

AKADEMIA GÓRNICZO-HUTNICZA

WYDZIAŁ FIZYKI I INFORMATYKI STOSOWANEJ
KIERUNEK INFORMATYKA STOSOWANA



ZAAWANSOWANE TECHNOLOGIE INTERNETOWE

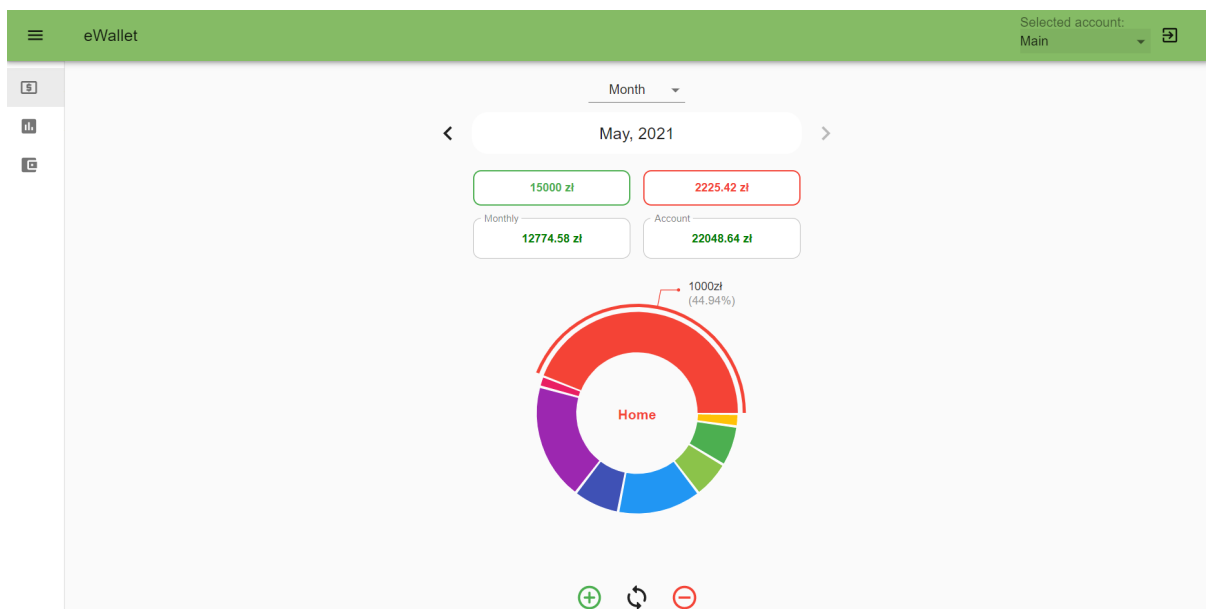
Dokumentacja projektu e-Wallet

Konrad Perłowski

Kraków, 30 maj 2021

1 Wstęp

Dokumentacja dotyczy projektu *e-Wallet* stworzonego w ramach przedmiotu Zaawansowane Technologie Internetowe. *E-Wallet* jest aplikacją internetową służącą do monitorowania i zarządzania wydatkami. Pozwala na logowanie i rejestrację nowych użytkowników, dodawanie kont oraz dodawanie wydatków i przychodów dla poszczególnych kont użytkownika i ich dokładną analizę. Wygląd aplikacji został przedstawiony na Rysunku 1.



Rysunek 1: Strona główna aplikacji

2 Technologie

Użyte technologie zostały podzielone ze względu na część serwerową oraz klienta aplikacji.

2.1 Serwer

Część stanowiąca backend aplikacji została napisana w języku Java w wersji 8 przy użyciu frameworka Spring. W projekcie znajdują się następujące zależności:

- Spring boot starter web - paczka zawierająca wszystkie niezbędne biblioteki z frameworku spring służące do pisania aplikacji webowych,
- Spring boot starter data jpa - paczka udostępniająca funkcje, które umożliwiają prostą komunikację z bazą danych,
- Spring security crypto - paczka służąca do szyfrowania hasła,
- Mysql connector java - paczka pozwalająca na połączenie z bazą danych MySQL,
- Lombok - paczka pozwalająca na generowanie metod typu getter lub setter za pomocą specjalnych adnotacji,
- Springdoc - paczka generująca interfejs użytkownika pozwalający na przeglądanie dostępnych retów.

2.2 Klient

Klient aplikacji został napisany w języku JavaScript przy użyciu biblioteki React. Klient posiada następujące zależności:

- React router dom - zależność umożliwiająca na routing w aplikacji, ponieważ react nie posiada domyślnie zaimplementowanego routingu,
- Axios - zależność ułatwiająca wykonywanie zapytań http,
- Lodash - zależność ułatwiająca wykonywanie operacji matematycznych oraz operacji na listach,
- Moment - zależność ułatwiająca operacje na datach,
- Formik - zależność służąca do pisania formularzy
- Yup - zależność służąca do walidacji formularzy
- Recharts - zależność ułatwiająca przedstawienia danych na wykresach,
- GH pages - zależność umożliwiająca publikowanie aplikacji w domenie github.io.

