Wykaz Algorytmów.....	32
Agregacja danych finansowych (Summary).....	32
Użytkownie.....	34
Proces uruchomieniowy.....	34
Proces testowy.....	35
Zrzuty ekranów.....	36
Podsumowanie.....	38
Bibliografia.....	39
Spis Rysunków.....	40
Spis Tabel.....	41

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 kształtowanie 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 określeniem 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 SQLite¹, co eliminuje ryzyko ich udostępnienia podmiotom zewnętrznym.

¹ <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](https://nextjs.org/)² (JavaScript/TypeScript) + [Drizzle ORM](https://orm.drizzle.team/)³ + [HeroUI](https://www.heroui.com/)⁴
2. Aplikacja będzie działać lokalnie, z wykorzystaniem lekkiej bazy danych SQLite⁵.
3. System zostanie zaprojektowany zgodnie z zasadami programowania obiektowego.
4. Zastosowany zostanie prosty model projektowy – **model kaskadowy**⁶ (Waterfall).
5. Diagramy zostaną wykonane za pomocą programu [Draw.io](https://draw.io)⁷.
6. Dokumentacja zostanie wyeksportowana do formatu PDF.
7. Projekt będzie rozwijany etapowo, z raportowaniem postępów na każdym zajęciach.

² <https://nextjs.org/>

³ <https://orm.drizzle.team/>

⁴ <https://www.heroui.com/>

⁵ <https://sqlite.org/>

⁶ https://pl.wikipedia.org/wiki/Model_kaskadowy

⁷ <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 jednoczesnym 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.

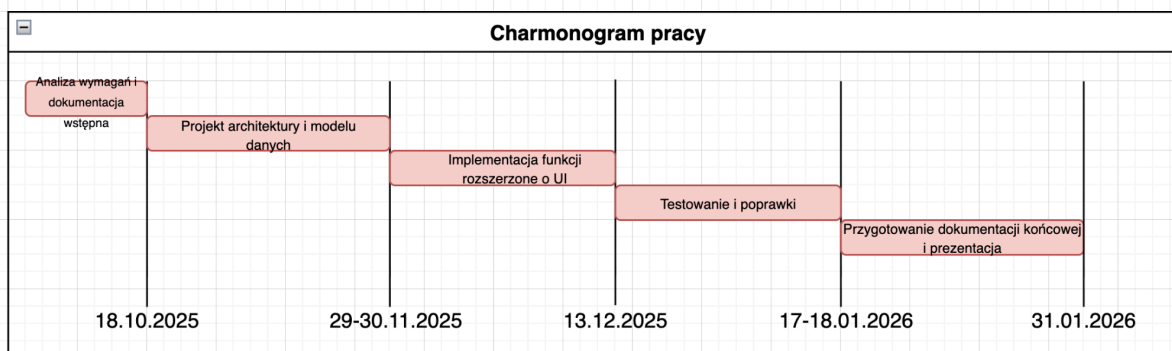


Tabela 1. przedstawiająca chronologię wykonywania projektu w stosunku do terminów ćwiczeń

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 нефункционалне
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

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

Tabela 3. Przedstawiająca metodę funkcjonalną

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

Tabela 4. Przedstawiająca metodę funkcjonalną

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

Tabela 5. Przedstawiająca metodę funkcjonalną

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

Tabela 5. Przedstawiająca metodę funkcjonalną

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

Obsługa kategorii

Tabela 6. Przedstawiająca metodę funkcjonalną

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

Tabela 7. Przedstawiająca metodę funkcjonalną

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

Tabela 8. Przedstawiająca metodę funkcjonalną

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

Tabela 9. Przedstawiająca metodę funkcjonalną

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

Podstawowa analiza finansowa

Tabela 10. Przedstawiająca metodę funkcjonalną

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

Tabela 11. Przedstawiająca metodę funkcjonalną

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

Tabela 12. Przedstawiająca metodę funkcjonalną

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

Tabela 12. Przedstawiająca metodę funkcjonalną

	wybranego zakresu dat.
Dane wejściowe	Data początkowa, Data końcowa
Dane wyjściowe	Zestaw obliczeń

Filtrowanie danych

Tabela 13. Przedstawiająca metodę funkcjonalną

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

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 rozdzielczości 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:

Jednostanowiskowość – 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.

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

Tabela 15. Przedstawiająca schemat tabeli Transaction			
Pole	Typ danych	Opis	Uzasadnienie

Tabela 15. Przedstawiająca schemat tabeli Transaction

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

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

Tabela: Settings

Przechowuje podstawowe ustawienia aplikacji.

