Rok akademicki 2013/2014

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

PRACA DYPLOMOWA INŻYNIERSKA

Piotr Hryciuk

Aplikacja internetowa do zarządzania finansami osobistymi przy użyciu szkieletów Spring oraz AngularJS

Opiekun pracy
dr inż. Łukasz Skonieczny

Ocena: .....

.....

Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego

Kierunek: Informatyka

Specjalność: Inżynieria Systemów Informatycznych

Data urodzenia: 1991.07.27

Data rozpoczęcia studiów: 2010.10.01

Życiorys

Urodziłem się 27 lipca 1991 roku w Łosicach. W 2010 roku ukończyłem Liceum Ogólnokształcące nr 1 w Łosicach. W październiku 2010 roku rozpocząłem studia na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. Dotychczas pracowałem jako programista w trzech firmach: Citi Bank Handlowy, Source International Corp. oraz Freeport Metrics, gdzie obecnie jestem zatrudniony.

.....
Podpis studenta

EGZAMIN DYPLOMOWY

Złożył egzamin dyplomowy w dniu ..... 20__ r

z wynikiem .....

Ogólny wynik studiów: .....

Dodatkowe wnioski i uwagi Komisji: .....

.....

.....

STRESZCZENIE

Praca przedstawia aplikację internetową, służącą do zarządzania finansami osobistymi, wspierającą budżetowanie środków oraz raportowanie. W początkowej części pracy uwaga poświęcona jest wymaganiom jakim oprogramowanie musi sprostać. Kolejna część opisuje technologie wybrane do implementacji. Najważniejszym rozdziałem pracy jest projekt tworzonego oprogramowania. Ostatni rozdział poświęcony jest opisowi narzędzi programistycznych wykorzystanych w implementacji.

Słowa kluczowe:
finanse osobiste, zarządzanie, aplikacja internetowa, spring, angularjs, java

WEB APPLICATION FOR MANAGING PERSONAL FINANCES DEVELOPED WITH USE OF SPRING, AND ANGULARJS FRAMEWORKS

Thesis presents web application, for managing personal finances, that supports budgeting and periodical reports. In the first part of the paper, main focus was put on software requirements, that the application should handle. Then, technologies used for implementation are described. The most significant part of the thesis is the chapter about the design of an application. Last chapter describes developer tools used during implementation process.

Keywords:
personal finances, managing, web application, spring, angularjs, java

Spis treści

1. Wstęp

1.1 Motywacja - zarządzanie pieniędzmi

Pieniądz jest powszechnie akceptowanym środkiem, za pomocą którego dokonujemy płatności za dostarczone dobra lub wywiązujemy się ze zobowiązań.[1] Wpływa na niemal wszystkie sfery życia człowieka.

Rozważne i przemyślane zarządzanie finansami, nawet przy niewielkich dochodach, może znacznie podnieść komfort życia. Jednym z efektywnych systemów zarządzania finansami osobistymi jest budżetowanie. Konsekwencja i systematyczność w gospodarowaniu budżetem pozwala na realizację długookresowych celów wymagających znacznych nakładów pieniężnych, niemożliwych do realizacji przy braku świadomego kontrolowania swoich wydatków i wpływów.

1.2 Aplikacja internetowa

Aplikacja internetowa to program komputerowy pracujący na serwerze. Komunikuje się poprzez sieć komputerową z hostem użytkownika przy użyciu przeglądarki internetowej, będącej klientem aplikacji.

Podstawową zaletą tworzenia oprogramowania tego typu jest fakt, że aplikacje internetowe są kompatybilne wieloplatformowo, gdyż nie zależą od systemu operacyjnego. Co więcej, dostawca oprogramowania może z łatwością przeprowadzać aktualizacje oprogramowania bez jakiejkolwiek ingerencji użytkownika, a jedynym wymaganiem do sprawnego korzystania z programu jest posiadanie zainstalowanej przeglądarki internetowej. Jako zaletę należy także uznać fakt, że użytkownik ma dostęp do swoich danych nawet w przypadku awarii własnego urządzenia, gdyż dane są gromadzone na serwerach dostawcy oprogramowania. Aplikacje internetowe są dostępne właściwie wszędzie. Można z nich korzystać nie tylko używając komputerów stacjonarnych czy laptopów, ale także tabletów, smartfonów czy urządzeń elektronicznych noszonych jako część garderoby.

Pierwszą wadą jaką należy wymienić jest całkowita zależność od serwera przez co użytkownik nie ma możliwości dokonywania operacji bez połączenia z siecią. Jednak

[1] D. Begg, S. Fischer, R. Dornbusch, Makroekonomia, wyd. PWE, Warszawa 2003, s. 94

