

Koncepcja i implementacja systemu do zarządzania budżetem domowym

(Concept and implementation of home budget management system)

Mateusz Kmita

Praca inżynierska

Promotor: dr inż. Leszek Grocholski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

31 stycznia 2022

Streszczenie

Celem pracy było stworzenie koncepcji i implementacja systemu służącego do zarządzania budżetem domowym. Motywacją do opracowania takiego systemu było dostarczenie prostego i użytecznego rozwiązania pomagającego użytkownikom w prowadzeniu osobistego budżetu. Do zrealizowania tego celu użyto nowoczesnych oraz popularnych technologii i narzędzi, które pomagają w stworzeniu aplikacji dostępnej z poziomu przeglądarki internetowej.

The aim of the work was to develop the concept and implementation of home budget management system. The motivation of the work was to provide a simple and useful tool aiding users in keeping their personal budget. This was achieved using modern and popular technologies and tools which help creating applications available from Web browsers.

Spis treści

1. Wprowadzenie	7
1.1. Analiza zagadnienia i motywacje autora	7
1.2. Przykłady innych rozwiązań	9
1.2.1. You Need a Budget (YNAB)	9
1.2.2. Mint	9
2. Opis systemu	11
2.1. Sposób instalacji i dostępu do systemu	11
2.2. Wymagania funkcjonalne	13
2.3. Wymagania niefunkcjonalne	14
2.4. Opis funkcji systemu	14
2.4.1. Zarządzanie kontami	15
2.4.2. Wyświetlanie operacji	16
2.4.3. Rejestrowanie operacji	17
2.4.4. Modyfikacja i usuwanie operacji	17
2.4.5. Wyświetlanie budżetów	19
2.4.6. Zarządzanie kategoriami	21
2.4.7. Wyświetlanie statystyk	23
3. Użyte rozwiązania technologiczne	25
3.1. Część serwerowa	25
3.1.1. Spring	26
3.1.2. Schemat bazy danych	27

3.1.3. Inne technologie użyte w części serwerowej	28
3.2. Część kliencka	29
3.2.1. Biblioteka React	29
3.2.2. Redux	29
3.2.3. Ant Design	30
3.2.4. Wykresy	31
4. Podsumowanie	33
4.1. Możliwości dalszej rozbudowy	33
4.1.1. Automatyczne dodawanie transakcji	33
4.1.2. Konta użytkowników	33
4.1.3. Podział transakcji na kilka kategorii	33
4.2. Wnioski końcowe	34
Bibliografia	35

Rozdział 1.

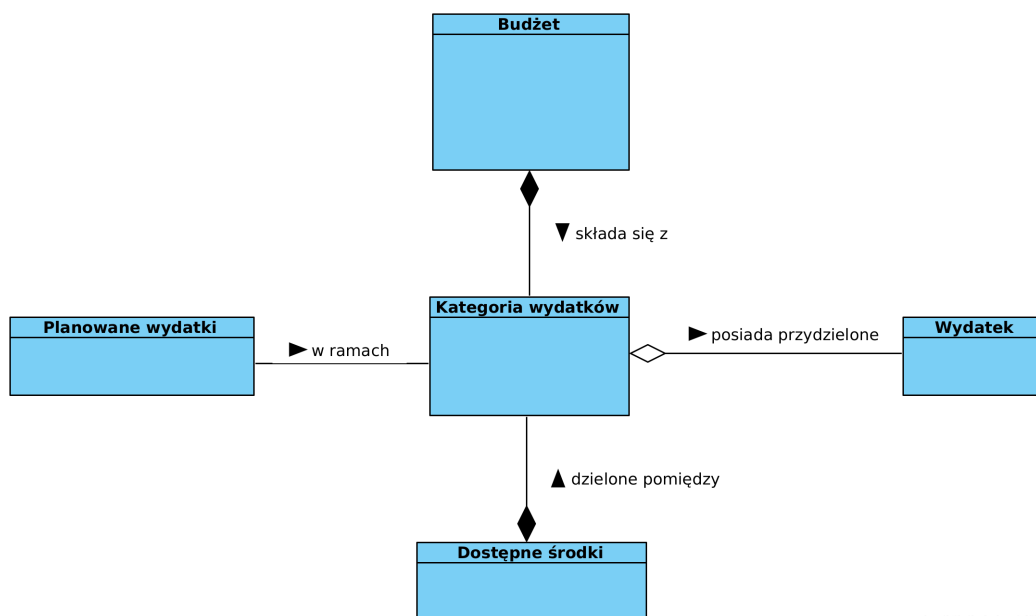
Wprowadzenie

1.1. Analiza zagadnienia i motywacje autora

W ramach tej pracy zajmuję się zagadnieniem prowadzenia budżetu domowego. Ze względu na obszerność tego zagadnienia oraz ograniczoną ilość czasu, zdecydowałem się na dostarczenie rozwiązania bardzo podobnego zagadnienia, jakim jest prowadzenie budżetu osobistego. Może ono zostać przeniesione także na problem prowadzenia budżetu domowego, czego przykłady podam w dalszej części tego rozdziału.

Istnieją różne podejścia do tego problemu [1] w zależności do indywidualnych potrzeb. Wyróżnia się kilka kroków, które należy zawsze wykonać podczas tworzenia budżetu. Powinniśmy stworzyć kategorie wydatków, aby podobne wydatki rozpatrywać razem. Kategorie możemy pogrupować na te dotyczące wydatków regularnych oraz dotyczące wydatków nieregularnych. Środki, jakimi dysponujemy, powinniśmy podzielić według tych kategorii, starając się pokryć wszystkie nasze potrzeby. Następnie potrzebne są dane dotyczące naszych wydatków. Należy więc przygotować podsumowanie kosztów jakie ponieśliśmy w danym czasie (zazwyczaj miesiącu) według kategorii, jakie wcześniej rozróżniliśmy. Na tym etapie mamy już wiedzę, w których obszarach jesteśmy na plusie, a w których wydaliśmy za dużo. Pozostało zdecydować co zrobić z pozostałymi środkami. Niektóre z popularnych metod prowadzenia budżetów to [2]:

- **Metoda 50/30/20** - polega na podziale środków poprzez przypisanie 50% przychodów na niezbędne wydatki (np. czynsz, zakupy spożywcze), 30% na wydatki dodatkowe (np. wyjście do restauracji) oraz pozostałych 20% na oszczędności i spłaty długów.
- **Metoda kopertowa** - możemy jej używać z gotówką. Kwotę przypisaną każdej kategorii umieszczamy w oddzielnej kopercie i robimy zakupy używając tylko pieniędzy z danej koperty.



Rysunek 1.1: Model przedstawiający zagadnienia budżetu osobistego na diagramie UML

- **Metoda “zapłać najpierw sobie”** - polega na odłożeniu odpowiedniej ilości środków na długi i oszczędności oraz przeznaczaniu reszty w dowolny wygodny dla siebie sposób

Model analityczny tego zagadnienia widoczny jest na diagramie (Rys. 1.1) stworzonym z użyciem narzędzia Visual Paradigm [3].

Jak wcześniej wspomniałem, w pracy tej zająłem się opracowaniem rozwiązania dla zagadnienia prowadzenia budżetu osobistego. Można jednak użyć podanego wyżej modelu, aby poradzić sobie także z prowadzeniem budżetu domowego. Rozważmy sytuację, w której budżet domowy chce prowadzić mieszkająca razem rodzina. W takim przypadku dostępnymi środkami mogą być zarobki rodziców. Stworzą oni kategorie dla wszystkich wydatków wymaganych do prowadzenia domu rodzinnego, oszczędzania, czy też oddzielne na tzw. kieszonkowe dla dzieci. Jeśli budżet domowy chcieliby prowadzić np. mieszkający razem studenci, dostępnymi środkami byłyby składki każdego z nich na wspólny budżet. Mogliby oni stworzyć kategorie dla wydatków dotyczących prowadzenia wspólnego domu takie jak czynsz, artykuły gospodarstwa domowego, opłaty za telewizję, Internet itd.

Z jednej strony budżet osobisty wymaga poświęcenia dodatkowego czasu na stworzenie oraz jego regularne prowadzenie. Z drugiej strony możemy w zamian liczyć na większą kontrolę nad stanem naszego portfela. Przy założeniu, że taki budżet prowadzony jest regularnie, mamy wiedzę o tym, ile pieniędzy możemy jeszcze wydać. Może to pozwolić nawet na uniknięcie długów. Osoba prowadząca budżet osobisty i historię wydatków może także łatwiej analizować swoje nawyki konsumenckie oraz je poprawiać, aby móc efektywniej zarządzać swoimi pieniędzmi w przyszłości.

Zdecydowałem się na stworzenie aplikacji, która będzie dostarczała podstawowe funkcjonalności do zrealizowania opisanego wyżej zadania oraz będzie na tyle prosta, aby nowy użytkownik mógł jak najmniejszym kosztem zacząć prowadzić swój budżet osobisty. W dalszej części pracy opisałem jakie funkcje udostępnia aplikacja. Następnie skupiłem się na przedstawieniu i uzasadnieniu mojego rozwiązania pod względem technologicznym. W końcowej części pracy podsumowałem zrealizowane zadania oraz podałem przykłady sposobów na dalsze ulepszenie aplikacji.