3 Struktura projektu

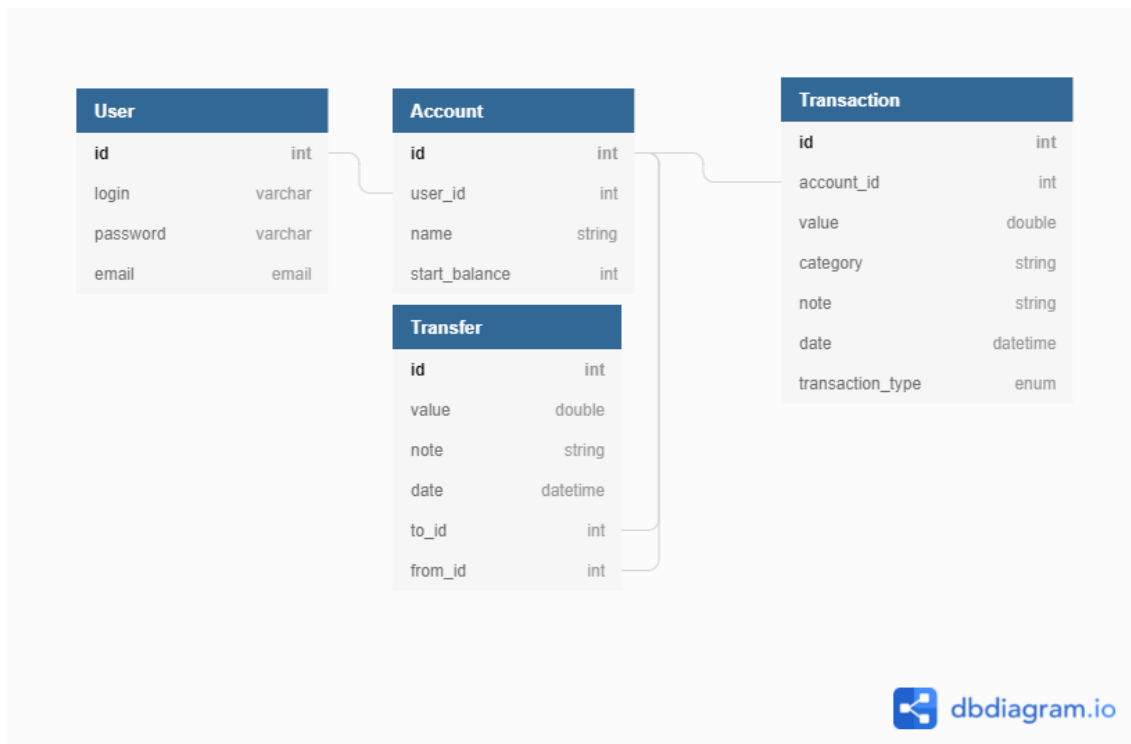
Aplikacja została napisana zgodnie ze wzorcem RESTful. Część serwerowa została opisana w rozdziale 3.2, natomiast struktura klienta jest opisana w rozdziale 3.3.

3.1 Baza danych

Dane w aplikacji przechowywane są w bazie MySQL. W bazie znajdują się cztery tabele, przechowujące informacje o danym użytkowniku, jego kontach i wydatkach.

- Tabela User zawiera informacje na temat użytkownika
- Tabela Account zawiera informacje na temat poszczególnego konta danego użytkownika
- Tabela Transaction reprezentuje poszczególny wydatek dla danego konta
- Tabela Transfer reprezentuje transfer pieniędzy pomiędzy danymi kontami użytkownika

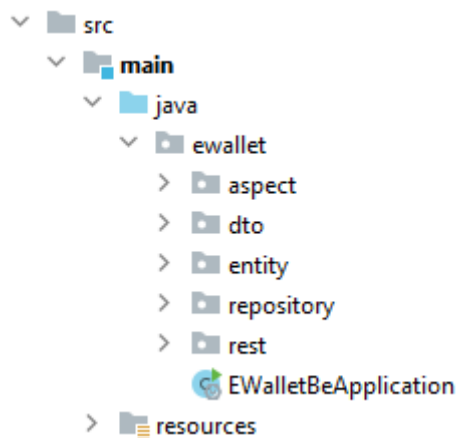
Schemat bazy danych aplikacji przedstawiony został na rysunku 2.



Rysunek 2: Schemat bazy danych

3.2 Serwer

Schemat struktury części serwerowej aplikacji został przedstawiony na rysunku 3.



Rysunek 3: Schemat części serwerowej

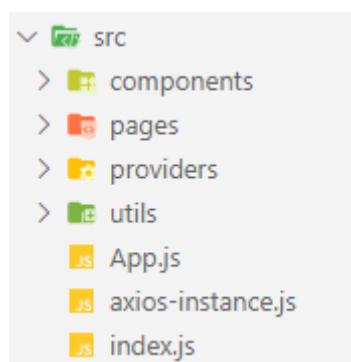
Wszystkie pliki aplikacji zostały umieszczone w odpowiednich katalogach:

- paczka *aspect* zawiera klasę wykorzystującą programowanie aspektowe w celu wypisywania dodatkowych informacji na ekran,
- paczka *dto* zawiera klasy reprezentujące *data transfer object* - typy wysyłane i odbierane w zapytaniach restowych,
- paczka *entity* zawiera klasy reprezentujące tabele w bazie danych,
- paczka *repository* zawiera repozytoria stanowiące warstwę dostępu do bazy danych,
- paczka *rest* zawiera klasy służące do udostępniania zasobów poprzez wystawienie odpowiednich restów.

Klasa *EWalletBeApplication* stanowi główną klasę aplikacji posiadającą metodę *main*.

3.3 Klient

Schemat struktury klienta aplikacji został przedstawiony na rysunku 4.



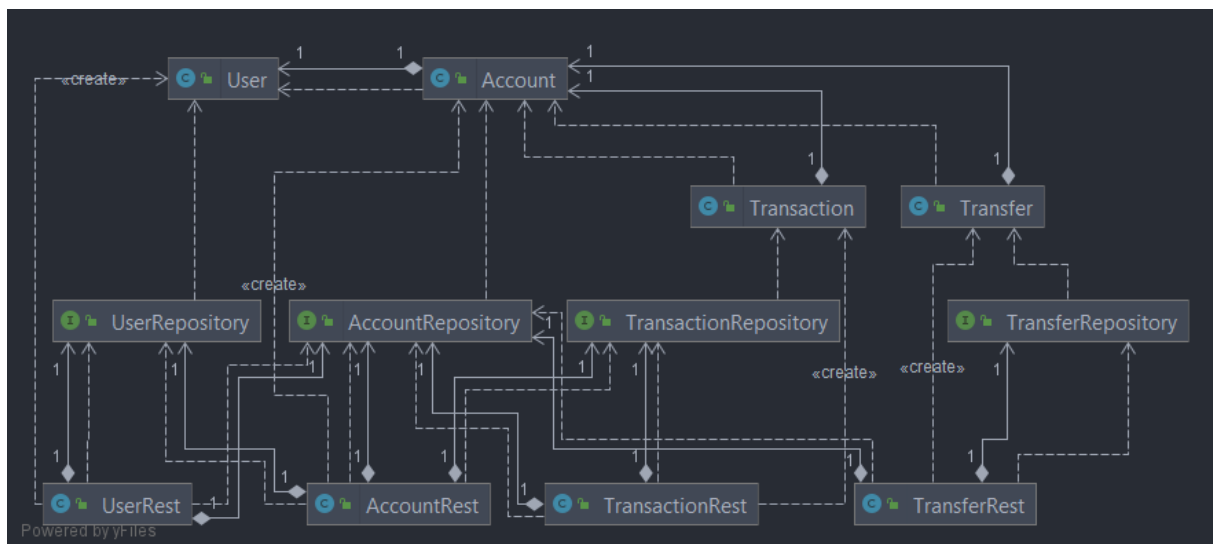
Rysunek 4: Schemat struktury klienta

Wszystkie pliki aplikacji zostały umieszczone w odpowiednich katalogach:

- katalog *components* zawiera funkcje będące reactowymi komponentami opisującymi wygląd poszczególnych elementów na stronie,
- katalog *pages* zawiera szablony poszczególnych podstron,
- katalog *providers* zawiera komponenty służące do przechowywania danych w kontekście globalnym,
- katalog *util* zawiera pliki posiadające pomocnicze funkcje,
- plik *App.js* stanowi główny plik aplikacji zawierający w sobie pozostałe komponenty,
- plik *axios-instance.js* zawiera niezbędną konfigurację do biblioteki *axios*,
- plik *index.js* renderuje aplikację na ekran.

3.4 Schemat UML

Schemat UML części serwerowej aplikacji został przedstawiony na rysunku 5.



Rysunek 5: Schemat UML serwera

4 Wdrożenie

4.1 Baza danych

Baza danych aplikacji znajduje się na stronie serwisu remotemysql. Stworzenie bazy danych wymagało stworzenie użytkownika na stronie, a następnie stworzenie bazy. Po utworzeniu użytkownik otrzymuje informacje o nazwie użytkownika, hasle oraz nazwie bazy, dzięki którym aplikacja jest w stanie na połączenie się zdalnie z serwisem bazy danych. Ustawienie niezbędnych zmiennych w aplikacji polega na ustawieniu następujących zmiennych środowiskowych: DB_USER, DB_PASSWORD, DB_HOST, DB_DATABASE

4.2 Serwer

Serwer aplikacji został umieszczony w serwisie IBM. Zbudowany jar zawierający kod aplikacji został wdrożony za pomocą komendy:

```
ibmcloud cf push ewallet-be -p eWallet-be-1.0.0.jar
```

4.3 Klient

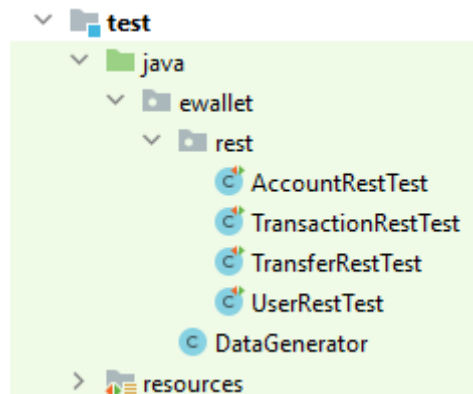
Klient aplikacji został opublikowany w domenie github.io za pomocą dodania zależności gh-pages. Po dodaniu zależności i ustawieniu odpowiednich skryptów wdrożenie aplikacji następuje po wywołaniu komendy:

```
npm run deploy
```

5 Testy aplikacji

Aplikacja posiada testy sprawdzające poprawność działania logiki części serwerowej. Testy zostały podzielone na cztery osobne klasy testujące funkcjonalność restową.

Struktura testów została przedstawiona na poniższym rysunku.



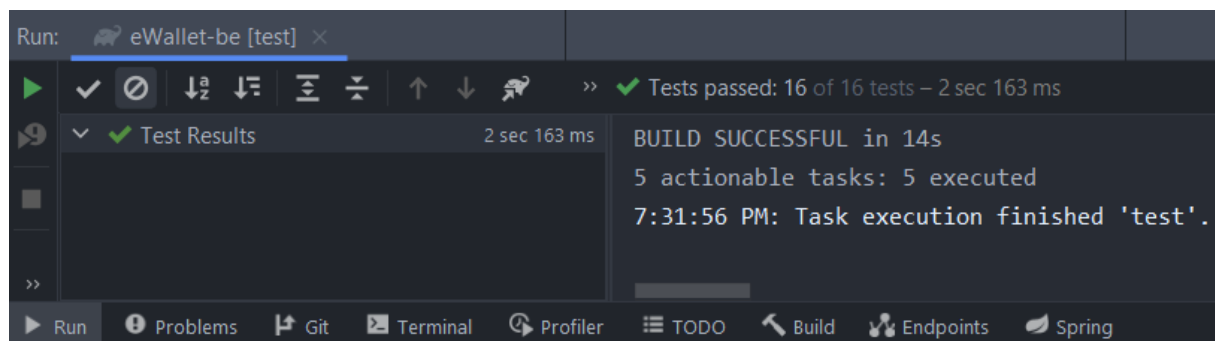
Rysunek 6: Struktura testów aplikacji

Klasa DataGenerator zawiera metody służące do generowania przykładowych użytkowników, kont, transferów oraz transakcji, pozostałe klasy zawierają testowe metody sprawdzające działanie aplikacji.

Uruchomienie testów odbywa się za pomocą komendy:

```
./gradlew test
```

Aplikacja zawiera łącznie 16 testów, poniżej przedstawiono rezultat uruchomienia:



Rysunek 7: Rezultat uruchomienia testów w programie IntelliJ

W ramach dokumentacji opisane zostaną metody testowe w ramach klasy AccountRestTest, pozostałe metody w innych klasach są analogiczne i dokładnie opisane w kodzie aplikacji.

Klasa zawiera cztery metody testowe testujące poprawność działania restów dotyczących konta użytkownika. Przed każdym testem wywoływana jest metoda `generateData` tworząca przykładowego użytkownika wraz z testowymi kontami, które następnie są usuwane po zakończonym teście w metodzie `deleteData`.