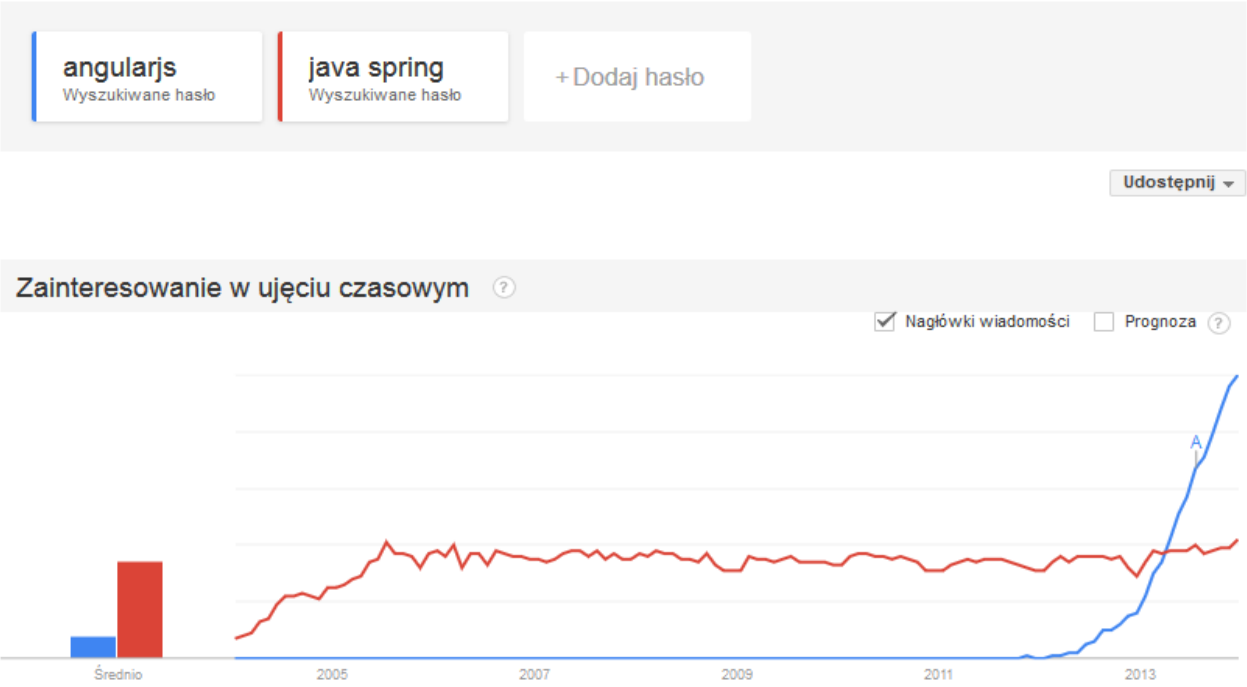
aplikacje internetowe są krytykowane przez sceptyków głównie ze względu na inne czynniki: kwestie prywatności oraz niewielką ilość programów z otwartym kodem źródłowym. Pierwszy z nich dotyczy danych, które użytkownik udostępnia dostawcy. Często są one wykorzystywane do celów marketingowych lub sprzedawane innym firmom. Ponadto, dostawca oprogramowania jest w stanie śledzić niemal każdy ruch użytkownika w trakcie jego korzystania z aplikacji. Jest to tematem gorących dyskusji w przypadku korzystania z portali społecznościowych, bowiem na podstawie aktywności w aplikacji łatwo można dociec wielu szczegółów z prywatnego życia użytkownika. Drugi czynnik jest poniekąd powiązany. Bardzo mała część najpopularniejszych aplikacji internetowych udostępnia kod źródłowy, przez co tak naprawdę nie wiemy jakie dane zostawiamy na serwerach i co się z nimi dzieje.

Aplikacje internetowe są obecnie najpopularniejszym typem programów komputerowych. Służą one nie tylko do rozrywki, ale także jako portale korporacyjne czy systemy obsługi klientów. Obecnie najbardziej rozpoznawalne aplikacje internetowe to Facebook, Gmail oraz Twitter.

1.3 Cel projektowania aplikacji

W dzisiejszych czasach aplikacje webowe ułatwiają zarządzanie pocztą elektroniczną, pozwalają robić zakupy czy też opłacać rachunki. Celem projektowanej przeze mnie aplikacji będzie ułatwienie kontroli finansów domowych, poprzez zapewnienie wsparcia dla budżetowania pieniędzmi. Dzięki niemu użytkownik będzie wiedział dokładnie kiedy będzie mógł sobie pozwolić na jaki wydatek, ustanawiać cele finansowe czy też obierać działania mające na celu wyjście z zadłużenia. Oprócz budżetowania pozwoli ona na obserwowanie stanu konta, kategoryzację wydatków oraz przegląd raportów z danego okresu z przeszłości. Podział na kategorie pozwoli łatwo rozpoznać typ wydatków, który najbardziej uszczupla budżet. Ponadto, uzmysłowi użytkownikowi ile faktycznie wydaje na poszczególny typ produktu czy usługi.

W osiągnięciu celu pomoże integracja nowoczesnych technologii programistycznych tj. szkieletu języka Java, Spring oraz szkieletu języka JavaScript, AngularJS.



Rysunek 1. Zainteresowanie hasłami "java spring" oraz "angularjs" na przestrzeni czasu, Źródło: www.google.pl/trends, Data dostępu 21.12.2013

1.4 Podział pracy

Rozdział drugi przedstawia założone wymagania stawiane aplikacji do zarządzania finansami osobistymi. W rozdziale trzecim znajduje się opis technologii wybranych do implementacji. Rozdział czwarty ma za zadanie przedstawić projekt oprogramowania. Celem rozdziału piątego jest wyliczenie i krótkie omówienie użytych narzędzi programistycznych. Ostatni rozdział podsumowuje całość prac projektowych i implementacyjnych. Na końcu pracy znaleźć można spis wykorzystanych rysunków i publikacji.