Tabela 16. Przedstawiająca tabeli settings

Pole	Typ danych	Opis	Uzasadnienie
currency	TEXT DEFAULT 'PLN'	Domyślna waluta	Użytkownik może zmienić walutę wyświetlania

Relacje między tabelami

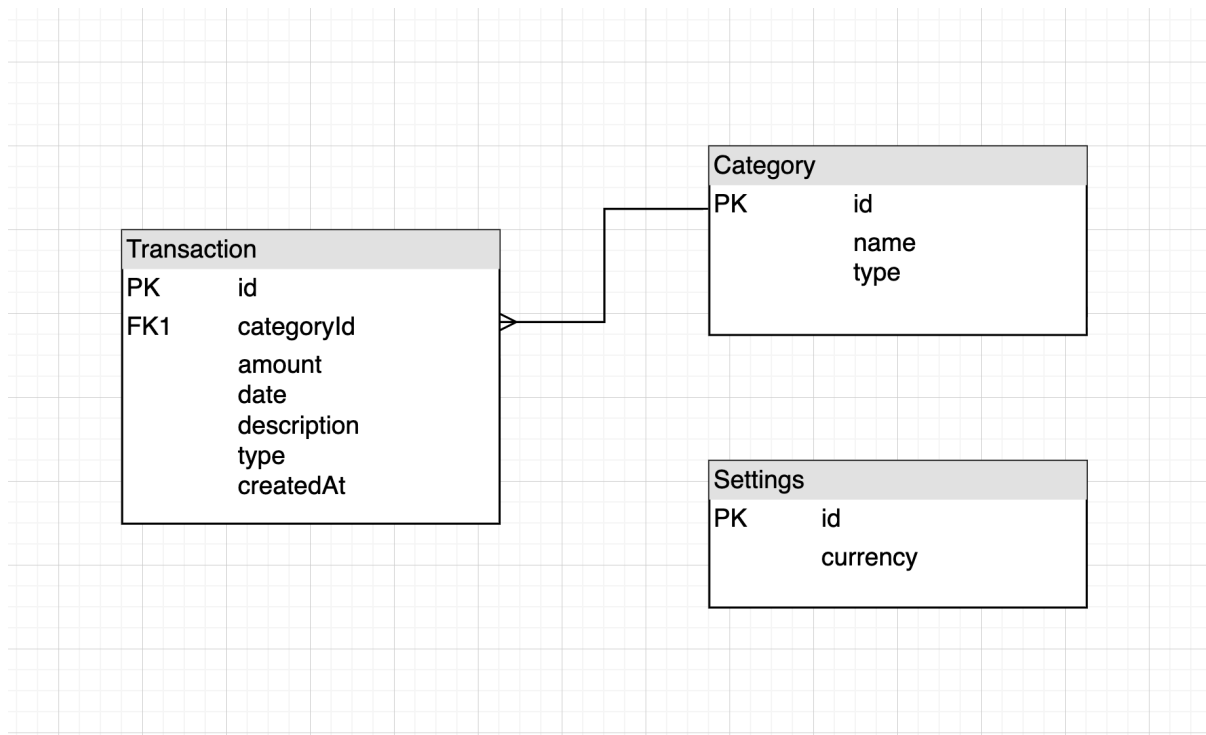
- **Category (1) → Transaction (N)**
Każda transakcja należy do jednej kategorii.
Jedna kategoria może mieć wiele transakcji.
- **Settings** nie posiada relacji — zawiera dane globalne.

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 opisanego 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](https://nextjs.org/docs/pages/api-reference/cli/create-next-app) napisanego w języku JavaScript.