1.2. Przykłady innych rozwiązań

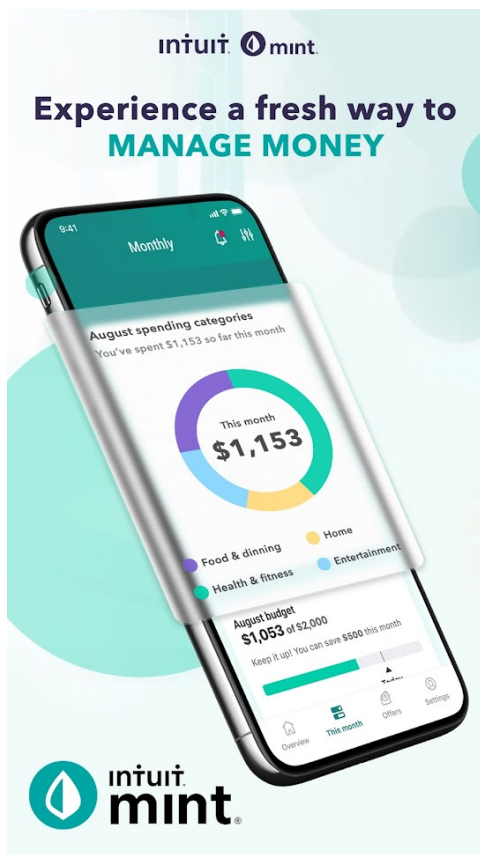
Użytkownicy zainteresowani tematyką prowadzenia budżetu osobistego mają duży wybór dostępnych rozwiązań. Istnieją zarówno aplikacje dostępne przez przeglądarkę, tradycyjne programy i aplikacje na urządzenia mobilne. Wiele programów dostępnych jest jednocześnie na wielu rodzajach urządzeń dla wygody użytkowników. Ja skupiłem się na rozwiązaniach, które dostępne są przez przeglądarkę, gdyż są najbardziej zbliżone do rozwiązania proponowanego przeze mnie.

1.2.1. You Need a Budget (YNAB)

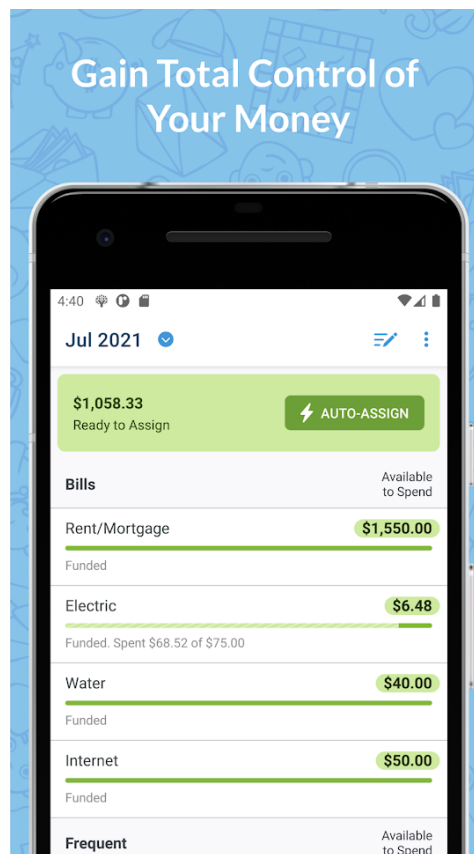
Aplikacja You Need a Budget [4] jest dostępna przez przeglądarkę internetową oraz w postaci aplikacji na urządzenia mobilne na systemy operacyjne Android i iOS. Jest to program płatny, który kosztuje 14,99\$ miesięcznie lub 89,99\$ rocznie [5]. Idea tej aplikacji jest nie tyle zautomatyzowanie całego procesu zarządzania budżetem i całkowite wyręczenie użytkownika, co nauczenie go jak taki budżet tworzyć, więc aplikacja YNAB wymaga od użytkownika poświęcenia czasu na ręczne tworzenie budżetu. Filozofia tworzenia budżetu osobistego w tej aplikacji polega na rozdzielaniu pieniędzy co do grosza na nasze cele. Program ten posiada użyteczną funkcję automatycznego pobierania wydatków z konta bankowego.

1.2.2. Mint

Mint [6] to darmowy program dostępny przez przeglądarkę internetową oraz jako aplikacja na urządzenia mobilne na systemy operacyjne Android i iOS. Aplikacja dostępna jest za darmo, ale wyświetla użytkownikom reklamy ofert od swoich partnerów biznesowych. To rozwiązanie posiada wiele funkcji automatyzujących cały proces zarządzania budżetem. Automatycznie pobiera wydatki z połączonego konta bankowego i przypisuje je do odpowiednich kategorii. Posiada też dodatkowe funkcje oprócz prowadzenia budżetu takie jak przypominanie o płatnościach za subskrypcje i rachunki lub wysyłanie użytkownikom porad, jak mogą lepiej zarządzać pieniędzmi. Dostępne są także informacje o zdolności kredytowej, funkcje związane z inwestowaniem oraz inne funkcjonalności dotyczące pieniędzy. Mamy więc tutaj więcej funkcji



(a) Mint



(b) You Need A Budget

Rysunek 1.2: Inne rozwiązania dotyczące prowadzenia budżetu osobistego

niż w poprzednim rozwiązaniu.

Rozdział 2.

Opis systemu

2.1. Sposób instalacji i dostępu do systemu

Część serwerowa

Część serwerowa wymaga do uruchomienia programu PostgreSQL [7] w wersji 13.4 oraz Java Runtime Enviroment (JRE) [8] w wersji 17.0.1. Część serwerowa została już zbudowana i dostępna jest w postaci pliku JAR, w którym zawarty jest wbudowany serwer aplikacji Tomcat.

Samodzielne budowanie aplikacji

Źródła programu dostępne są w repozytorium Git [9] na stronie Github [10] pod adresem <https://github.com/mat-kmita/budget-manager> i możemy ich użyć, jeśli chcemy samodzielnie zbudować program. Do pobrania plików źródłowych możemy użyć na przykład polecenia:

```
$ git clone  
https://github.com/mat-kmita/budget-manager.git
```

Do zbudowania programu wymagane jest narzędzie Gradle [11]. Aby zbudować program należy w katalogu `thesis_project` z plikami źródłowymi części serwerowej wykonać:

```
$ gradle bootJar
```

Przygotowanie bazy danych

Program wymaga stworzenia bazy danych PostgreSQL. Potrzebujemy do tego użytkownika bazy danych. Żeby go stworzyć użyjemy narzędzia `createuser` [12]:

```
$ createuser -d -P -h localhost -p 5432 -U postgres -W  
thesis
```

Program zapyta wtedy o hasło dla tworzonego użytkownika, które należy podać i zapamiętać. Nazwa użytkownika to thesis, ale można ją dostosować według własnego uznania przy czym będzie to wymagało zmiany konfiguracji aplikacji. Opcja h ustawia nazwę hosta, na którym działa baza danych, opcja p ustawia port, na którym baza nasłuchuje, opcja U ustawia istniejącego użytkownika bazy, jako który chcemy wykonać to polecenie. Opcje należy dostosować zgodnie z lokalną konfiguracją. Następnym krokiem jest już stworzenie bazy danych. Użyjemy do tego narzędzia createdb [13]:

```
$ createdb -h localhost -p 5432 -O thesis -U thesis  
thesis_db
```

Podobnie jak wyżej opcje h i p odpowiadają za ustalenie adresu bazy danych. Opcja O ustawia stworzonego wcześniej użytkownika jako właściciela bazy. W powyższym poleceniu nadajemy bazie nazwę thesis_db, ale możemy to dostosować pamiętając, że będzie to wymagało zmiany domyślnej konfiguracji aplikacji.

Konfiguracja aplikacji

Aplikacja konfigurowana jest przez plik konfiguracyjny `application.properties`. Znajduje się ona w źródłach aplikacji w katalogu `thesis_project/src/main/resources/application.properties`. Możemy dostosować opcje w tym pliku zanim zbudujemy aplikację. Jeśli aplikacja jest już zbudowana, możemy umieścić w katalogu, w którym znajduje się plik JAR plik o nazwie `application.properties`, który będzie nadpisywał domyślne opcje. Przed uruchomieniem aplikacji należy ustawić hasło użytkownika bazy danych. W pliku konfiguracyjnym należy wpisać:

```
spring.datasource.password=<hasło użytkownika bazy danych>
```

Jeśli użyliśmy niestandardowej nazwy użytkownika należy także zmienić opcję:

```
spring.datasource.username=<nazwa użytkownika bazy danych>
```

Jeśli chcemy zmienić hosta, port lub nazwę bazy danych, z którą aplikacja ma się łączyć powinniśmy zmodyfikować opcję:

```
spring.datasource.url=<adres bazy danych>
```

Format adresu bazy danych opisany jest na stronie internetowej <https://www.postgresql.org/docs/current/libpq-connect.html#LIBPQ-CONNSTRING>. Pozostałe opcje konfiguracji opisane są na stronie internetowej <https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>.

Uruchamianie programu

Gdy skończyliśmy konfigurację aplikacji dla naszego środowiska, możemy ją uruchomić. W katalogu, w którym znajduje się plik JAR o nazwie `application.jar` wykonujemy polecenie:

```
$ java application.jar
```

Część kliencka

Do uruchomienia części klienckiej wymagane jest Node.js [14] w wersji 17.0.1 oraz npm [15] w wersji 8.3.2.