2. Specyfikacja wymagań

Niniejszy rozdział przedstawia wymagania stawiane aplikacji służącej do zarządzania finansami osobistymi. Omawiane są kolejno wymagania biznesowe, wymagania funkcjonalne użytkownika, wymagania niefunkcjonalne - użytkownika i systemowe. Lista reguł biznesowych stanowi zwieńczenie rozdziału.

2.1 Wymagania biznesowe

Nazwa oprogramowania to Money. Aplikacja ma za zadanie uproszczenie zarządzania finansami osobistymi poprzez metodę budżetowania oraz generowanie raportów pozwalających na głębszą analizę. Celem projektu jest dostarczenie użytkownikom możliwości rejestrowania swoich wydatków oraz budżetowania pieniędzy w okresach miesięcznych. Użytkownik powinien mieć także możliwość przeglądania wizualizacji odzwierciedlających wydatki w wybranym przedziale czasowym. Język angielski jest językiem interfejsu użytkownika.

2.2 Dziedzinowy słownik pojęć

Użytkownik systemu - osoba korzystająca z aplikacji do zarządzania finansami osobistymi Money

Kategoria (ang. category) - określa rodzaj wydatku; jest powiązana z użytkownikiem

Podkategoria (ang. subcategory) - określa bardziej szczegółowy rodzaj wydatków podrzędny do Kategorii

Transakcja (ang. transaction) - pojedynczy wydatek finansowy z określoną nazwą, datą, podkategorią oraz kwotą.

Budżet (ang. budget) - Zbiór transakcji związanych z użytkownikiem i określający jego stan finansowy

Kwota zabudżetowana (ang. budgeted amount) - Kwota przypisana do danej podkategorii, określająca planowaną maksymalną sumę wydatków w danym miesiącu

Kwota wydana (ang. spent) - Suma wydatków danej podkategorii lub kategorii określająca sumę wydatków.

2.3 Wymagania funkcjonalne użytkownika

W1.
Użytkownik może uwierzytelnić się przy pomocy adresu e-mail oraz hasła. Przy procesie rejestracji podawane jest także imię oraz nazwisko użytkownika.

W2.
Hasło jest zapisywane w bazie danych przy pomocy funkcji skrótu MD5.

W3.
Użytkownik systemu może zmienić swoje hasło pod warunkiem poprawnego, dwukrotnego wpisania starego hasła.

W4.
Każdy użytkownik systemu ma przypisane domyślne kategorie i powiązane podkategorie. Kategoria ma określoną nazwę.
Użytkownik systemu ma możliwość:

- dodania nowej kategorii powiązanej z kontem,
- usunięcia kategorii powiązanej z kontem,
- edycji nazwy kategorii powiązanej z kontem.

W5.
Z każdą kategorią jest powiązana lista podkategorii. Podkategoria ma określoną nazwę.
Każdy użytkownik systemu ma możliwość:

- dodania nowej podkategorii powiązanej z istniejącą kategorią,
- edycji istniejącej podkategorii,
- usunięcia istniejącej podkategorii.

W6.

Każdy użytkownik systemu może dodać nową transakcję. Dane dostarczane z każdą transakcją to:

- nazwa transakcji,
- data transakcji,
- nazwa podkategorii,
- kwota.

Dane wprowadzane w formularzu podlegają walidacji tj. kwota musi być liczbą, data musi być wprowadzona w odpowiednim formacie.

W7.

Każdy użytkownik systemu może na wprowadzonych przez siebie transakcjach przeprowadzać następujące operacje:

- zmiana nazwy,
- zmiana kwoty,
- zmiana daty,
- zmiana podkategorii transakcji.

W8.

Każdy użytkownik systemu ma możliwość przypisania kwoty zabudżetowanej dla przypisanych do siebie podkategorii.

W9.

Kwota zabudżetowana podkategorii, jest sumowana i przedstawiana jako kwota zabudżetowana dla kategorii.

W10.

Każdy użytkownik systemu ma możliwość przeglądania historii kwot zabudżetowanych dla podkategorii i kategorii.

W11.

Kwotę zabudżetowaną można przypisywać dla miesięcy z przyszłości.

W12.

Każdy użytkownik systemu może przeglądać raporty podsumowujące ogólną sumę wydatków dla różnych okresów. Możliwe okresy podsumowania to:

- tydzień,
- miesiąc,
- rok.

Raporty zwizualizują sumę wydatków dla wybranego okresu z wyszczególnieniem odpowiednio:

- dla tygodnia - dni,
- dla miesiąca - dni,
- dla roku - miesięcy.

Dodatkowo dla wybranego okresu przedstawiona zostanie również wizualizacja ukazująca dla tygodni i miesięcy podział wydatków ze względu na podkategorie, a dla roku podział wydatków ze względu na kategorie.

W13.

Każdy użytkownik systemu może przeglądać raporty podsumowujące wydatki z uwzględnieniem wybranej przez użytkownika systemu kategorii dla różnych okresów. Możliwe okresy podsumowania to:

- tydzień,
- miesiąc,
- rok.