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

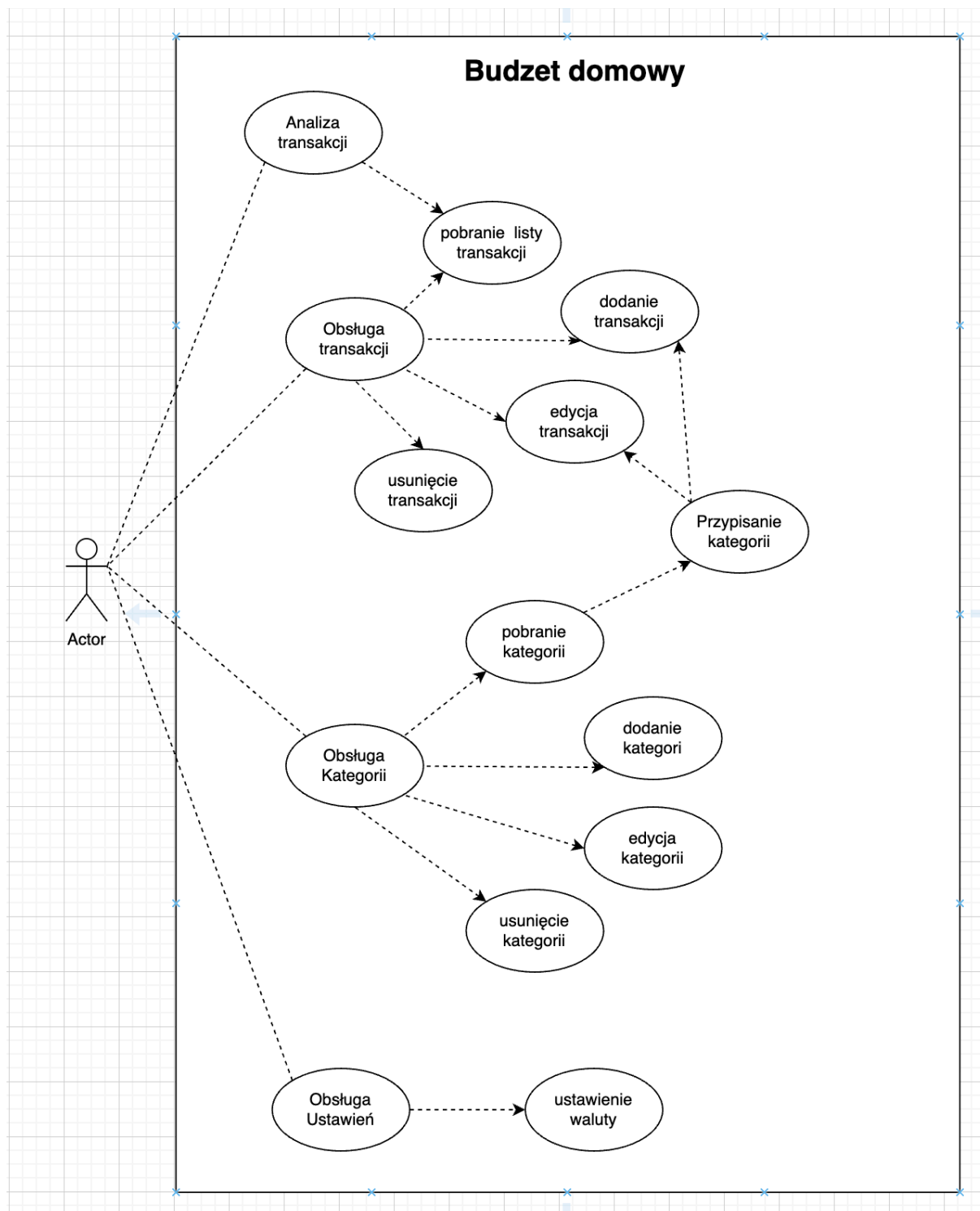
Baza danych obsługiwana będzie za pomocą Drizzle ORM