Testowe metody:

- **getForUser** - metoda sprawdzająca poprawność REST zwracającego listę kont dla danego użytkownika. Test sprawdza czy REST zwraca poprawną liczbę kont i czy w liście zawarte jest konto z id 0 zawierające wszystkie transakcje użytkownika
- **createAccount** - metoda sprawdzająca poprawność REST pozwalającego na stworzenie nowego konta. Sprawdzana jest zawartość bazy po wywołaniu REST w celu znalezienia nowo utworzonego konta
- **updateAccount** - metoda sprawdzająca poprawność REST pozwalającego na modyfikację konta. Metoda pobiera istniejące konto z bazy, nadpisuje jego wartość balansu startowego a następnie wywołuje REST w celu zaktualizowania danego konta. Sprawdzane jest czy zaktualizowane konto znajduje się w bazie i czy posiada poprawną wartość salda początkowego
- **deleteAccount** - metoda sprawdzająca poprawność REST pozwalającego na usunięcie poszczególnego konta. Pobierane są konta przykładowego użytkownika, dodawane zostają transakcje dla tego konta, a następnie konto zostaje usunięte. Sprawdzane jest czy dane konto na pewno zostało usunięte oraz czy nie znajdują się żadne transakcje powiązane dla danego konta.

6 Uruchomienie

6.1 Serwer

Do uruchomienia serwera aplikacji wymagana jest Java oraz gradle. Wymagane jest również ustawienie zmiennych środowiskowych opisanych w rozdziale 4.1.

Uruchomienie części serwerowej aplikacji polega na zbudowaniu pojedynczego pliku jar zawierającego kod aplikacji, a następnie uruchomieniu go. Budowa pliku jar polega na wywołaniu komendy:

```
./gradlew bootJar
```

Komenda stworzy plik eWallet-be-1.0.0.jar znajdujący się w folderze build/libs. Uruchomienie pliku następuje poprzez wywołanie komendy:

```
java -jar eWallet-be-1.0.0.jar
```

Istnieje również komenda pozwalająca na uruchomienie aplikacji bez konieczności budowy pliku jar:

```
./gradlew bootRun
```

6.2 Klient

Do uruchomienia klienta wymagany jest node. W celu poprawnego działania należy ustawić link do aplikacji serwera w pliku axios-instance.js.

Uruchomienie klienta aplikacji wymaga zainstalowanie potrzebnych zależności poprzez wywołanie komendy:

```
npm install
```

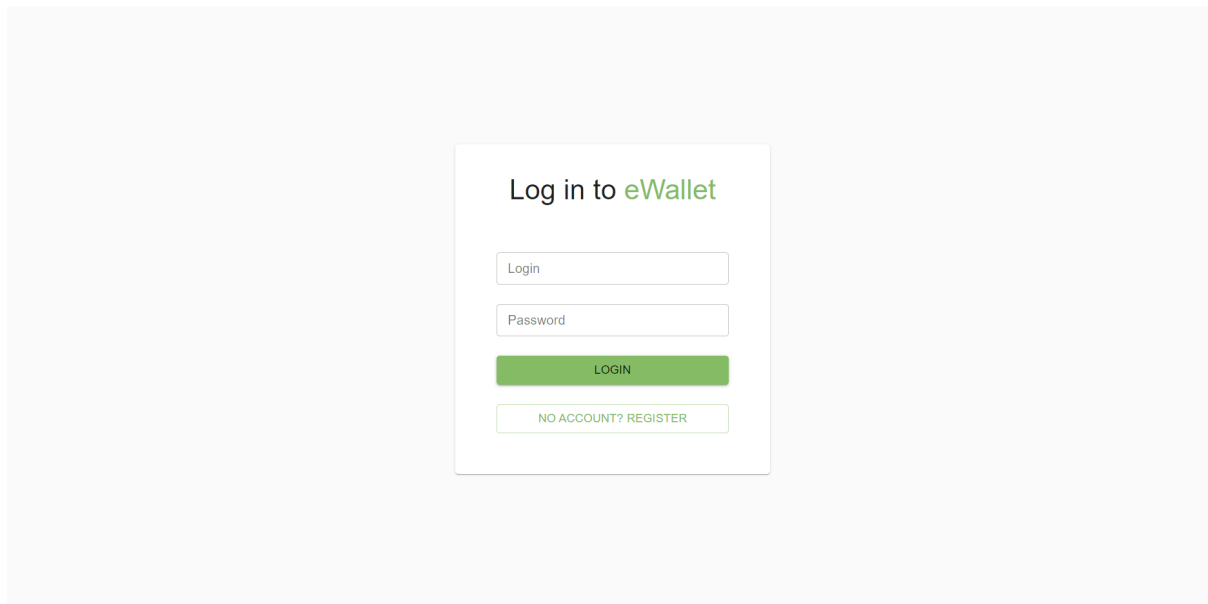
Po instalacji zależności można uruchomić aplikację:

```
npm start
```