Raporty zwizualizują sumę wydatków dla wybranego okresu z wyszczególnieniem odpowiednio:

- dla tygodnia - dni,
- dla miesiąca - dni,
- dla roku - miesięcy.

Dodatkowo dla wybranego okresu przedstawiona zostanie również wizualizacja ukazująca podział wydatków ze względu na podkategorie przypisane do wybranej przez użytkownika systemu kategorii.

W14.

Każdy użytkownik systemu może przeglądać raporty podsumowujące stosunek kwoty zabudżetowanej do kwoty wydanej dla miesiąca.

Raporty zwizualizują sumę kwot wydanych oraz zabudżetowanych dla każdej kategorii w wybranym miesiącu.

Dodatkowo przedstawiona zostanie wizualizacja stosunku kwot wydanych oraz zabudżetowanych dla podkategorii powiązanych do wybranej przez użytkownika kategorii.

2.4 Wymagania niefunkcjonalne użytkownika

WNU1.

Aplikacja powinna zachowywać się poprawnie, gdy użytkownik korzysta z jednej z wymienionych poniżej przeglądarek:

- Mozilla Firefox, w wersji 26.0 lub wyższej,
- Google Chrome, w wersji 31.0.

2.5 Wymagania niefunkcjonalne systemowe

WNS1.

Aplikacja powinna korzystać z dostępnych darmowych rozwiązań systemów informatycznych.

WNS2.

Aplikacja powinna być wieloplatformowa tj. niezwiązana ściśle z wyłącznie jednym systemem operacyjnym.

WNS3.

Aplikacja powinna poprawnie uruchamiać się na dowolnym kontenerze aplikacyjnym, który implementuje specyfikację Servlet 3.0.

2.6 Reguły biznesowe

RB1.

Do użytkownika systemu w momencie stworzenia konta, przypisane są 3 domyślne kategorie wraz z podkategoriami:

- Daily expenses,
    - Bakery,
    - Dairy,
    - Meat,
    - Alcohol,
    - Restaurants,
    - Vegetables,
    - Fruits,
    - Candies
- Periodic expenses,
    - Bills,
    - Public transport,
    - Rent,
    - Credit,
    - Insurance
- Occasional,
    - Christmas,
    - Birthdays,
    - Valentine’s Day.

RB2.

Jeżeli czas od ostatniego użycia aplikacji nie jest dłuższy niż 10 minut i użytkownik nie wylogował się z aplikacji nie jest wymagane ponowne logowanie się.

RB3.

Jeżeli czas od ostatniego użycia aplikacji jest dłuższy niż 10 minut wymagane jest ponowne zalogowanie się użytkownika.

RB4.

Usunięcie kategorii wiąże się z usunięciem wszystkich powiązanych do niej podkategorii oraz transakcji.

RB5.

Usunięcie podkategorii wiąże się z usunięciem wszystkich powiązanych do niej transakcji.

RB6.

Nie ma limitu powiązanych z użytkownikiem kategorii.

RB7.

Nie ma limitu powiązanych z kategorią podkategorii.

RB8.

Usunięcie transakcji, podkategorii lub kategorii jest czynnością nieodwracalną.

RB9.

Osoba niezalogowana lub nieposiadająca odpowiednich uprawnień nie może wywoływać funkcji zwracających danych z systemu.

RB10.

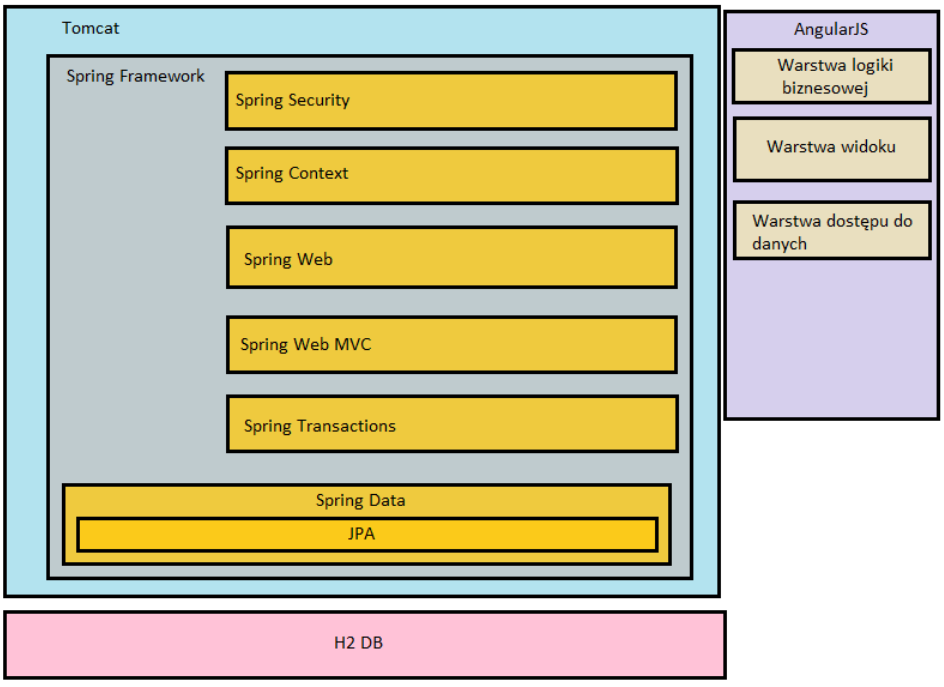
Osoba niezalogowana lub nieposiadająca odpowiednich uprawnień w chwili próby dostania się do widoku aplikacji innego niż logowanie lub rejestracja powinna zostać przekierowana na stronę główną aplikacji.