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

Elementy interface'u dostarczone są przez bibliotekę Open Source HeroUI

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

Diagram Przypadków Użycia



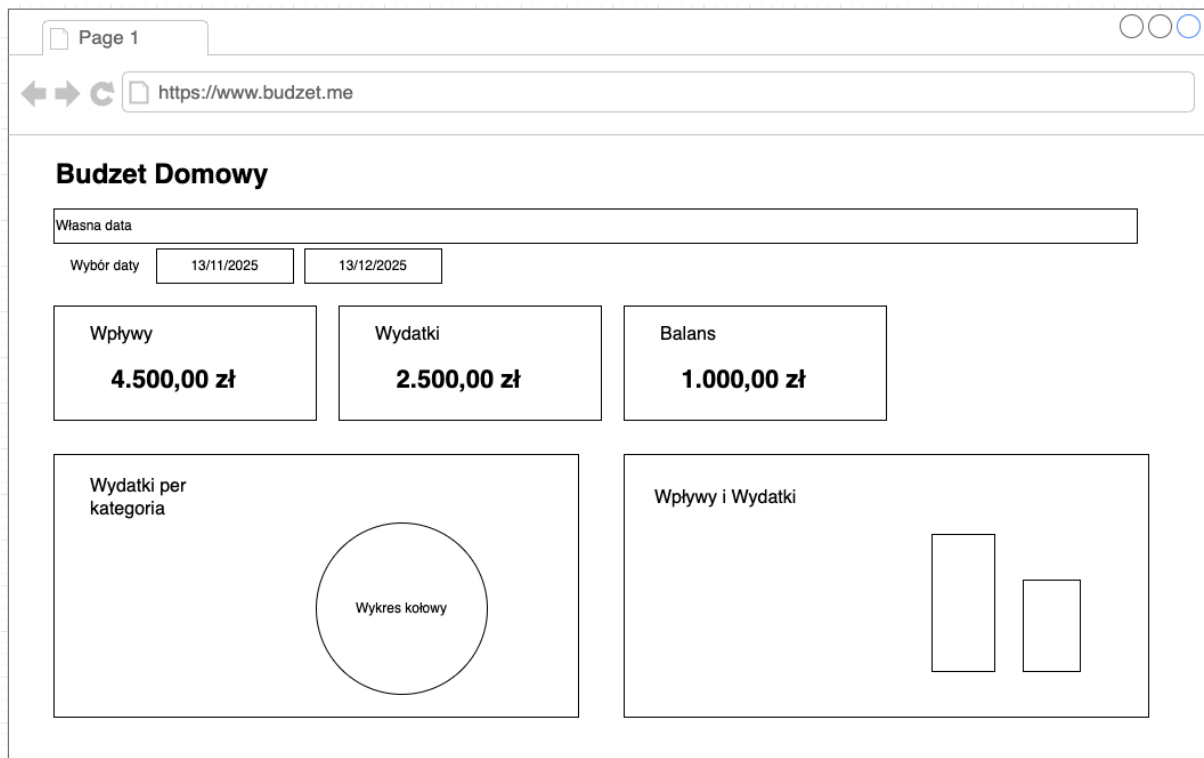
Rysunek 2 - Diagram przypadków użycia

Diagram Klas

Rysunek 3 - Diagram klas

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 window with the address `https://www.budzet.me`. The page title is "Transakcje". On the left, there is a "Dodaj Transakcje" form with fields for "Kwota" (Amount), "Typ" (Type), "Data" (Date), "Kategoria" (Category), and "Opis (Opcjonalny)" (Optional Description). A "Dodaj Transakcje" button is at the bottom. On the right, there is a "Lista Transakcji" (List of Transactions) section. It displays three transaction entries, each with a "Typ" (Type), "Data" (Date), "Kategoria" (Category), "Opis opcjonalny" (Optional Description), and a value of "1.200,00 zł".

Transakcje				
Dodaj Transakcje				
Kwota	Typ			
<input type="text"/>	<input type="text"/>			
Data	Kategoria			
<input type="text"/>	<input type="text"/>			
Opis (Opcjonalny)				
<input type="text"/>				
<input type="button" value="Dodaj Transakcje"/>				