7 Podręcznik użytkownika

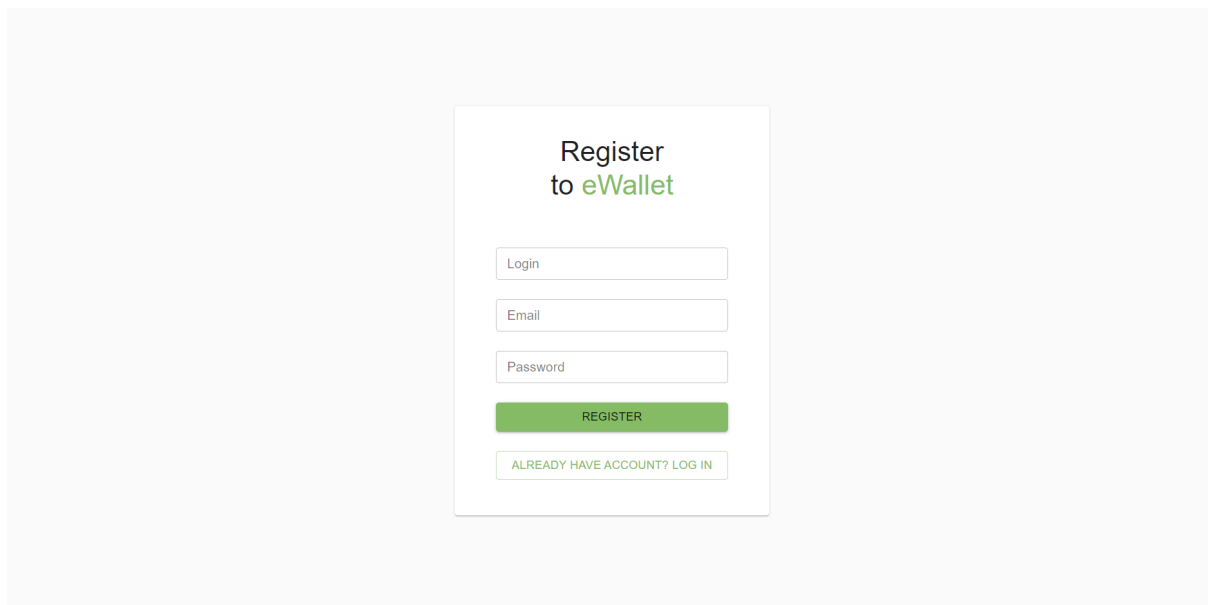
7.1 Logowanie i rejestracja

Po uruchomieniu aplikacji użytkownik zostaje przekierowany na stronę logowania. Tutaj jeśli dany użytkownik posiada już wcześniej utworzone konto możemy za pomocą loginu i hasła zalogować się do strony.

The image shows a login form for 'eWallet'. The title 'Log in to eWallet' is at the top. Below it are two input fields: 'Login' and 'Password'. A green 'LOGIN' button is positioned below the password field. At the bottom, there is a link that says 'NO ACCOUNT? REGISTER'.

Rysunek 8: Strona logowania

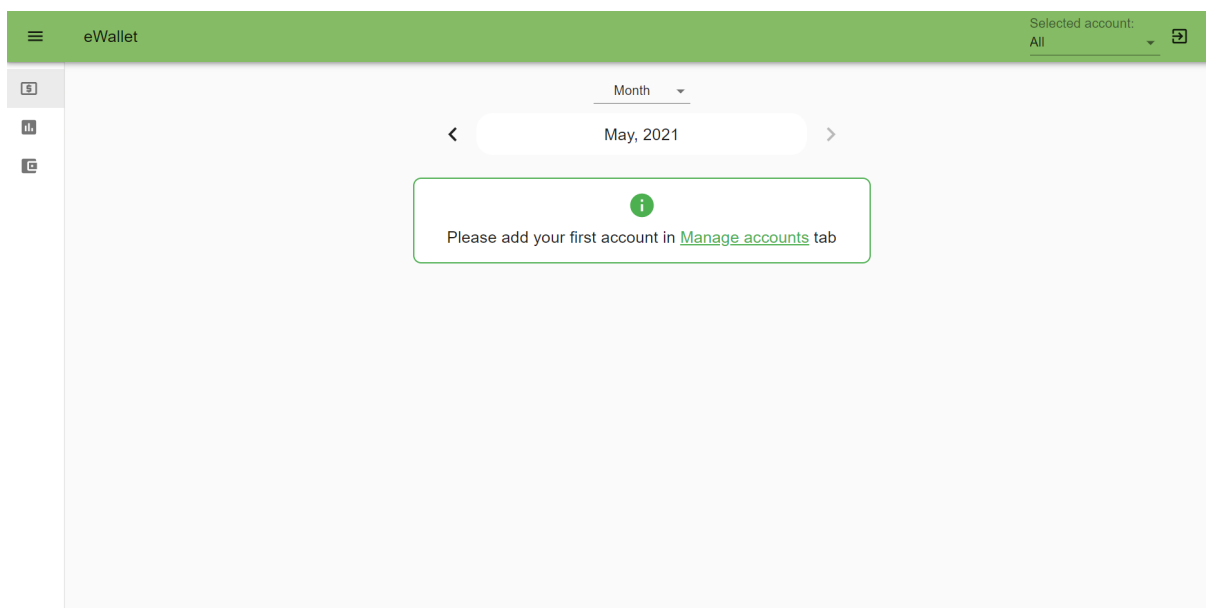
W celu rejestracji nowego użytkownika należy nacisnąć przycisk *No account? Register*, a następnie podać wymagane dane na ekranie rejestracji. Login musi być unikalny, w przypadku podania już istniejącego loginu aplikacja poinformuje o błędzie. Email musi zostać podany w poprawnym formacie, a hasło musi się składać z minimum pięciu znaków.

The image shows a registration form for 'eWallet'. The title 'Register to eWallet' is at the top. Below it are three input fields: 'Login', 'Email', and 'Password'. A green 'REGISTER' button is positioned below the password field. At the bottom, there is a link that says 'ALREADY HAVE ACCOUNT? LOG IN'.

Rysunek 9: Strona rejestracji

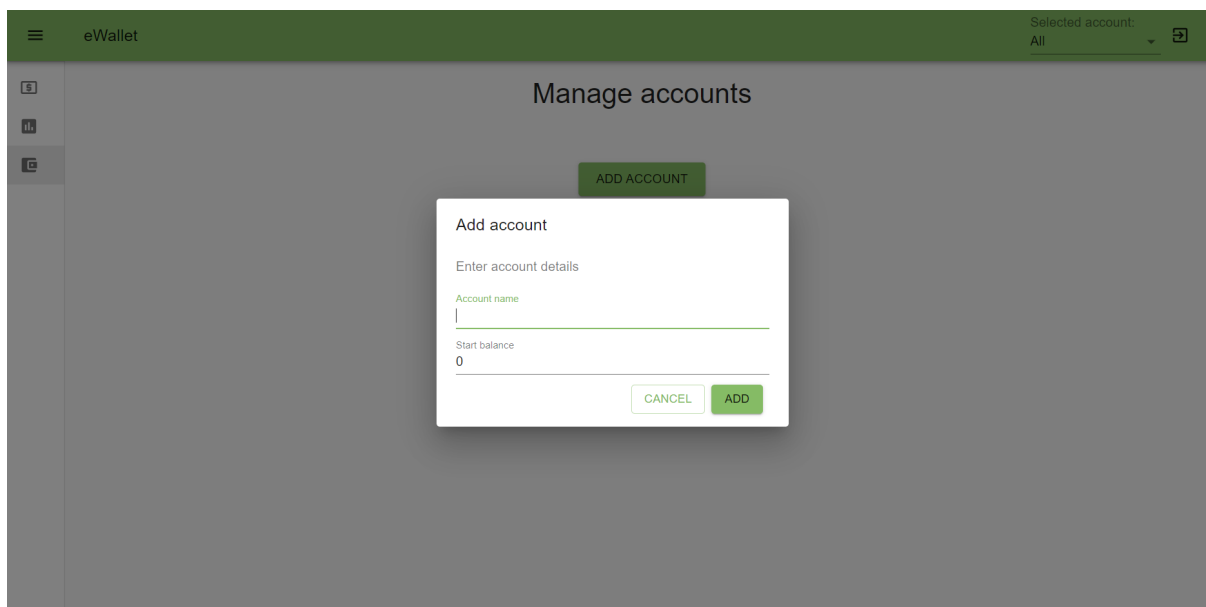
7.2 Nowy użytkownik

Po pierwszym zalogowaniu do aplikacji zostanie wyświetlona informacja o potrzebie utworzenia pierwszego konta w zakładce *Manage accounts*.