3. Użyte technologie

Decyzja o użyciu przedstawionych poniżej technologii w implementacji aplikacji padła głównie ze względu na zainteresowania autora. Dodatkowym czynnikiem determinującym wybór technologii była licencja określająca sposoby możliwej dystrybucji produktów. Autorowi zależało na tym aby używać rozwiązań:

- ogólnodostępnych,
- popularnych,
- darmowych,
- o otwartym kodzie źródłowym,
- mających społeczność wzajemnie sobie pomagających użytkowników.

Wyżej wymienione cechy gwarantują szybkie rozwiązywanie ewentualnych problemów. Co więcej produkty tego typu są aktywnie i dynamicznie rozwijane co dobrze rokuje na dalszy rozwój aplikacji. Poniżej znajduje się schemat architektury systemu wraz z technologiami realizującymi poszczególne warstwy aplikacji.



Rysunek 2. Schemat architektury aplikacji Money

3.1 Relacyjna baza danych H2

H2 to relacyjny system zarządzania bazą danych napisany w całości w języku Java. Kod bazy jest otwarty. Wspiera interfejs programistyczny JDBC API. Co ciekawe, udostępnia także aplikację do zarządzania systemem bazodanowym możliwą do uruchomienia w przeglądarce internetowej. Charakteryzuje się małym rozmiarem – plik *.jar zajmuje zaledwie 1.5 MB.

H2 jest bazą danych wykorzystaną przez aplikację Money. Produkt ten jest ciągle rozwijany, posiada dość rzadko spotykane możliwości i jest dostępny bez jakichkolwiek opłat.

	H2	Derby	HSQLDB	MySQL	PostgreSQL
Pure Java	Yes	Yes	Yes	No	No
Memory Mode	Yes	Yes	Yes	No	No
Encrypted Database	Yes	Yes	Yes	No	No
ODBC Driver	Yes	No	No	Yes	Yes
Fulltext Search	Yes	No	No	Yes	Yes
Multi Version Concurrency	Yes	No	Yes	Yes	Yes
Footprint (jar/dll size)	~1 MB	~2 MB	~1 MB	~4 MB	~6 MB

Rysunek 3. Porównanie popularnych baz danych w aplikacjach Java. Źródło: http://www.h2database.com/html/main.html z dnia 17.01.2014

3.2 Mapowanie relacyjno-obiektowe przy użyciu JPA

3.2.1 Mapowanie relacyjno-obiektowe

Relacyjne bazy danych działają na wartościach skalarnych rozmieszczonych w tabelach. Z kolei w programowaniu obiektowym operacje wykonywane są na obiektach, które najczęściej posiadają nieskalarną, złożoną strukturę. Zatem nie możemy ich opisać przy użyciu pojedynczego łańcucha znaków bądź liczby.

Z uwagi na powyższy fakt powstała potrzeba powstania sprawnego mechanizmu automatyzującego proces mapowania obiektów na tabele i tabel na obiekty. W rezultacie powstało wiele bibliotek, które implementują powyższe założenia jednocześnie pomagając unikać typowych błędów oraz powtarzającego się kodu.

Należy zaznaczyć, że mapowanie relacyjno-obiektowe powoduje spadek wydajności aplikacji, toteż należy używać tego mechanizmu w sposób rozsądny.

3.2.2 JPA – Java Persistence API

JPA to skrót oznaczający Java Persistence API. Jest to oficjalny standard mapowania relacyjno-obiektowego na platformie Java. Daje programistom możliwość operowania na encjach (ang. entity). JPA komunikuje się z bazą danych dzięki interfejsowi `EntityManager` (zarządca encji). Sposób w jaki obiekty i ich relacje są tłumaczone na elementy bazy danych definiowane jest przy pomocy adnotacji Java lub plików XML. [2]

Oprócz operacji określonych przez interfejs `EntityManager`, standard definiuje także język zapytań JPQL (Java Persistance API Query Language). Składnia języka przypomina SQL (Structured Query Language) jednak język zorientowany jest obiektowo.