Lista Transakcji				
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

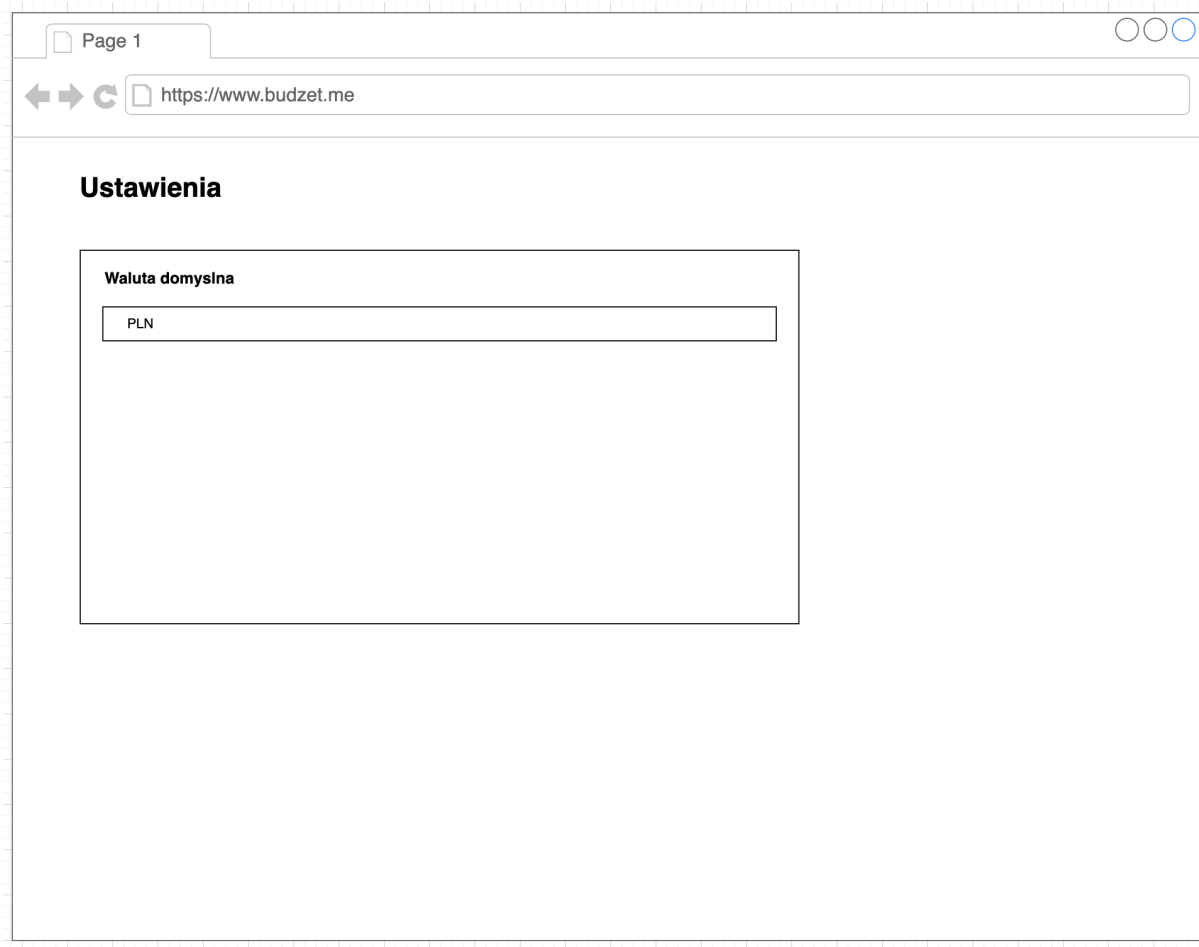
The wireframe shows a web browser window with the address `https://www.budzet.me`. The page title is "Kategorie". On the left, there is a "Dodaj Kategorie" form with a "Nazwa" (Name) field and a "Dodaj Kategorię" button. On the right, there is a "Lista Kategorii" (List of Categories) section. It displays a list of categories with "Edytuj" (Edit) and "Usuń" (Delete) buttons for each.

Kategorie	
Dodaj Kategorie	
Nazwa	
<input type="text"/>	
<input type="button" value="Dodaj Kategorię"/>	

Lista Kategorii	
Jedzenie	Edytuj Usun
Edukacja	Edytuj Usun
Inne	Edytuj Usun
Mieszkanie	Edytuj Usun
Premia	Edytuj Usun
Jedzenie	Edytuj Usun

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
totalIncome	number	Suma wszystkich wpływów w wybranym okresie
totalExpense	number	Suma wszystkich wydatków w wybranym okresie
balance	number	Bilans finansowy (totalIncome - totalExpense)

Tabela 18. Przedstawia opis struktury strony głównej

expensesByCategory	<code>CategoryExpense[]</code>	Tablica wydatków pogrupowanych według kategorii
monthlyData	<code>MonthlyData[]</code>	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
id	<code>string</code>	Tak	Unikalny identyfikator transakcji (UUID)
amount	<code>number</code>	Tak	Kwota transakcji (musi być > 0)
type	<code>"INCOME" "EXPENSE"</code>	Tak	Typ transakcji: przychód lub wydatek
date	<code>Date</code>	Tak	Data transakcji (nie może być z przyszłości)
description	<code>string null</code>	Nie	Opcjonalny opis transakcji (max 500 znaków)
categoryId	<code>string null</code>	Nie	Identyfikator kategorii (referencja do categories.id)
createdAt	<code>Date</code>	Tak	Data utworzenia rekordu

Tabela 19. Przedstawia opis struktury transaction

updatedAt	Date	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
id	string	Tak	Unikalny identyfikator kategorii (UUID)
name	string	Tak	Nazwa kategorii (unikalna, 1-100 znaków)
createdAt	Date	Tak	Data utworzenia rekordu
updatedAt	Date	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
id	string	Tak	"singleton"	Identyfikator (zawsze "singleton" - tylko jeden rekord)
currency	"PLN" "EUR" "USD"	Tak	"PLN"	Waluta wyświetlania