Aplikacja została już zbudowana i jest gotowa do uruchomienia. Kod źródłowy znajduje się w katalogu `thesis-front-end`. Będąc w tym katalogu należy wykonać polecenia:

```
$ npm install -g serve  
$ serve -s build
```

Aplikacja zostanie uruchomiona i będzie nasłuchiwała na porcie 3000.

2.2. Wymagania funkcjonalne

- Udostępnianie użytkownikowi gotowego zestawu kategorii wydatków do podziału budżetu.
- Możliwość dodawania nowych kategorii przez użytkownika.
- Możliwość zmiany nazwy kategorii.
- Możliwość usuwania istniejących kategorii.
- Możliwość podziału portfela na konta odpowiadające rzeczywistym kontom użytkownika lub będące wirtualnym podziałem środków.
- Możliwość edycji właściwości konta oraz usuwania konta.
- Możliwość transferu środków pomiędzy kontami wraz zapisem informacji dodatkowych o transferze, takich jak: data jego przeprowadzenia, komentarz.
- Możliwość edycji danych zarejestrowanego transferu i całkowitego usunięcia go.
- Możliwość rejestracji przychodów i wydatków wraz z informacjami takimi jak: data operacji, kwota, kategoria, komentarz, płatnik.
- Możliwość edycji informacji o przychodach i wydatkach oraz usuwania tych operacji.

- Możliwość tworzenia budżetów poprzez przydzielanie dostępnych środków pomiędzy istniejące kategorie.
- Możliwość zmiany środków przydzielonych do kategorii w ramach budżetu.
- Możliwość wyświetlania budżetu w postaci listy kategorii wraz z informacjami o sumie środków w ramach każdej kategorii wydanej oraz dostępnymi jeszcze środkami.
- Możliwość wyświetlania informacji o sumie środków wydanych każdego dnia z danego przedziału czasowego.
- Możliwość wyświetlania statystyk o procentowym udziale wydatków w każdej kategorii we wszystkich wydatkach.

2.3. Wymagania niefunkcjonalne

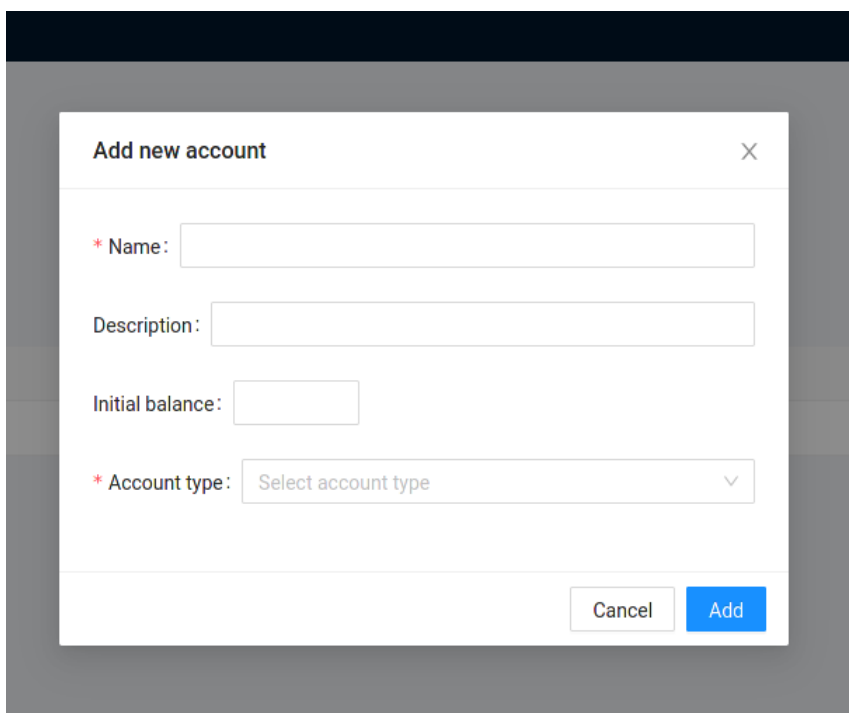
- Interfejs graficzny aplikacji powinien być intuicyjny, spójny i prosty w obsłudze.
- Aplikacja powinna sprawdzać poprawność danych wprowadzanych przez użytkownika.
- Błędy w działaniu aplikacji powinny być przekazywane użytkownikowi z prostym i zrozumiałym wytłumaczeniem, nieudostępniając informacji o wewnętrznym działaniu aplikacji.
- Dostęp do bazy danych musi być chroniony przed nieuprawnionymi osobami.

2.4. Opis funkcji systemu

Koncepcja aplikacji polega na założeniu, że użytkownik będzie samodzielnie regularnie rejestrował w aplikacji wszystkie swoje wydatki oraz wpływy oraz ręcznie tworzył budżet rozdzielając pieniądze pomiędzy różne kategorie. Program będzie te informacje zapisywał oraz obliczał środki dostępne w ramach budżetu.

Poniżej wytłumaczyłem podstawowe pojęcia używane w aplikacji.

- **Transakcje** - operacje pieniężne, które są wydatkami lub wpływami z zewnętrznych źródeł. Te operacje mogą wpływać na stan budżetu.
- **Transfery** - operacje pieniężne pomiędzy dwoma różnymi kontami, które nie wpływają na budżet, ale dotyczą przeniesienia środków wewnątrz portfela użytkownika
- **Kategoria transakcji** - podział wszystkich transakcji na różne kategorie dotyczące tej samej dziedziny. Ocena do jakiej kategorii należy transakcja leży całkowicie po stronie użytkownika.

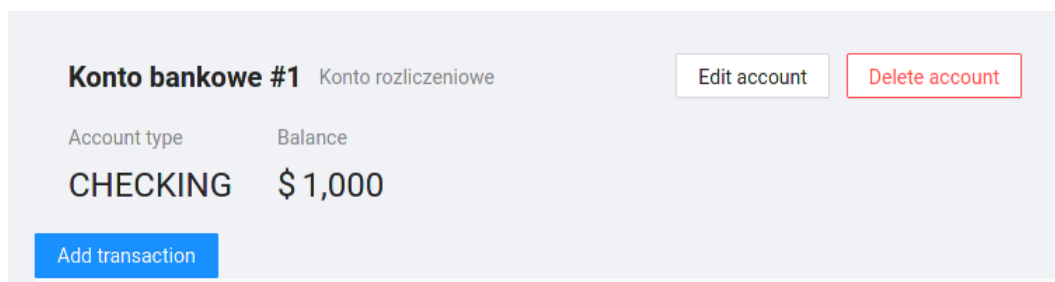


Rysunek 2.1: Formularz tworzenia nowego konta

- **Budżet** - budżet to comiesięczny podział wpływów pomiędzy różne kategorie wraz z wydatkami w tym miesiącu oraz informacją o środkach dostępnych jeszcze w ramach każdej z kategorii.

2.4.1. Zarządzanie kontami

Użytkownik posiada możliwość podziału swojego portfela na różne konta. Ta funkcja pozwala śledzić ilość pieniędzy ulokowanych w różnych źródłach. Dla wygody użytkownika konta są podzielone pomiędzy cztery różne rodzaje: rozliczeniowe, oszczędnościowe, gotówkę oraz inne. Lista kont wyświetlana jest w panelu nawigacyjnym po lewej stronie ekranu. Możemy stamtąd przejść do listy transakcji lub transferów przypisanych do każdego z kont. Na górze strony będą wtedy wyświetlane informacje o koncie, którego ekran dotyczy. Znajdziemy tam nazwę konta, opis, rodzaj i aktualny stan środków na koncie. Obok tych informacji znajdują się przyciski “Delete account” oraz “Edit account” służące do usuwania konta i edycji danych o nim. Aby dodać nowe konto należy nacisnąć na przycisk “Create new account” w panelu nawigacyjnym. Wyświetla się wtedy formularz, w którym należy podać nazwę i rodzaj konta oraz opcjonalnie opis i początkowy stan tego konta.



Rysunek 2.2: Widok informacji o koncie

Add transaction							
Id	Date	Payee	Category	Memo	Amount	Delete	Edit
95	2022-01-03	Sklep spożywczy	Groceries	Zakupy	-250		
97	2022-01-02	Stacja paliw	Gas		-300.01		
99	2021-12-15	Sklep z prezentami	Gifts	Prezenty świąteczne	-349.99		
100	2021-12-15	Właściciel mieszkania	Rent/Mortgage		-1800		
101	2021-12-14	Weterynarz	Medical		-300		
98	2021-12-09	Restauracja	Eating Out		-53.12		
96	2021-12-03	Sklep spożywczy	Groceries	Zakupy	-267.81		
94	2021-12-01	Pracodawca	Incomes	Wyplata	5000		

< 1 > 10 / page

Rysunek 2.3: Tabela przedstawiająca transakcje dla danego konta

2.4.2. Wyświetlanie operacji

Aby wyświetlić operacje danego typu należy w panelu nawigacyjnym po lewej stronie wybrać “Transactions” lub “Transfers” dla jakiegoś konta. Zostanie wtedy wyświetlona tabela z transakcjami lub transferami dotyczącymi tego konta. Operacje posortowane są według daty ich wykonania od najnowszych do najstarszych. Dla każdej transakcji wyświetlane są: identyfikator tej operacji, data wykonania, odbiorca płatności (lub nadawca dla przychodów), kategoria, opis, kwota, przycisk otwierający okienko modyfikacji operacji i przycisk usuwający operację. Dla transferów wyświetlane są: identyfikator transferu, data wykonania, nazwa drugiego konta, opis oraz kwota. Dane w tabeli podzielone są na strony. Użytkownik może się między nimi przemieszczać używając przycisków z numerem strony lub wpisując numer dowolnej strony, do której chce przejść. Ilość elementów na stronie może zostać wybrana spośród 10, 20, 50, 100.