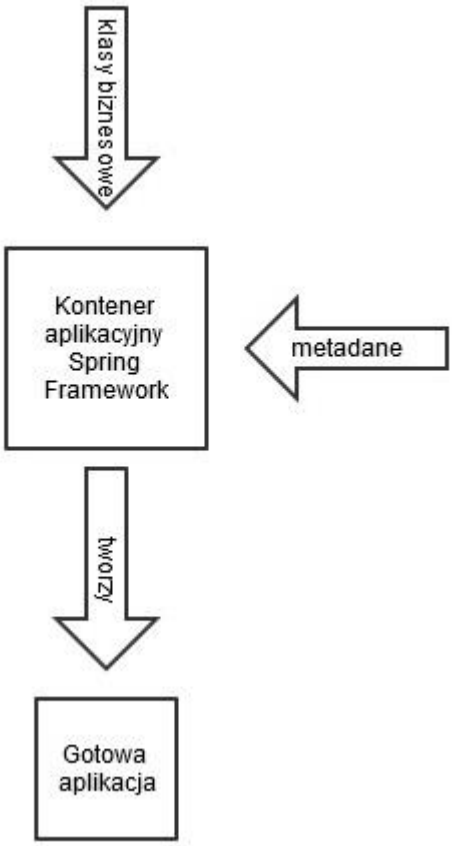
Standard JPA jest implementowany przez wiele różnych produktów. Najbardziej popularne to: Hibernate, OpenJPA oraz EclipseLink. Najnowsza wersja standardu JPA to 2.1.

[2] Specyfikacja JPA 2.1: https://jcp.org/en/jsr/detail?id=338

3.3 Spring Framework

Spring to szkielet aplikacyjny platformy Java. Szkielet ten powstał jako projekt z otwartym kodem źródłowym w październiku 2002 roku, kiedy to Rod Johnson w swojej książce pod tytułem „*Expert One-On-One J2EE Design*” opublikował propozycje usprawnienia mechanizmów Javy. Projekt ten otrzymał nazwę Spring Framework [1]. Jest on efektem wielu sprawdzonych i dobrze przetestowanych rozwiązań znacznie usprawniających prace implementacyjne nad programem komputerowym. Szkielet ten może być używany zarówno do budowania aplikacji internetowych jak i tzw. aplikacji biurkowych.[8]

Inne podobne produkty do Spring Framework to np. Google Guice, czy Enterprise JavaBeans 3.1.



Rysunek 4. Schemat działania szkieletu Spring Framework

3.4 AngularJS

AngularJS to szkielet aplikacyjny języka JavaScript wspierany i rozwijany przez firmę Google. Kod źródłowy szkieletu jest otwarty i rozwijany przez społeczność zgromadzoną wokół projektu. Szkielet ten wspiera model MVC, co w rezultacie przekłada się na łatwiejsze zarządzanie kodem oraz testowanie.

Twórcy przyjęli założenie, że programowanie deklaratywne dobrze nadaje się do budowania interfejsu użytkownika, z kolei programowanie imperatywne do wyrażania logiki [4]. AngularJS w swojej składni stał się adaptacją i rozszerzeniem języka HTML zwięźle i czytelnie zamieniającym stronę internetową w dynamiczną.

Jako najważniejsze cechy tego szkieletu należy wymienić wstrzykiwanie zależności oraz bindowanie danych. Pozwalają one uniknąć powtarzającego się i mało znaczącego kodu. Wstrzykiwanie zależności to wzorzec projektowy i wzorzec architektury oprogramowania pozwalający pozbycia się ustalonych zależności i zmiany ich w trakcie wykonywania programu lub w trakcie kompilacji. Obustronne bindowanie danych z kolei pozwala na automatyczne synchronizowanie widoków i modeli. Przy użyciu dyrektyw dostarczonych przez szkielet, a zagnieżdżonych w znacznikach HTML, użytkownik jest w stanie w łatwy sposób kontrolować zmiany stanu kontrolek interfejsu użytkownika, iterować po kolekcjach, oznaczać i odwoływać się do modelu aplikacji itp. Pozwalają one na tworzenie znaczników HTML do wielokrotnego użytku, które wpływają na zachowanie oznaczonych elementów. Użytkownik może także samodzielnie zdefiniować własne dyrektywy.

Do najważniejszych dyrektyw dostarczonych przez AngularJS należą [3]:

- **ng-app** - deklaruje element jako główny element aplikacji. Oznacza także zezwolenie na zmianę zachowania elementów poprzez własne tagi HTML.
- **ng-bind** - Zmienia automatycznie tekst elementu HTML na wartość danego wyrażenia.
- **ng-model** - Podobny do *ng-bind*, ale pozwala na obustronne bindowanie danych pomiędzy widokiem a zakresem działania kodu (ang. scope).
- **ng-class** - Pozwala na to, by atrybuty klasy były ładowane dynamicznie.

- **ng-controller** - Określa klasę kontrolera JavaScript która zajmuje się przetwarzaniem danych zwracanych do HTML.
- **ng-repeat** - Tworzy instancję dla każdego elementu z kolekcji
- **ng-show & ng-hide** - Warunkowo pokazuje lub ukrywa element w zależności od wartości zmiennej boolowskiej.
- **ng-switch** - Warunkowo tworzy instancję jednego szablonu ze zbioru w zależności od wartości wyrażenia decydującego.
- **ng-view** - Bazowa dyrektywa odpowiedzialna za obsługę ścieżek (ang. routes), które rozwikłują dane JSON przed wyrenderowaniem szablonów obsługiwanych przez określone kontrolery.
- **ng-if** - Dyrektywa obsługująca wyrażenia typu if, które pozwalają na pokazywanie danego elementu, jeżeli warunki są spełnione.