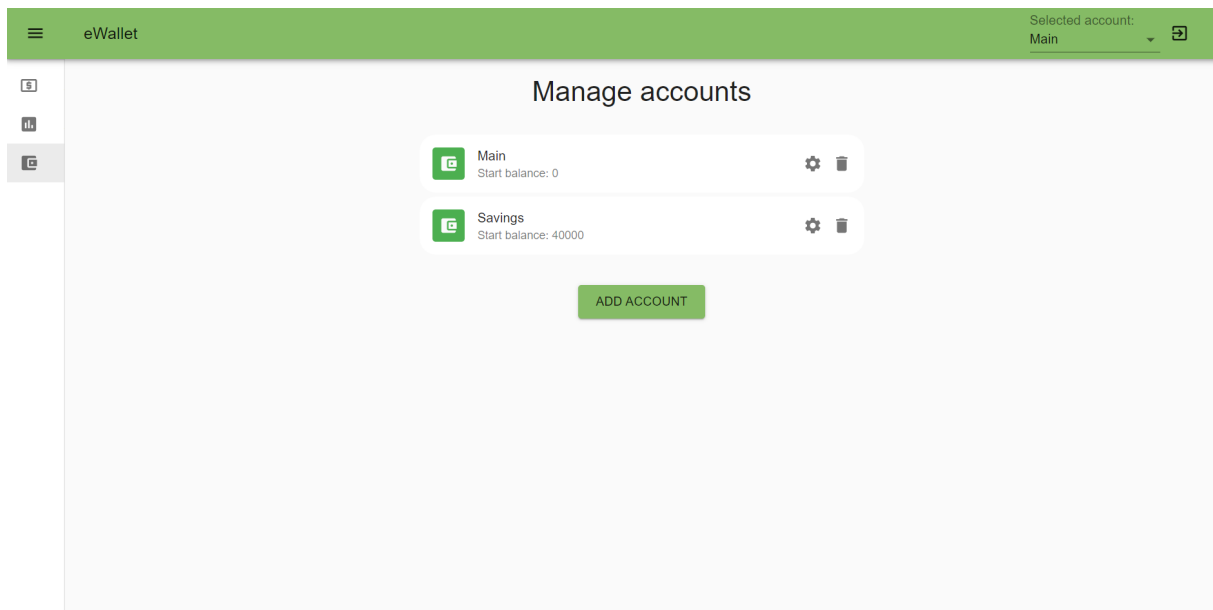
Rysunek 10: Strona startowa

Po przejściu w zakładkę *Manage accounts* i naciśnięciu przycisku Add account użytkownik zostanie poproszony o wypełnienie pola z nazwą konta oraz o podanie liczby określającej wartość początkowego balansu konta.



Rysunek 11: Dodawanie konta

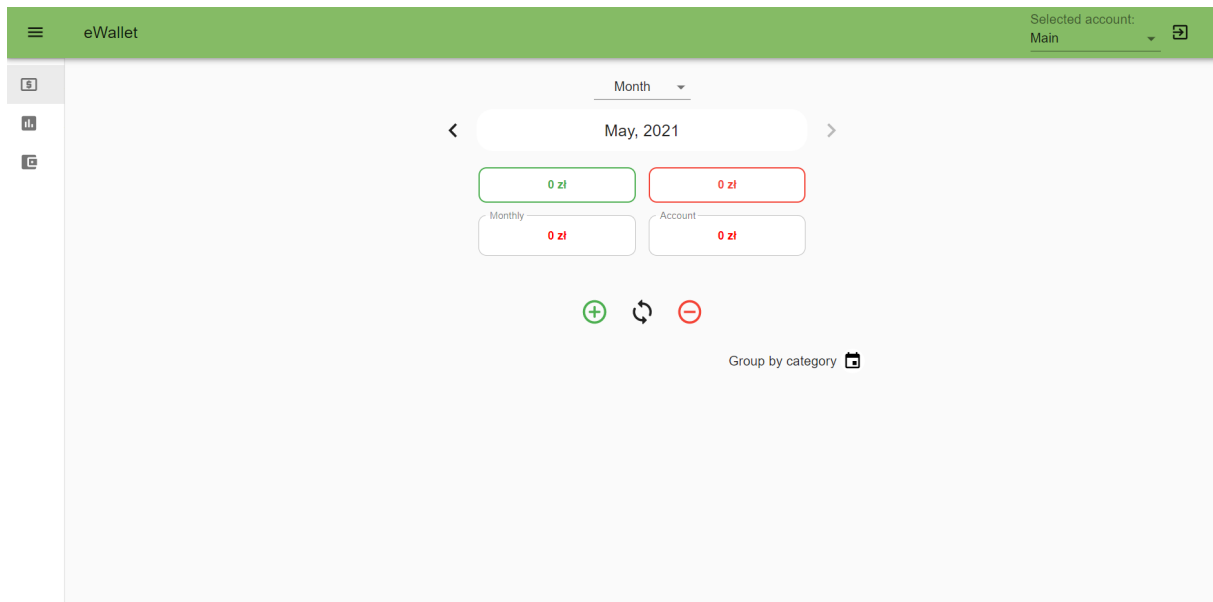
Po dodaniu konta w zakładce *Manage accounts* pojawi się lista zawierająca wszystkie założone przez użytkownika konta. Użytkownik jest w stanie modyfikować nazwę konta oraz jego wartość salda początkowego, usuwać konta lub dodawać kolejne.