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

Tabela 22. Metody obsługi transakcji (Services)

getExpensesByCategory	<code>filters?: TransactionFilters</code>	<code>Promise<CategoryExpense[]></code>	Grupuje wydatki według kategorii. Zwraca sumę dla każdej kategorii.
getMonthlyComparison	<code>filters?: TransactionFilters</code>	<code>Promise<MonthlyData[]></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
addTransaction	<code>input: Omit<NewTransaction, "id" "createdAt" "updatedAt"></code>	<code>Promise<ActionResult></code>	Dodaje nową transakcję z walidacją. Odświeża cache stron / i /transactions.
editTransaction	<code>id: string</code> <code>input: Partial<Omit<NewTransaction, ...>></code>	<code>Promise<ActionResult></code>	Edytuje transakcję z walidacją. Odświeża cache.
removeTransaction	<code>id: string</code>	<code>Promise<ActionResult></code>	Usuwa transakcję. Odświeża cache.
validateTransaction	<code>input: { amount?, date?, description? }</code>	<code>string null</code>	(Wewnętrzna) Waliduje dane transakcji. Zwraca błąd lub null.

Warstwa serwisowa - `services/categories.ts`

Tabela 24. Metody obsługi kategorii (Services)

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

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

Tabela 25. Metody obsługi kategorii (Server Actions)

Metoda	Parametry	Zwraca	Opis
addCategory	<code>input: Omit<NewCategory, "id" "createdAt" "updatedAt"></code>	<code>Promise<ActionResult></code>	Dodaje kategorię z walidacją nazwy. Odświeża cache <code>/categories</code> .
editCategory	<code>id: string</code> <code>input: Partial<Omit<NewCategory, ...>></code>	<code>Promise<ActionResult></code>	Edytuje kategorię. Odświeża cache <code>/categories</code> , <code>/transactions</code> , <code>/</code> .

Tabela 25. Metody obsługi kategorii (Server Actions)

removeCategory	id: string	Promise<ActionResult>	Usuwa kategorię. Odświeża cache wszystkich powiązanych stron.
----------------	------------	-----------------------	---

Warstwa serwisowa - **services/settings.ts**

Tabela 26. Metody obsługi ustawień (Services)

Metoda	Parametry	Zwraca	Opis
getSettings	brak	Promise<Settings>	Pobiera ustawienia aplikacji. Tworzy domyślne (PLN) jeśli nie istnieją. Pattern: Singleton.
updateSettings	currency: Currency	Promise<Settings>	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	currency: Currency	Promise<ActionResult>	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

getDashboardData	<code>filters?: TransactionFilters</code>	<code>Promise<DashboardData></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
formatCurrency	<code>amount: number</code> <code>currency: Currency = "PLN"</code>	<code>string</code>	Formatuje kwotę według wybranej waluty. Używa Intl.NumberFormat z locale "pl-PL".
formatDate	<code>date: Date</code>	<code>string</code>	Formatuje datę w formacie DD.MM.YYYY (polski).
formatDateInput	<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>{</code> <code> dateFrom: Date,</code> <code> dateTo: Date</code> <code>}</code>	Konwertuje preset lub custom range na obiekt z datami początku i końca.

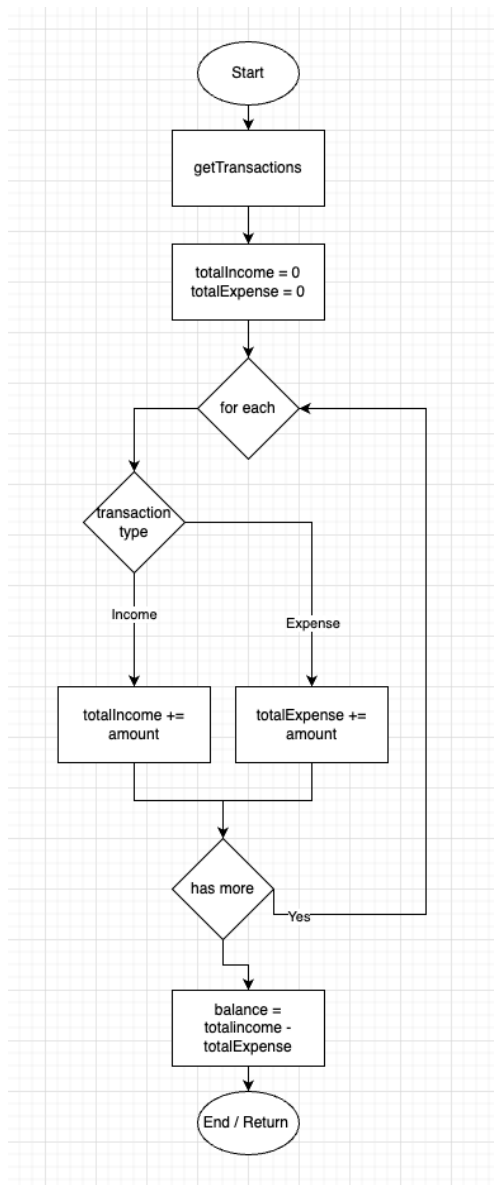
Wykaz Algorytmów**Agregacja danych finansowych (Summary)****Opis**