3.5 Wzorzec MVC – Model-Widok-Kontroler

Wzorzec MVC (Model-Widok-Kontroler) to wzorzec projektowy stosowany do podziału struktury aplikacji na moduły. Zakłada on wyodrębnienie trzech zasadniczych części systemu komputerowego:

- Model – moduł odpowiedzialny za model danych oraz reguły biznesowe
- Widok – moduł odpowiedzialny za sposób prezentacji danych użytkownikowi systemu
- Kontroler – moduł odpowiedzialny za interakcję użytkownika z systemem. Przyjmuje żądania od użytkownika, przekazuje przetworzone dane do modelu oraz zajmuje się zarządzaniem widoków – ich odświeżaniem etc.

W przypadku aplikacji internetowych wykorzystanie tego wzorca projektowego wygląda następująco:

1. Klient wysyła żądanie HTTP do kontrolera, który odbiera dane wejściowe, interpretuje je i przetwarza.
2. Przetworzone dane służą jako dane wejściowe do operacji wykonywanych na modelu np. zmienia się stan jakiegoś obiektu.

3. Końcowy etap to zwrócenie odpowiedzi do użytkownika przeważnie w formie JSON, XML czy HTML oraz odświeżenie odpowiednich widoków.

3.4.1 Wzorzec MVC w szkielecie Spring

Szkielet Spring wspiera wzorzec MVC poprzez moduł Spring Web MVC. Najważniejszym komponentem ww. modułu jest `DispatcherServlet` (serwlet dyspozytor). Dyspozytor wysyła żądania użytkownika do specjalnie przygotowanych kontrolerów implementowanych przez programistów pracujących przy projekcie. Moduł MVC w Spring jest szczególnie przydatny gdy zarówno cała logika jak i widoki są generowane po stronie serwera. Aplikacja Money korzysta głównie z mechanizmów kontrolera.

3.4.2 Wzorzec MVC w szkielecie AngularJS

W szkielecie AngularJS wzorzec MVC pozwala na utrzymaniu kodu w uporządkowanej i testowalnej postaci. Kontroler AngularJS to klasa języka JavaScript, która jest podczepiana do zasięgu wybranego widoku. To sprawia, że współdzielenie modelu pomiędzy widokiem a kontrolerem jest bardzo proste.

Model to zbiór obiektów i prymitywów, do których można odwoływać się z obiektu Scope ($scope). To sprawia, że testowanie kontrolera w izolacji jest bezproblemowe. AngularJS zezwala na stworzenie instancji oraz testowanie kontrolera bez widoku, gdyż nie ma powiązania pomiędzy kontrolerem a widokiem.

Ciekawsze cechy MVC w AngularJS:

- Kontroler nie ma żadnego bezpośredniego powiązania z logiką renderowania widoku.
- Instancja kontrolera jest tworzona przez AngularJS i wstrzykiwana do widoku
- Kontroler może być zainicjowany w izolacji (bez widoku) i kod się wykona. Umożliwia to wygodne testowanie.
- Widok HTML jest odzwierciedleniem modelu.
- Zmiana modelu, powoduje automatyczną zmianę widoku.
- Widok może wywołać jakąkolwiek funkcję z kontrolera.

3.6 Spring Data

Spring Data to biblioteka która ma za zadanie ułatwić programistom dostęp do źródła danych. Dzieli się na wiele mniejszych podprojektów związanych ściślej z konkretnymi technologiami baz danych. Jest to jeden z najmłodszych produktów z rodziny Spring.

W wyniku obserwacji cech wspólnych niezależnych od systemu baz danych, w pakiecie Spring Data znajdziemy moduł Spring Data Commons. Wszystkie specyficzne moduły dziedziczą z tego właśnie pakietu.

Spring Data pozwala na obsłużenie zarówno tradycyjnych, relacyjnych baz danych jak i baz NoSQL.

Do głównych projektów związanych ze Spring Data należą:

- Spring Data JPA
- Spring Data MongoDB
- Spring Data Neo4J
- Spring Data Redis
- Spring Data For Hadoop
- Spring Data GemFire
- Spring Data Rest
- Spring Data JDBC Extensions

Aplikacja Money wykorzystuje moduł Spring Data JPA. Ułatwia on implementowanie repozytoriów typu Java Persistence API. W przypadku implementacji bez użycia Spring Data programista wytwarza bardzo wiele powtarzających się linijek kodu dla każdej prostej operacji. Skutkuje to zmniejszeniem motywacji do pracy i zarazem zwiększeniem prawdopodobieństwa popełnienia błędów. Spring Data JPA pozwala ograniczyć tego typu operacje i zmniejszyć znacząco wysiłek programisty. Wspiera stronicowanie oraz audyt, a także przejmuje od programisty obsługę transakcji oraz mapowanie wyjątków, przekazując te obowiązki do Spring Transactions.

Poniżej znajduje się kod przykładowego interfejsu repozytorium.

```
public interface SubcategoryRepository extends JpaRepository<Subcategory, Long> {
    public List<Subcategory> findByName(String name);
}
```

Rysunek 5. Kod przykładowego interfejsu repozytorium

Zadaniem programisty jest jedynie przygotowanie podobnego fragmentu kodu. Implementacją zadeklarowanych metod zajmuje się biblioteka, w trakcie wykonywania programu. Metoda `findByName` ma posłużyć do wydobycia z bazy danych wszystkich podkategorii o podanej nazwie. Co ciekawe, w przypadku gdy podkategoria ma w swojej klasie pole `name`, nie są potrzebne żadne zapytania JPQL czy SQL. Biblioteka na podstawie określonych reguł[3], sama odnajduje określone informacje [7].