Rysunek 12: Zakładka Manage accounts

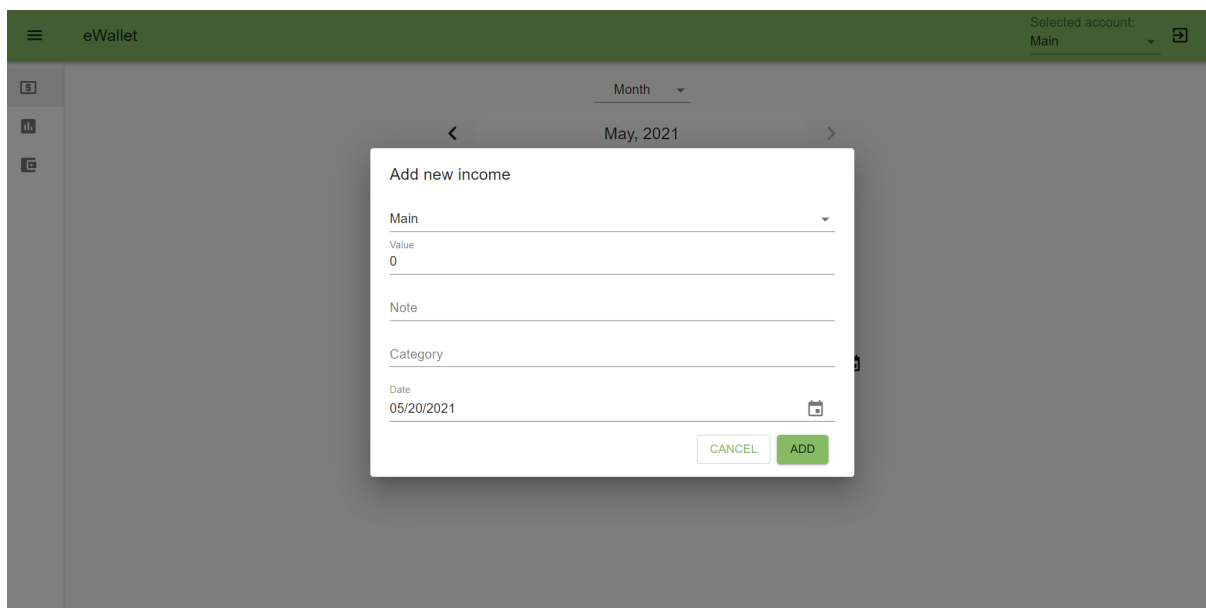
7.3 Transakcje i transfery

Po utworzeniu konta w zakładce Account page użytkownik może zobaczyć sumę wydatków i przychodów dla danego przedziału czasu, sumę tych wartości oraz łączny balans konta. Poza tym użytkownik jest w stanie dodać nowe wydatki lub transfery pomiędzy poszczególnymi kontami. W tym celu należy kliknąć w jeden z przycisków - zielony plus (nowy przychód), czarne strzałki (nowy transfer), czerwony minus (nowy wydatek), a następnie wypełnić wszystkie wymagane pola.

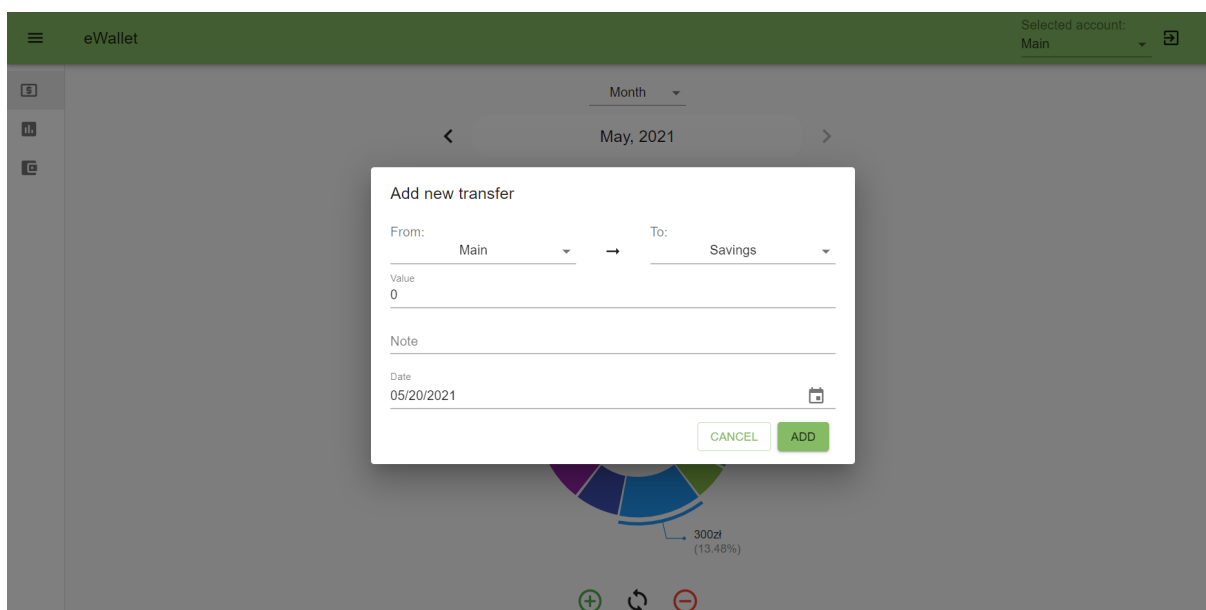


Rysunek 13: Zakładka Account

W przypadku nowej transakcji wymagane pola to konto dla którego zostanie dodana nowa transakcja, wartość kwoty transakcji, liczba ta musi być większa niż zero, notatkę (opcjonalnie), kategorie do której zostanie przypisana transakcja oraz datę transakcji. Dla transferów dochodzi pole określające konto do którego ma dojść dany transfer oraz nie wymagane jest podanie kategorii.