Make a transfer						
Id	Date	To account	Memo	Amount	Delete	Edit
3	2022-06-08	mBank RR		1000		
9	2022-06-08	mBank RR		850		
6	2022-03-29	mBank RR		980		
5	2021-11-18	mBank RR	Wpłata w wplatomacie	-150		
8	2021-11-17	mBank RR	Wpłata w oddziale banku	-2000		
10	2021-09-19	mBank RR		50		
2	2021-09-10	mBank RR	Wypłata z bankomatu	200		
11	2021-06-10	mBank RR		350		
1	2021-06-09	mBank RR	Wypłata z bankomatu	600		
4	2021-05-20	mBank RR		950		

< 1 2 > 10 / page Go to Page

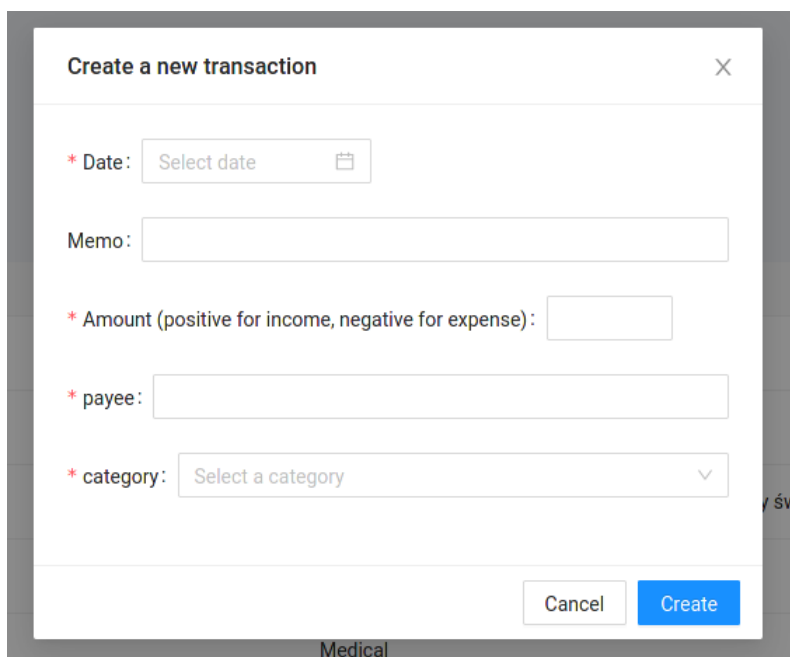
Rysunek 2.4: Tabela przedstawiająca transfery dla danego konta

2.4.3. Rejestrowanie operacji

Zgodnie z koncepcją aplikacji, użytkownik powinien regularnie rejestrować w programie swoje wydatki, przychody i transfery pieniędzy. Aby to zrobić, należy nacisnąć na przycisk “Add transaction” lub “Make a transfer”. Są one widoczne na ekranach transakcji i transferów dla danego konta. Pierwszy z nich służy do rejestrowania transakcji, a drugi do przenoszenia środków na inne konto. Naciśnięcie tych przycisków powoduje wyświetlenie okienka z formularzem. Należy w nim podać dane o operacji. Niektóre pola są wymagane i oznaczone są one znakiem czerwonej gwiazdki. Pozostałe pola są opcjonalne.

2.4.4. Modyfikacja i usuwanie operacji

Przy każdej operacji znajdują się przyciski pozwalające na jej edycję lub usunięcie. Naciśnięcie przycisku do edycji operacji spowoduje wyświetlenie okienka z formularzem, w którym możemy zmienić dane operacji a następnie je zapisać. Naciśnięcie przycisku do usuwania operacji spowoduje wyświetlenie okienka, w którym należy usunięcie potwierdzić lub anulować. Ma to za zadanie chronić przed przypadkowym usuwaniem operacji.

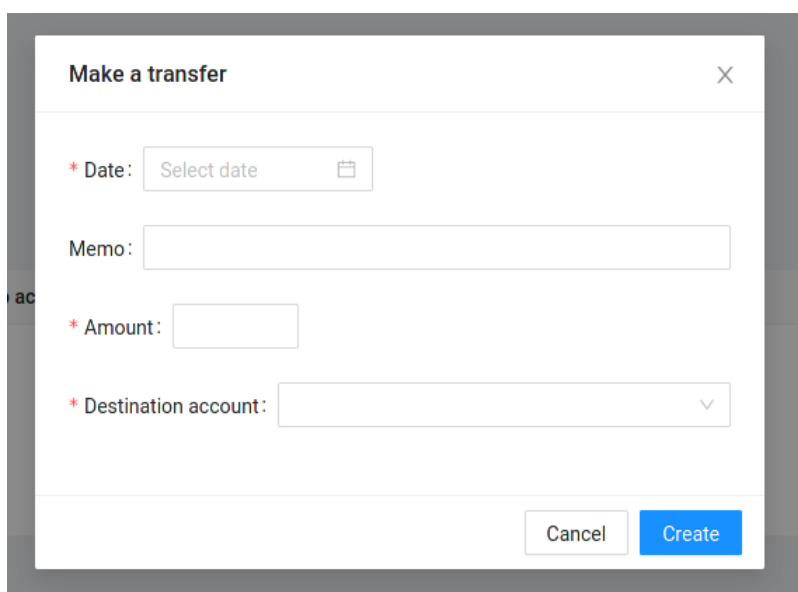


The dialog box is titled "Create a new transaction" and has a close button (X) in the top right corner. It contains the following fields:

- * Date: A date picker with the text "Select date" and a calendar icon.
- Memo: A text input field.
- * Amount (positive for income, negative for expense): A text input field.
- * payee: A text input field.
- * category: A dropdown menu with the text "Select a category" and a downward arrow.

At the bottom right, there are two buttons: "Cancel" and "Create".

Rysunek 2.5: Formularz dodawania nowej transakcji

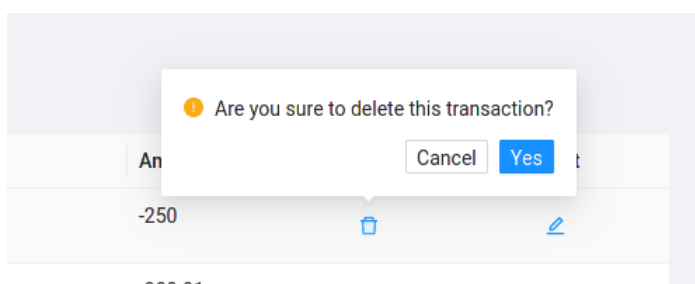


The dialog box is titled "Make a transfer" and has a close button (X) in the top right corner. It contains the following fields:

- * Date: A date picker with the text "Select date" and a calendar icon.
- Memo: A text input field.
- * Amount: A text input field.
- * Destination account: A dropdown menu with a downward arrow.

At the bottom right, there are two buttons: "Cancel" and "Create".

Rysunek 2.6: Formularz dodawania nowego transferu pomiędzy kontami



Rysunek 2.7: Potwierdzenie usunięcia operacji

Rysunek 2.8: Okienka z formularzem do edycji transakcji

2.4.5. Wyświetlanie budżetów

Naciśnięcie na przycisk "Budget" w panelu nawigacyjnym powoduje przejście do widoku budżetu. Początkowo wyświetlany jest budżet dla aktualnego miesiąca. W górnej części tej strony umieszczone są przyciski służące do zmieniania miesiąca. Widoczna jest tam też kwota, która nie została jeszcze rozdzielona w ramach budżetu i możemy ją rozdysponować pomiędzy kategorie. Poniższy kod pochodzący z kodu źródłowego aplikacji oblicza tę kwotę (pominięto nieistotne fragmenty):

```
/**
 * Calculate amount available to budget
 */
public int available() {
    Budget previousBudget = this.getPreviousBudget();
    int overSpentAmount = 0; // Amount overspent in all
        categories for previous budget. Should be negative
    int previousBudgetAvailable = 0; // Amount available to
        budget in previous budget

    if (previousBudget != null) {
        previousBudgetAvailable =
            previousBudget.available();
        overSpentAmount =
            previousBudget.getBudgetCategories()
```

```

        .stream()
        .map(BudgetCategory::available)
        .filter(i -> i < 0)
        .reduce(0, Integer::sum);
    }

    // Sum amount already budgeted this month
    int budgetedAmount = this.getBudgetCategories()
        .stream()
        .map(BudgetCategory::getAmount)
        .reduce(0, Integer::sum);

    // Incomes for this month
    int incomesSum = this.getBudgetCategories()
        .stream()
        .filter(x -> x.getCategory().getId() == 1)
        .getSpent();

    return previousBudgetAvailable + incomesSum +
        overSpentAmount - budgetedAmount;
}

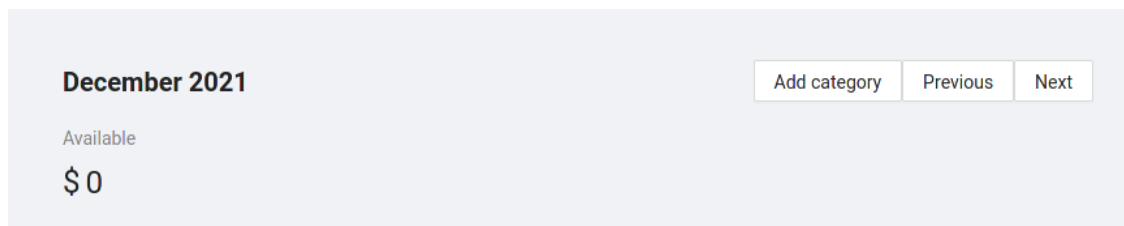
```