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 }

Użytkownie

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
```

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

Uruchomienie produkcyjne

1. Budowa aplikacji
`pnpm build`
2. Uruchomienie serwera produkcyjnego
`pnpm 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`
- Uruchomienie testów jednorazowo
`pnpm test:run`
- Uruchomienie testów z interfejsem graficznym
`pnpm test:ui`
- Uruchomienie testów z raportem pokrycia
`pnpm test:coverage`

Testy E2E

- Uruchomienie testów E2E
`pnpm test:e2e`
- Uruchomienie testów E2E z interfejsem graficznym
`pnpm test:e2e:ui`

- Uruchomienie testów E2E w trybie debug

```
pnpm test:e2e:debug
```

Struktura testów

Projekt wykorzystuje piramidę testową:

- **70%** - Testy jednostkowe (funkcje, serwisy, utility)
- **20%** - Testy integracyjne (komponenty, akcje serwerowe)
- **10%** - Testy E2E (pełne przepływy użytkownika)

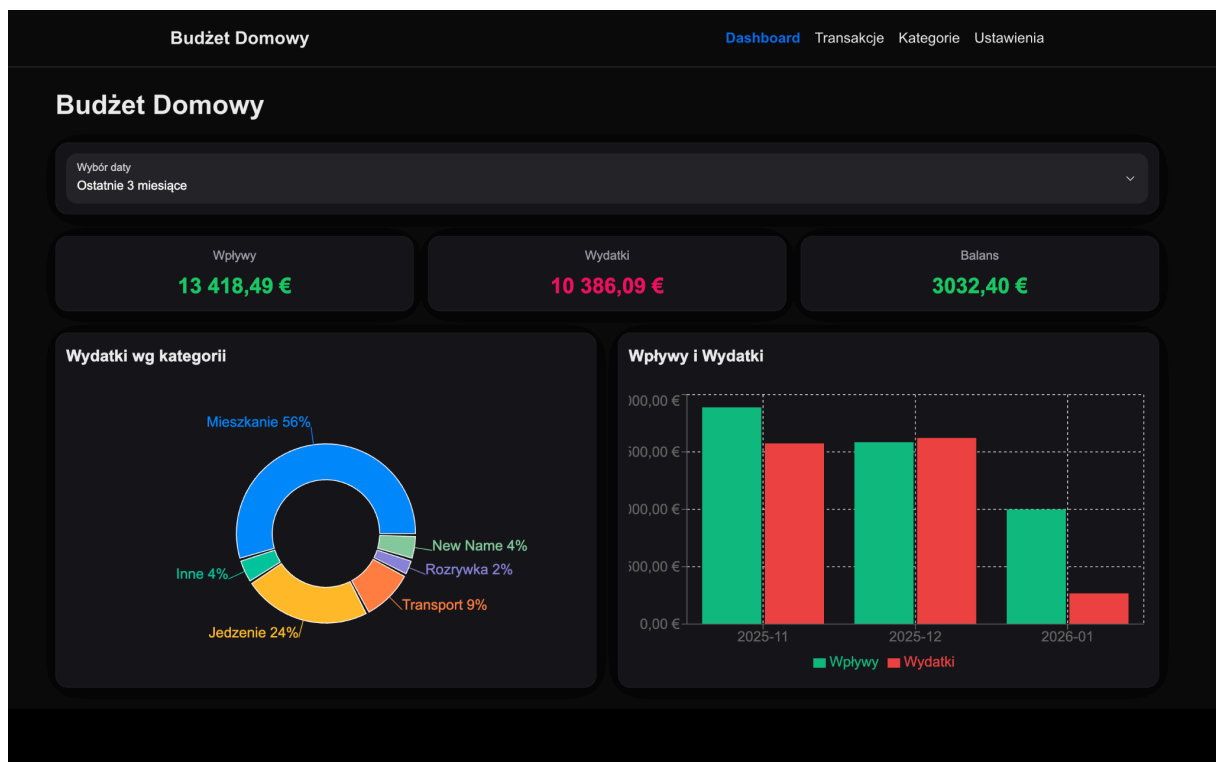
Lokalizacja testów

- Testy jednostkowe: `src/**/*.test.ts` lub `src/**/*.__tests__/*.test.ts`
- Testy komponentów: `src/components/**/*.test.tsx`
- Testy E2E: `e2e/**/*.spec.ts`

Narzędzia testowe

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

Zrzuty ekranów



Budżet Domowy

Dashboard **Transakcje** Kategorie Ustawienia

Transakcje

Dodaj transakcję

Kwota *
0.00

Typ *
Wybierz typ

Kwota musi być większa od 0

Data *
17/01/2026

Kategoria
Wybierz kateg...

Data nie może być z przyszłości

Opis
Opcjonalny opis transakcji

0/500 znaków

Dodaj transakcję

Lista transakcji 366