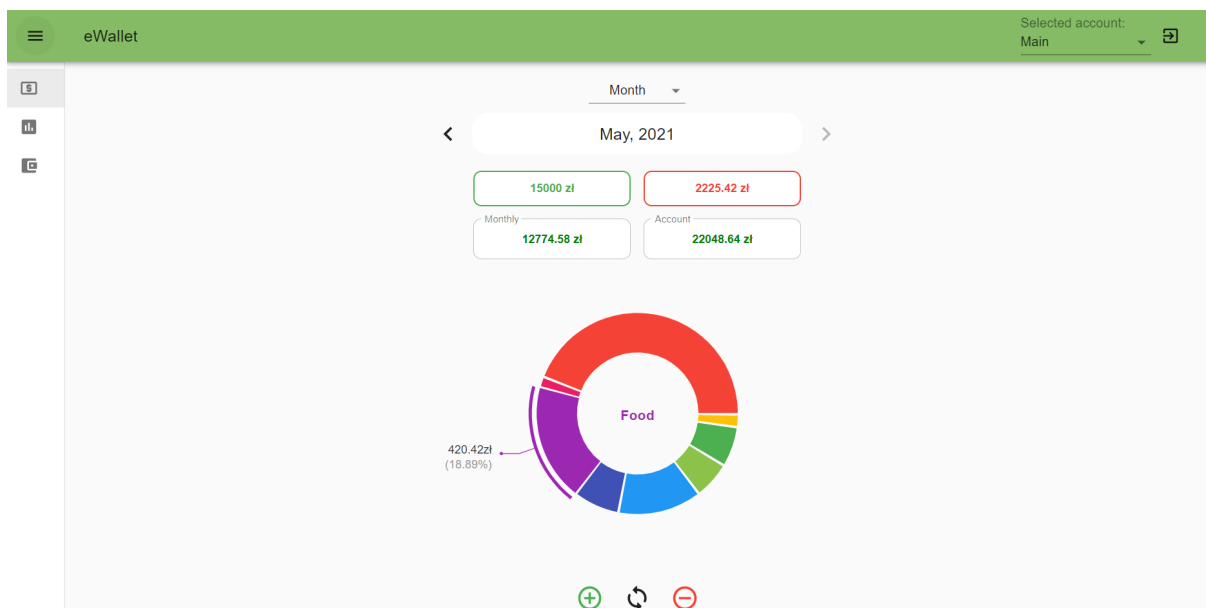


Rysunek 14: Dodanie nowej transakcji



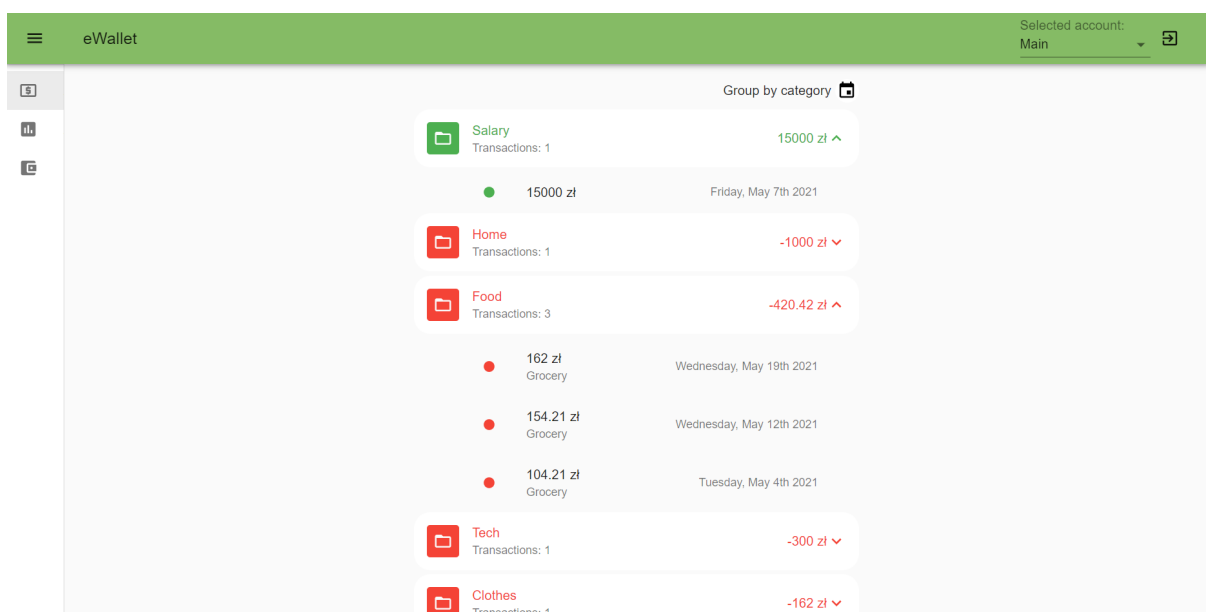
Rysunek 15: Dodanie nowego transferu

Po dodaniu transakcji oraz transferów w zakładce Account widoczny będzie wykres wydatków oraz lista transakcji dla określonego przedziału czasowego dla wybranego konta. Wybór konta znajdują się w prawym górnym rogu aplikacji, a zmiana przedziału czasowego polega na wybór zakresu (tydzień, miesiąc, rok) a następnie wybór konkretnego przedziału. Na środku strony znajdują się wykres kołowy przedstawiający wszystkie wydatki, ich wartość, kategorię oraz jaką część całości wydatków stanowią.



Rysunek 16: Wykres, zakładka Account

Poniżej znajduje się lista transakcji oraz transferów grupowana ze względu na kategorię lub ze względu na datę.



Rysunek 17: Lista transakcji i transferów

7.4 Wykresy

W aplikacji istnieje również zakładka Account stats dzięki której użytkownik ma możliwość na wizualną reprezentację danych dla wybranego konta na wykresie. Zakładka zawiera wybór roku dla którego zostaną wyświetlone dane, wykres oraz listę kategorii które mają zostać wyświetlone na wykresie poszerzoną o dwie pozycje odpowiadające sumie wszystkich wydatków i sumie wszystkich przychodów. Dzięki wizualnemu przedstawieniu danych użytkownik może w łatwy sposób analizować swoje przychody i wydatki oraz porównać wydatki dla różnych kategorii.



Rysunek 18: Zakładka Account stats

Szukanie poszczególnych kategorii jest ułatwione poprzez dodanie pola filtrującego kategorie, wszystkie kategorie związane z wydatkami na wykresie zaznaczane są w odcieniu czerwonego, natomiast wszystkie kategorie związane z przychodami zaznaczane są kolorami w odcieniu zielonego.



Rysunek 19: Filtrowanie kategorii