W budżecie dostępne mamy środki pochodzące z kategorii Incomes w danym miesiącu. Środki nie rozdzielone w ramach budżetu na dany miesiąc przechodzą na kolejny. Dodatkowo wszystkie wydatki ponad zaplanowane w poprzednim miesiącu są odejmowane od następnego budżetu. Budżet przedstawiony jest w postaci tabeli, która zawiera listę kategorii oraz kolumny z informacją o kwocie rzeczywiście wydanej w ramach tej kategorii, kwocie zaplanowanej do wydania oraz dostępnych nadal środkach. Jeśli ilość środków wydanych w ramach kategorii jest dodatnia to przychody w tej kategorii były większe niż wydatki. Jeśli ilość dostępnych środków jest ujemna to wydano więcej niż pozwala na to budżet. Naciśnięcie przycisku w kolumnie z zaplanowaną do wydania kwotą powoduje wyświetlenie formularza, w którym można tę kwotę zmienić. Środki dostępne w ramach kategorii liczone są przez następujący kod:

```

/**
 * Calculate amount available to spent in this budgeting
 * category
 */
public int available() {
    int previousMonthAvailable =
        Optional.ofNullable(this // May be null if this
            is first budget
        ).getBudget()

```



Rysunek 2.9: Nagłówek ekranu budżetu

```

        .getPreviousBudget()
        .map(Budget::getBudgetCategories)
        .orElse(List.of())
        .stream()
        .filter(x ->
            Objects.equals(x.getCategory().getId(),
                this.getCategory().getId()))
        .map(BudgetCategory::available)
        .findFirst()
        .orElse(0);

    // Ignore overspent categories
    if (previousMonthAvailable < 0) {
        previousMonthAvailable = 0;
    }

















    return previousMonthAvailable + this.amount +
        this.getSpent();
}

```

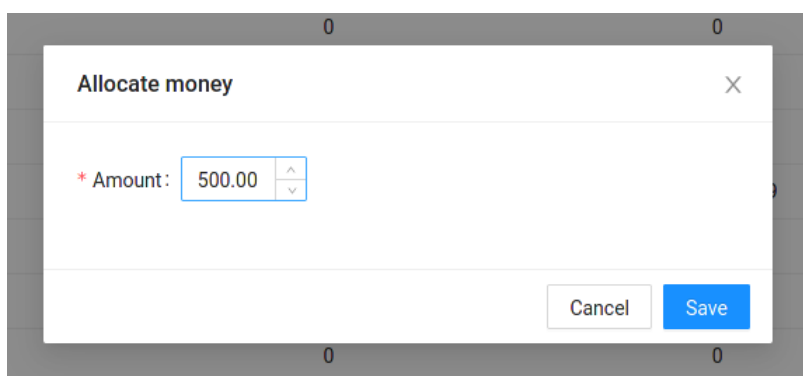
Wynika z niego, że środki, których nie wydamy w danym miesiącu przechodzą na kolejny miesiąc.

2.4.6. Zarządzanie kategoriami

Na ekranie budżetu możemy także zarządzać kategoriami budżetowymi. W nagłówku widoku budżetu znajduje się przycisk “Add category”, którego naciśnięcie powoduje wyświetlenie formularza do wpisania danych kategorii. W tabeli prezentującej stan budżetu dla każdej z kategorii dwie pierwsze kolumny zawierają przyciski do edycji i usuwania kategorii. Naciśnięcie przycisku edycji także powoduje wyświetlenie formularza. Usuwanie kategorii nie następuje od razu po naciśnięciu przycisku, gdyż należy jeszcze tę operację potwierdzić.

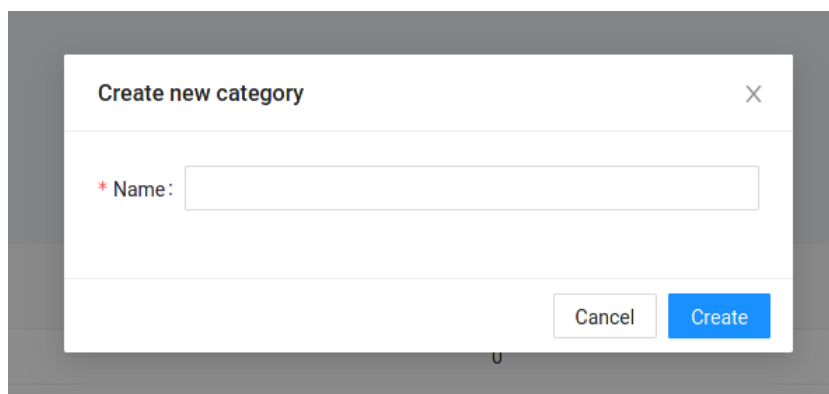
Edit category	Delete category	Category	Budgeted	Spent	Available
		Rent/Mortgage	2000	0	2000
		Electric Bills	500	-512.93	-12.93
		Water Bills	0	0	0
		Internet Bills	80	-79.99	0.01
		Cellphone Bills	25	-25	0
		Groceries	600	-218.35	381.65
		Eating Out	0	0	0
		Transportation	0	0	130.01

Rysunek 2.10: Widok budżetu w postaci tabeli



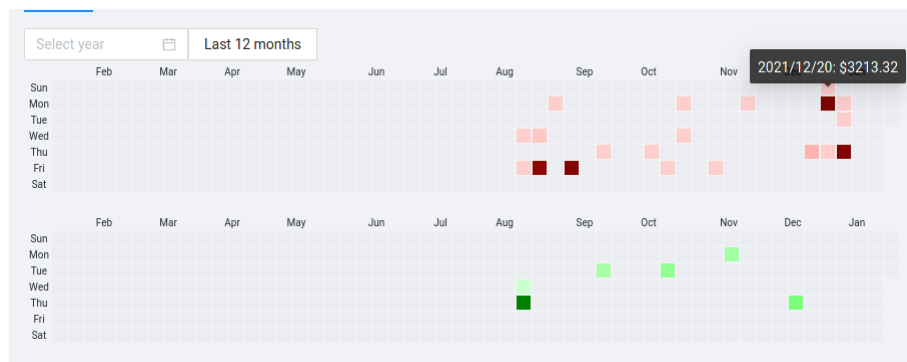
A dialog box titled "Allocate money" with a close button (X) in the top right corner. It contains a label "* Amount:" followed by a text input field containing "500.00" and a spinner control. At the bottom right, there are two buttons: "Cancel" and "Save".

Rysunek 2.11: Formularz planowania kwoty do wydania w ramach kategorii



A dialog box titled "Create new category" with a close button (X) in the top right corner. It contains a label "* Name:" followed by a text input field. At the bottom right, there are two buttons: "Cancel" and "Create".

Rysunek 2.12: Formularz tworzenia nowej kategorii budżetowej



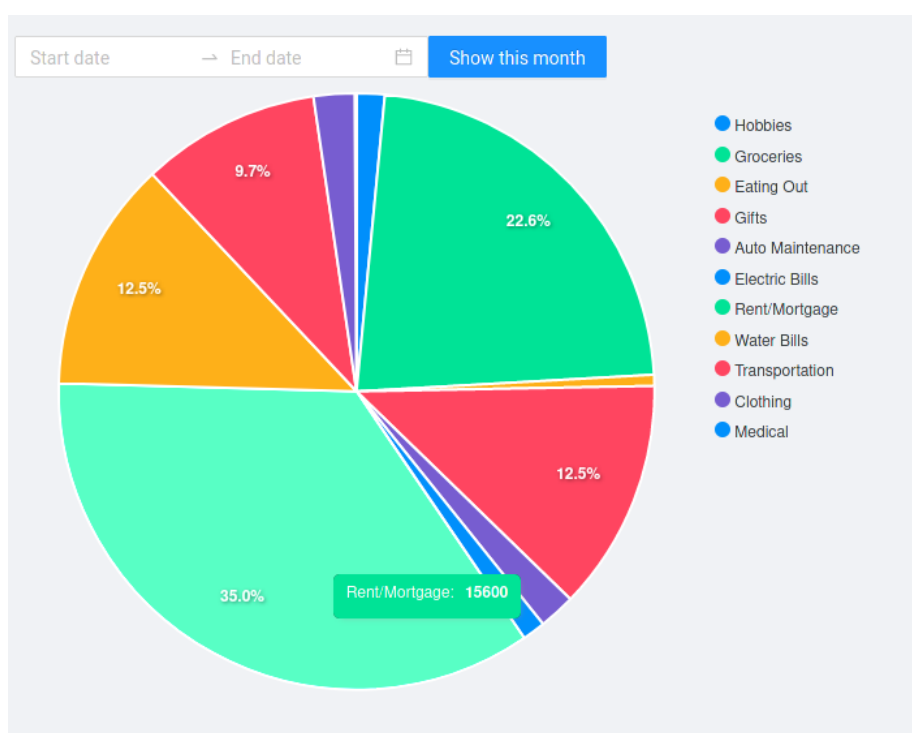
Rysunek 2.13: Raport sumy wydatków i przychodów dziennych

2.4.7. Wyświetlanie statystyk

Aplikacja udostępnia użytkownikom możliwość przeglądania raportów dotyczących budżetu przedstawionych w postaci diagramów. Strona raportów dostępna jest po wybraniu “Reports” z panelu nawigacyjnego. Różne rodzaje statystyk dostępne są poprzez wybór odpowiedniej zakładki.