Przychód	10.01.2026	Salary Za prace	+3000,00 zł	Edytuj	Usuń
Wydatek	10.01.2026	Mieszkanie Czynsz	-800,00 zł	Edytuj	Usuń
Przychód	25.12.2025	Inne Wynagrodzenie miesięczne	+4753,35 zł	Edytuj	Usuń
Wydatek	24.12.2025	Inne Wyposażenie domu	-199,99 zł	Edytuj	Usuń
Wydatek	23.12.2025	Jedzenie	-379,37 zł	Edytuj	Usuń

Budżet Domowy

Dashboard Transakcje **Kategorie** Ustawienia

Kategorie

Dodaj Kategorię

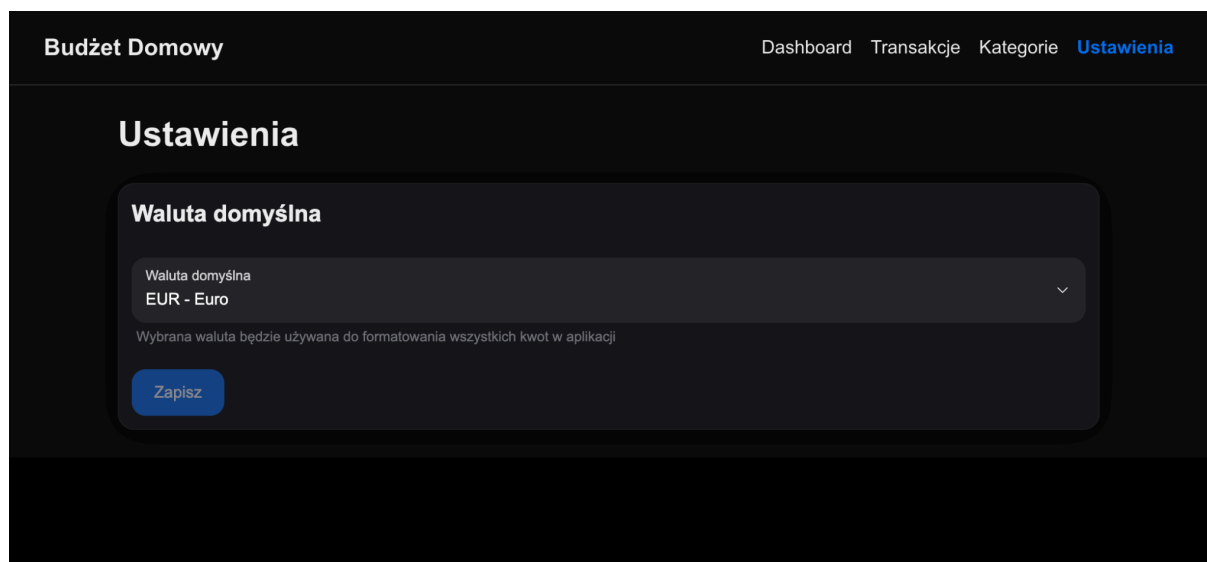
Nazwa *
Nazwa kategorii

0/100 znaków

Dodaj Kategorię

Lista Kategorii

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



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 i E2E z izolowaną bazą danych oraz raportami pokrycia kodu, w proporcji według piramidy testowej. 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](https://nextjs.org/docs)⁸ 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](https://react.dev)⁹.

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

Microsoft Corporation. *TypeScript Documentation*. [TypeScript](https://www.typescriptlang.org/docs)¹⁰.

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](https://orm.drizzle.team/docs/overview)¹¹.

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](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Intl/NumberFormat)¹².

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

⁸ <https://nextjs.org/docs>

⁹ <https://react.dev>

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

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

¹²

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

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`