W zakładce “Heat map” udostępniony jest raport dotyczący dziennych wydatków i przychodów. Dzięki niemu użytkownik ma możliwość szybkiego zobaczenia jak duże są wydatki lub przychody każdego dnia. Pokazywany jest on w postaci diagramu heatmap. Każdy kafelek reprezentuje jeden dzień, a jego kolor zależy od kwoty w danym dniu. Im ciemniejszy, tym jest ona większa. Domyślnie pokazywane są dane z ostatnich 12 miesięcy. Użytkownik może wyświetlić raport dotyczący wybranego przez siebie roku poprzez użycie formularza wyboru roku znajdującego się nad diagramami. Obok tego formularza znajduje się przycisk poprzez który można znowu wyświetlić dane dla ostatnich 12 miesięcy.

W zakładce “Expenses by category” udostępniony jest raport przedstawiający graficznie procentowy udział wydatków z każdej kategorii, który pomaga analizować na co przeznaczane są środki. Użytkownik ma możliwość wyboru zakresu dat z jakich mają być brane pod uwagę transakcje. Domyślnie pokazywany jest diagram dla transakcji z aktualnego miesiąca.



Rysunek 2.14: Raport wydatków przypadających na kategorie

Rozdział 3.

Użyte rozwiązania technologiczne

Program został podzielony na dwie części. Pierwsza z nich to część serwerowa. Odpowiada ona za logikę programu i przechowywanie danych w trwałej pamięci. Druga część to część kliencka. Jej głównym zadaniem jest prezentacja danych otrzymanych od części serwerowej użytkownikowi. Komunikacja między nimi odbywa się przez protokół HTTP. W dalszej części tego rozdziału dokładniej obie te części programu.

3.1. Część serwerowa

Część serwerowa programu została napisana z użyciem języka Java. W kodzie źródłowym możemy wydzielić trzy warstwy. Pierwsza z nich to warstwa dostępu do danych. Ta część programu komunikuje się z bazą danych i tłumaczy jej relacyjny system typów na architekturę obiektową języka Java. Druga warstwa to tzw. warstwa serwisowa. Udostępnia ona główną logikę działania programu i komunikuje się z jednej strony z warstwą dostępu do danych, aby zapisywać informacje w trwałej pamięci, a z drugiej strony udostępnia wyższej warstwie zbiór dostępnych do wykonywania w ramach logiki programu operacji. Tą wyższą warstwą jest warstwa kontrolerów. Kontrolery to obiekty wyspecjalizowane do odbierania żądań ze świata zewnętrznego. Korzystają z warstwy serwisów, aby zlecone żądania wykonać. Kontrolery udostępniają bezstanowy interfejs API dostępu do programu. Aplikacja stworzona jest z wykorzystaniem architektury Representational State Transfer (REST) [16], więc udostępnia zasoby (zasobem jest np. konto, transakcja, budżet), na których można wykonywać operacje zdefiniowane przez protokół HTTP, które powodują zmianę stanu tych zasobów.

3.1.1. Spring

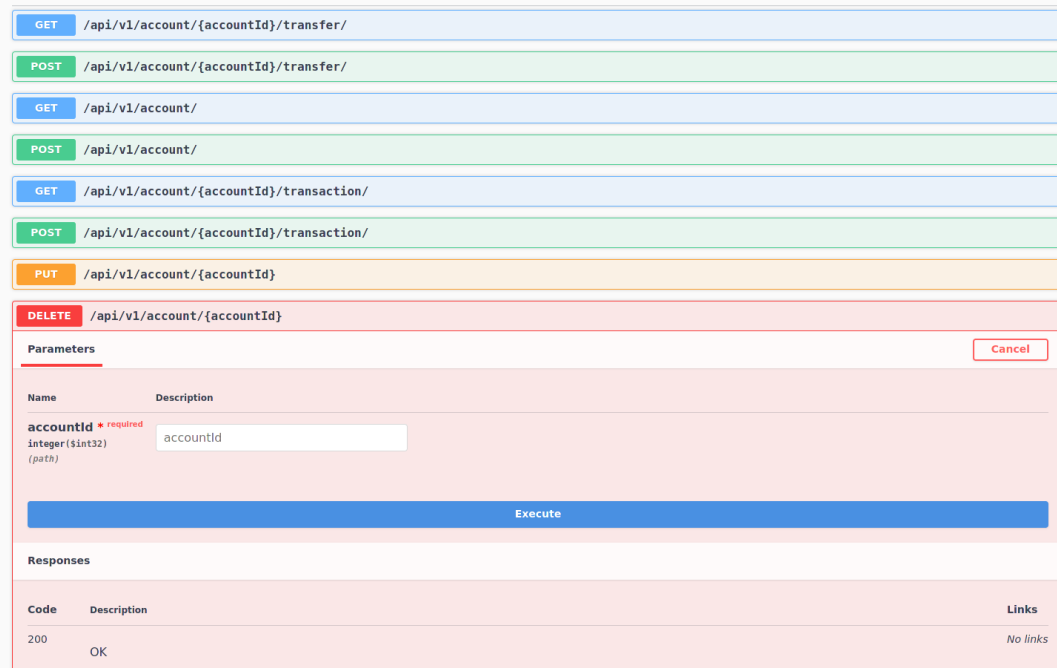
Główną technologią użytą w tworzeniu części serwerowej jest framework Spring [17]. Jest to obecnie najpopularniejszy framework do tworzenia aplikacji w języku Java [18]. Pod nazwą Spring kryje się tak naprawdę wiele różnych bibliotek. Cały framework podzielony jest więc na moduły rozwiązujące różne problemy [19] takie jak udostępnianie kontenera Inversion of Control (IoC) [20], tworzenie aplikacji internetowych, tworzenie aplikacji dla środowiska rozproszonego uruchamianych w chmurze, zabezpieczanie aplikacji, dostęp do baz danych i inne. Udostępnianie kontenera IoC jest jedną z głównych funkcji tego frameworka, a implementacja tego wzorca w postaci zasady Dependency Injection jest używana także w mojej aplikacji.

W moim rozwiązaniu użyłem między innymi modułu Spring MVC [21], który używany jest w warstwie prezentacji danych. Dzięki temu rozwiązaniu mogłem z łatwością odczytywać żądania przesyłane do kontrolerów w aplikacji z zewnątrz przez protokół HTTP. Spring MVC bazuje na technologii Servlet API i ułatwia definiowanie zasobów w architekturze REST. Wystarczy stworzyć metody i zdefiniować jakie adresy i metody HTTP mają one obsługiwać. Spring MVC zajmie się za nas niskopoziomowymi kwestiami obsługi żądań takimi jak odbieranie danych, tłumaczenie ich na obiekty języka Java i wysyłanie odpowiedzi.

Aby ułatwić sobie tworzenie aplikacji użyłem także modułu Spring Boot [22]. Spring Framework wymaga konfiguracji używanych elementów poprzez pliki XML, kod Java lub adnotacje w języku Java [23]. Twórcy tej biblioteki zauważyli, że większość programistów wybiera przy tworzeniu programów podobne opcje konfiguracji biblioteki Spring. Aby ułatwić z nią pracę, Spring Boot automatyzuje proces konfiguracji, zarówno elementów dotyczących frameworka Spring jak i innych bibliotek, udostępniając zdefiniowane wcześniej konfiguracje tych elementów. Pozwala to o wiele szybciej rozpocząć tworzenie programów nieprzejmując się pisaniem konfiguracji dopóki nie będziemy mieli żadnych specyficznych wymagań. Dzięki temu będziemy mieli mniej powtarzalnego kodu w naszej aplikacji.

Dokumentacja API REST

Dokumentacja interfejsu programistycznego w architekturze REST stworzona jest automatycznie w formacie OpenAPI [24] z użyciem narzędzia Swagger UI [25], które na podstawie wygenerowanej specyfikacji OpenAPI udostępnia interfejs graficzny. Możemy tam zobaczyć wszystkie udostępnianie zasoby, format danych wejściowych i wyjściowych, a także wykonać zapytania do interfejsu. Interfejs Swagger UI dostępny jest domyślnie pod adresem <http://localhost:8080/swagger-ui.html>, natomiast specyfikacja w formacie OpenAPI dostępna jest pod adresem <http://localhost:8080/v3/api-docs/>. Narzędzie te pozwalają w łatwy i szybki sposób udokumentować stworzony interfejs REST bez pisania specjalnego kodu, który by za to odpowiadał.

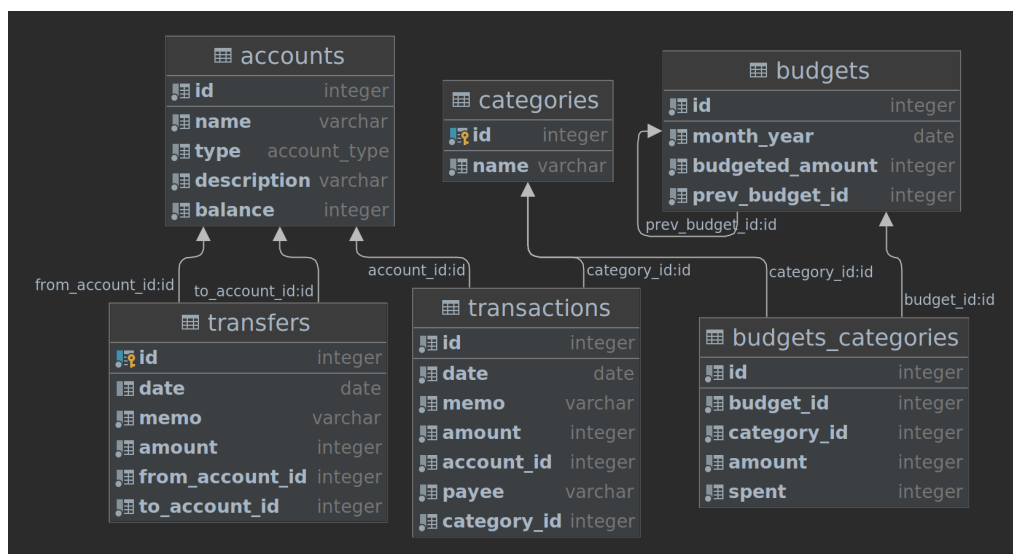


Rysunek 3.1: Dokumentacja API stworzona z użyciem Swagger UI

3.1.2. Schemat bazy danych

Do przechowywania danych użyłem relacyjnej bazy danych PostgreSQL [7]. Schemat bazy wygenerowany z użyciem IDE IntelliJ IDEA [26] widoczny jest poniżej (Rys. 3.2). Widzimy na nim tabele przechowujące dane kont, transakcji, transferów i kategorii. Miesięczne budżety przechowywane są w tabeli budgets. Każdy taki miesięczny budżet to kopia wszystkich kategorii wraz z przeznaczoną na nie kwotą i kwotą w ramach tej kategorii wydaną. Kategorie dla każdego budżetu przechowywane są w tabeli budgets_categories.

Warstwa dostępu do danych używa frameworków Spring Data [27] oraz Hibernate ORM [28]. Hibernate używany jest w celu tłumaczenia danych z relacyjnego systemu typów wewnątrz bazy danych do obiektowego systemu w języku Java i używa wzorca projektowego Unit of Work [29]. Pozwala on oznaczać klasy jako tzw. encje, które odpowiadają tabelom w bazie danych. Hibernate automatycznie generuje zapytania w języku SQL na podstawie języka DSL o nazwie HQL lub na podstawie kodu w języku Java. Na bazie tego frameworka stworzony jest kolejny używany w tej warstwie, czyli Spring Data JPA, który używa wzorca Repository. [30]. Idzie on jeszcze o krok dalej niż Hibernate i dostarcza gotowe interfejsy, które implementują podstawowe metody dostępu do danych, czyli CRUD od create, retrieve, update i delete. Te rozwiązania wyręczają nas w tworzeniu prostych zapytań SQL, które wygenerują za programistę. W moim rozwiązaniu używam takich wygenerowanych zapytań, ale też definiuję swoje tam, gdzie potrzeby aplikacji wykraczają poza opcje tych bibliotek.



Rysunek 3.2: Schemat bazy danych

3.1.3. Inne technologie użyte w części serwerowej

MapStruct

W aplikacji używam wzorca projektowego Data Transfer Object [31] (DTO), aby oddzielić dane udostępniane w API REST od reprezentacji danych w programie. Biblioteka MapStruct [32] używana jest w aplikacji, aby zautomatyzować proces tworzenia obiektów DTO, który jest żmudny i powtarzalny dla każdej klasy. MapStruct automatycznie generuje kod tłumaczący obiekty modelu aplikacji na obiekty transferu danych.

Lombok

Kod w języku Java uważany jest za zbyt rozwlekły w pewnych przypadkach. Programista musi pisać powtarzalny kod dla konstruktorów obiektów, metod dostępu i ustawiania danych (tzw. getters i setters) lub wzorca projektowego Builder [33]. Biblioteka Lombok stara się wyręczyć programistę w tych zadaniach poprzez automatyczną generację kodu. W mojej aplikacji używam przede wszystkim automatycznego tworzenia konstruktorów obiektów oraz metod tzw. getter i setter. Dzięki temu kod jest zwięzły i nie muszę zajmować się pisaniem takiego samego kodu dla różnych klas.

Hibernate Validator

W mojej aplikacji używam także biblioteki Hibernate Validator [34] do walidacji modelu danych, aby wszystkie wartości były poprawne. Pozwala ona zdefiniować

walidację ograniczeń dla danych w aplikacji za pomocą adnotacji w języku Java. Biblioteka ta udostępnia zbiór gotowych reguł takich jak ograniczenia dla minimalnych i maksymalnych wartości danych liczbowych, ograniczenie wartości daty do dat przeszłych, ograniczenie długości napisów i wiele innych. Wybrałem tę bibliotekę, gdyż oferuje takie często używane ograniczenia dla danych, dzięki czemu nie muszę ich po raz kolejny implementować. Hibernate Validator automatycznie sprawdzi, czy dane są poprawne. Kod aplikacji nie jest wtedy zaśmiecany walidacją i skupia się na logice biznesowej.

3.2. Część kliencka

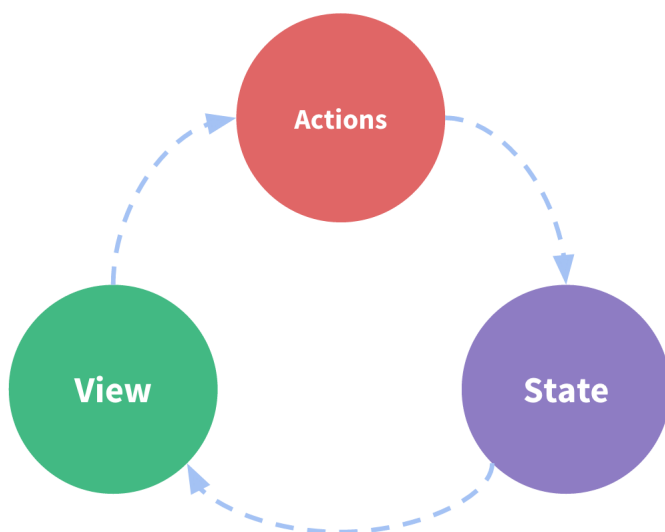
W odróżnieniu od części serwerowej, część kliencka została stworzona z użyciem języka JavaScript. Jej zadaniem jest komunikacja z częścią serwerową przy użyciu protokołu HTTP i prezentacja danych użytkownikowi. Została napisana przy użyciu biblioteki React [35], czyli popularnego rozwiązania do tworzenia interfejsu użytkownika.

3.2.1. Biblioteka React

React to biblioteka stworzona przez programistów Facebooka. Jest obecnie jedną z najchętniej wybieranych technologii do tworzenia interfejsów użytkownika [36], co przemawia nad jej wyborem z powodu mnogości dostępnych materiałów. Używanie tej technologii polega na podziale interfejsu na tak zwane komponenty. Są to podstawowe części tworzące interfejs, które są od siebie izolowane i łączone w większą całość oraz zarządzające swoim własnym stanem. Pozwala to tworzyć interaktywny interfejs strony, który zmienia się pod wpływem interakcji ze strony użytkownika bez potrzeby odświeżania całej strony. Efektywność tego rozwiązania jest polepszona dzięki użyciu wirtualnego DOM [37]. Wzorzec ten poprawia responsywność strony poprzez aktualizowanie tylko zmienionych elementów struktury DOM [38] zamiast wszystkich co jest tańszą operacją. Dzięki temu mogłem z łatwością stworzyć złożony i szybki interfejs myśląc o podstawowych elementach jakimi są komponenty. Niektóre z nich używane są w kilku miejscach dzięki czemu cały program składa się z mniejszej ilości kodu.

3.2.2. Redux

Wraz z rozwojem aplikacji interfejs stawał się coraz bardziej skomplikowany, a przy tym zarządzanie stanem komponentów stawało się coraz trudniejsze, gdyż ilość danych także rosła. Rozwiązaniem tego problemu jest użycie biblioteki Redux [39]. Używając tej technologii przechowujemy dane w jednym miejscu - obiekcie store. Komponenty interfejsu subskrybują zmiany w tym obiekcie i zmieniają się pod ich



Rysunek 3.3: Przepływ danych w architekturze używającej biblioteki Redux

wplywem. Przepływ danych następuj więc w jedną stronę od obiektu store do interfejsu. Stan przechowywanych danych zmieniany jest z użyciem czystych funkcji nazywanych w tym wzorcu reducers:

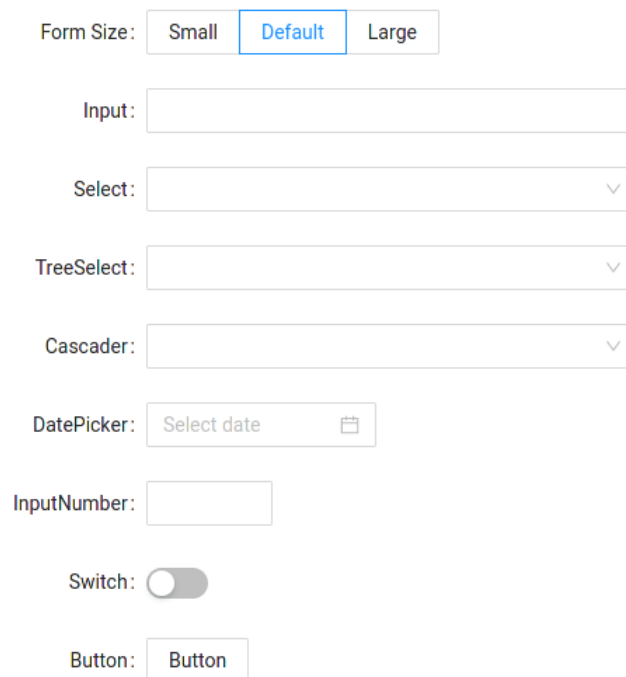
$$f(state, action) => newState$$

Takie funkcje nie mogą mieć efektów ubocznych ani zmieniać aktualnego stanu. Na podstawie danych wejściowych zwracają nowy stan powstały na bazie poprzedniego.

Kod części klienckiej podzielony jest według funkcjonalności, których obsługa znajdują się w różnych podkatalogach. Obiekt store także jest podzielony między różne pliki obsługujące dane różnych rodzajów. Komponenty interfejsu użytkownika zlecają wykonywanie akcji obiektowi store oraz odczytują i aktualizują przechowywane w nim dane. Biblioteka Redux zajmuje się także komunikacją z częścią serwerową pobierając dane do obiektu store oraz wysyłając zmiany danych, gdy zmienia się ich stan po stronie klienckiej.

3.2.3. Ant Design

Aby ułatwić sobie tworzenie dobrze wyglądającego interfejsu użytkownika, użyłem biblioteki gotowych komponentów Ant Design [40]. Udostępnia ona spójne wizualnie elementy interfejsu użytkownika, które zostały stworzone z zachowaniem najważniejszych zasad dziedziny User Experience (UX). Jako, że elementy te są komponentami biblioteki React, połączenie ich z aplikacją jest niezwykle łatwe.



The image shows a collection of form components from the Ant Design library. At the top, there's a 'Form Size' selector with three tabs: 'Small', 'Default' (which is highlighted with a blue border), and 'Large'. Below this, there are several input fields: an 'Input' field, a 'Select' dropdown menu, a 'TreeSelect' dropdown menu, and a 'Cascader' dropdown menu. Further down is a 'DatePicker' with a text input 'Select date' and a calendar icon. Below that is an 'InputNumber' field. Then there's a 'Switch' control, which is a toggle switch currently in the 'off' position. Finally, at the bottom, there's a 'Button' component with the text 'Button' inside.

Rysunek 3.4: Przykład gotowego komponentu z biblioteki Ant Design - formularz składający się z różnych komponentów do wprowadzania danych

3.2.4. Wykresy

Do stworzenia wykresów używanych na podstronie dotyczącej statystyk użyłem gotowych bibliotek. Jedną z nich jest biblioteka Apex Charts [41], która udostępnia zbiór wielu dopracowanych wizualnie i interaktywnych wykresów dla różnych frameworków front-endowych, w tym używanego przeze mnie React.

Aby stworzyć wykres typu heatmap użyłem kolejnej biblioteki React Heat Map [42], która stworzona została specjalnie do wyświetlania danych w postaci kalendarza, tzn. dla każdego dnia oddzielnie.



Rysunek 3.5: Różne rodzaje wykresów dostępne w bibliotece Apex Charts

Rozdział 4.

Podsumowanie

4.1. Możliwości dalszej rozbudowy

Pisząc aplikację starałem się, aby była ona otwarta na dalsze usprawnienia. Poniżej przedstawiam kilka możliwych sposobów na ulepszenie aplikacji.

4.1.1. Automatyczne dodawanie transakcji

Dodawanie wszystkich transakcji i transferów ręcznie jest żmudne. Można więc to poprawić poprzez automatyczne pobieranie transakcji z kont bankowych użytkownika. Wymagałoby to dostępu do raportów z bankowości elektronicznej lub użycia API udostępnianego przez różne instytucje finansowe. Aplikacja mogłaby też posiadać funkcję odczytywania danych z paragonów, aby użytkownik nie musiał wpisywać ich ręcznie, a także pozwalałaby je zapisywać, aby były od razu dostępne w historii operacji.

4.1.2. Konta użytkowników

Obecnie aplikacja przeznaczona jest dla jednego konta co jest dużym ograniczeniem. Dodanie kont użytkowników, którzy prowadziliby oddzielnie dla siebie budżety byłoby więc dużym usprawnieniem aplikacji. Wymagałoby to dodania mechanizmu autoryzacji i uwierzytelniania oraz bezpiecznego przechowywania danych użytkowników. Po zaimplementowaniu takiej funkcjonalności można by było też udostępnić bezpiecznie aplikację przez Internet.

4.1.3. Podział transakcji na kilka kategorii

Niektóre transakcje stanowią pojedynczą operację finansową, ale składają się z transakcji w wielu kategoriach, np. w ramach jednej zapłaty za produkty w sklepie

wydajemy pieniądze w różnych kategoriach. Dodanie opcji podziału transakcji na kilka kategorii usprawniłoby używanie aplikacji, gdyż historia operacji zapisanych w aplikacji lepiej odzwierciedlałaby rzeczywiste operacje.

4.2. Wnioski końcowe

Pomysł na aplikację wziął się także z moich prywatnych potrzeb, gdyż nie chciałem przechowywać informacji o budżecie na zewnętrznych serwerach. W ramach tej pracy udało mi się stworzyć aplikację z podstawowymi funkcjonalnościami, które są wystarczające, aby rozpocząć zarządzanie budżetem osobistym. Tworząc ten program miałem okazję poznać nowoczesne technologie z różnych dziedzin - zarówno używane po stronie serwera, jak i przeglądarki internetowej.

Bibliografia

- [1] Przykładowe strategie prowadzenia budżetu osobistego. <https://studentloanhero.com/featured/simple-budgeting-methods-for-financial-health/>.
- [2] Metody prowadzenia budżetu. <https://logicaldollar.com/budgeting-methods/>.
- [3] Program Visual Paradigm do tworzenia diagramów uml. <https://www.visual-paradigm.com/>.
- [4] Strona aplikacji You Need A Budget. <https://www.youneedabudget.com/>.
- [5] Cennik używania aplikacji You Need A Budget. <https://www.youneedabudget.com/pricing/>.
- [6] Strona aplikacji Mint. <https://mint.intuit.com/>.
- [7] Baza danych PostgreSQL. <https://www.postgresql.org/>.
- [8] Java. <https://java.com/>.
- [9] System kontroli wersji Git. <https://git-scm.com>.
- [10] Github. <https://github.com/>.
- [11] Gradle. <https://gradle.org/>.
- [12] Narzędzie createuser bazy danych PostgreSQL. <https://www.postgresql.org/docs/current/app-createuser.html>.
- [13] Narzędzie createdb bazy danych PostgreSQL. <https://www.postgresql.org/docs/current/app-createdb.html>.
- [14] Node.js. <https://nodejs.org/en/>.
- [15] NPM. <https://www.npmjs.com/>.
- [16] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

- [17] Framework Spring. <https://spring.io/>.
- [18] Raport popularności narzędzi używanych w ekosystemie tworzenia aplikacji w języku Java. <https://snyk.io/jvm-ecosystem-report-2021/>.
- [19] Moduły frameworka Spring. <https://spring.io/projects>.
- [20] Wzorzec Inversion of Control i zasada Dependency Injection. <https://www.martinfowler.com/articles/injection.html>.
- [21] Dokumentacja Spring MVC. <https://docs.spring.io/spring-framework/docs/current/reference/html/web.html>.
- [22] Spring Boot. <https://spring.io/projects/spring-boot>.
- [23] Adnotacje w języku Java. <https://docs.oracle.com/javase/tutorial/java/annotations/>.
- [24] Specyfikacja OpenAPI. <https://swagger.io/specification/>.
- [25] Narzędzie Swagger UI. <https://swagger.io/tools/swagger-ui/>.
- [26] Środowisko programistyczne IntelliJ IDEA. <https://www.jetbrains.com/idea/>.
- [27] Spring Data JPA. <https://spring.io/projects/spring-data-jpa>.
- [28] Hibernate ORM. <https://hibernate.org/orm/>.
- [29] Wzorzec projektowy Unit of Work. <https://martinfowler.com/eaCatalog/unitOfWork.html>.
- [30] Wzorzec projektowy Repository. <https://www.martinfowler.com/eaCatalog/repository.html>.
- [31] Wzorzec projektowe Data Transfer Object. <https://martinfowler.com/eaCatalog/dataTransferObject.html>.
- [32] Biblioteka MapStruct. <https://mapstruct.org/>.
- [33] Wzorzec projektowy Builder. <https://refactoring.guru/design-patterns/builder>.
- [34] Biblioteka Hibernate Validator. <https://hibernate.org/validator/>.
- [35] Biblioteka ReactJS. <https://reactjs.org/>.
- [36] Najpopularniejsze technologie do tworzenia interfejsu użytkownika w języku Javascript. https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/front_end_frameworks_experience_ranking.
- [37] Wirtualny DOM. <https://reactjs.org/docs/faq-internals.html>.

- [38] Document Object Model. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model.
- [39] Biblioteka Redux. <https://redux.js.org/>.
- [40] Ant Design. <https://ant.design/>.
- [41] Biblioteka wykresów Apex Charts. <https://apexcharts.com/>.
- [42] Biblioteka React Heat Map. <https://uiwjs.github.io/react-heat-map/>.