Oczywiście w przypadku gdy programista chce uzyskać wyniki zapytań niespełniających wyżej wymienionych warunków, może to zrobić przy użyciu adnotacji `@Query`, czyli przy użyciu języka Java Persistence Query Language.

```
public interface TransactionRepository extends JpaRepository<Transaction, Long> {
    @Query("SELECT t FROM Transaction t WHERE LOWER(t.budgetId) = LOWER(:budgetId)")
    public List<Transaction> findByBudget(@Param("budgetId") Long budgetId);
    }
```

Rysunek 6. Kod metody z użyciem JPQL

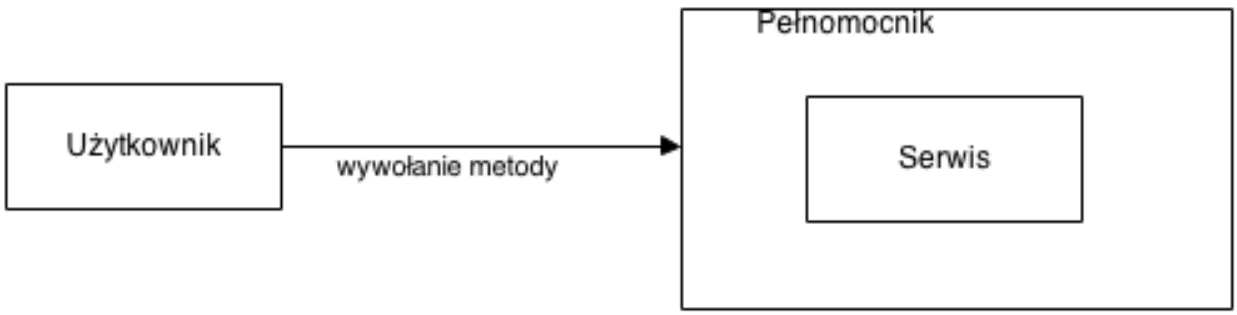
W powyższym przykładzie w liście argumentów, przy jedynym argumencie, możemy zauważyć adnotację `@Param` określającą nazwę danego argumentu w zapytaniu JPQL. W adnotacji `@Query` po dwukropku wykorzystujemy parametr dostarczony przez użytkownika do zapytania. W tym wypadku jest to `budgetId`.

[3]Nomenklatura metod: http://docs.spring.io/spring-data/jpa/docs/1.4.3.RELEASE/reference/html/

3.7 Spring Security

Spring Security to kolejna biblioteka z rodziny Spring, mająca na celu ułatwienie zarządzania bezpieczeństwem w aplikacji. Dostarcza możliwość autentykacji, autoryzacji oraz innych opcji przydatnych w aplikacjach biznesowych. Biblioteka ta powstała w roku 2003 jako Acegi Security. Inicjatorem tego projektu był Ben Alex. Obecnie nad rozwojem projektu pracuje firma SpringSource zajmująca się całym szkieletem Spring.

Spring Secuirty realizuje ochronę metod serwisowych – dla każdego uruchamianego serwisu jest tworzony pełnomocnik (ang. proxy). Jest to wzorzec projektowy, którego celem jest stworzenie obiektu, który zastępuje inny obiekt. Zatem, gdy użytkownik wywołuje jakąś metodę serwisową, najpierw sprawdzane są uprawnienia, a dopiero potem następuje ewentualne oddelegowanie do metody serwisowej. Na poniższym rysunku przedstawiony jest schemat działania wzorca pełnomocnika.



Rysunek 7. Schemat działania wzorca pełnomocnika

Spring Security jest rozpowszechniany na licencji Apache License 2.0.

3.8 AngularUI i Twitter Bootstrap

AngularUI to biblioteka rozszerzająca szkielet AngularJS o dodatkowe możliwości. Skupiają się one wokół interfejsu użytkownika. Aplikacja Money korzysta z modułu UI-Bootstrap. Jest to Twitter Bootstrap przepisany przy użyciu wyłącznie AngularJS.

Twitter Bootstrap to szkielet CSS rozwijany przez programistów firmy Twitter, wydawany na licencji Apache License 2.0. Dostarcza zbioru szablonów formularzy, przycisków, czcionek i innych komponentów wyświetlanych na stronach WWW. Jest napisany głównie przy użyciu języka HTML i CSS, ale używa też języka JavaScript, dzięki czemu można rozszerzać funkcjonalność komponentów.

Aplikacja Money korzysta z niżej wymienionych elementów pakietu Bootstrap:

- przyciski,
- okna modalne (ang. modal),
- okno do wybierania dat (ang. datepicker),
- grupowanie nagłówków tabel (ang. accordion).

3.9 Angular-charts

Angular-charts to biblioteka rozszerzająca możliwości szkieletu AngularJS o rysowanie wykresów. Jest udostępniana na licencji MIT. Możliwe rodzaje wykresów to:

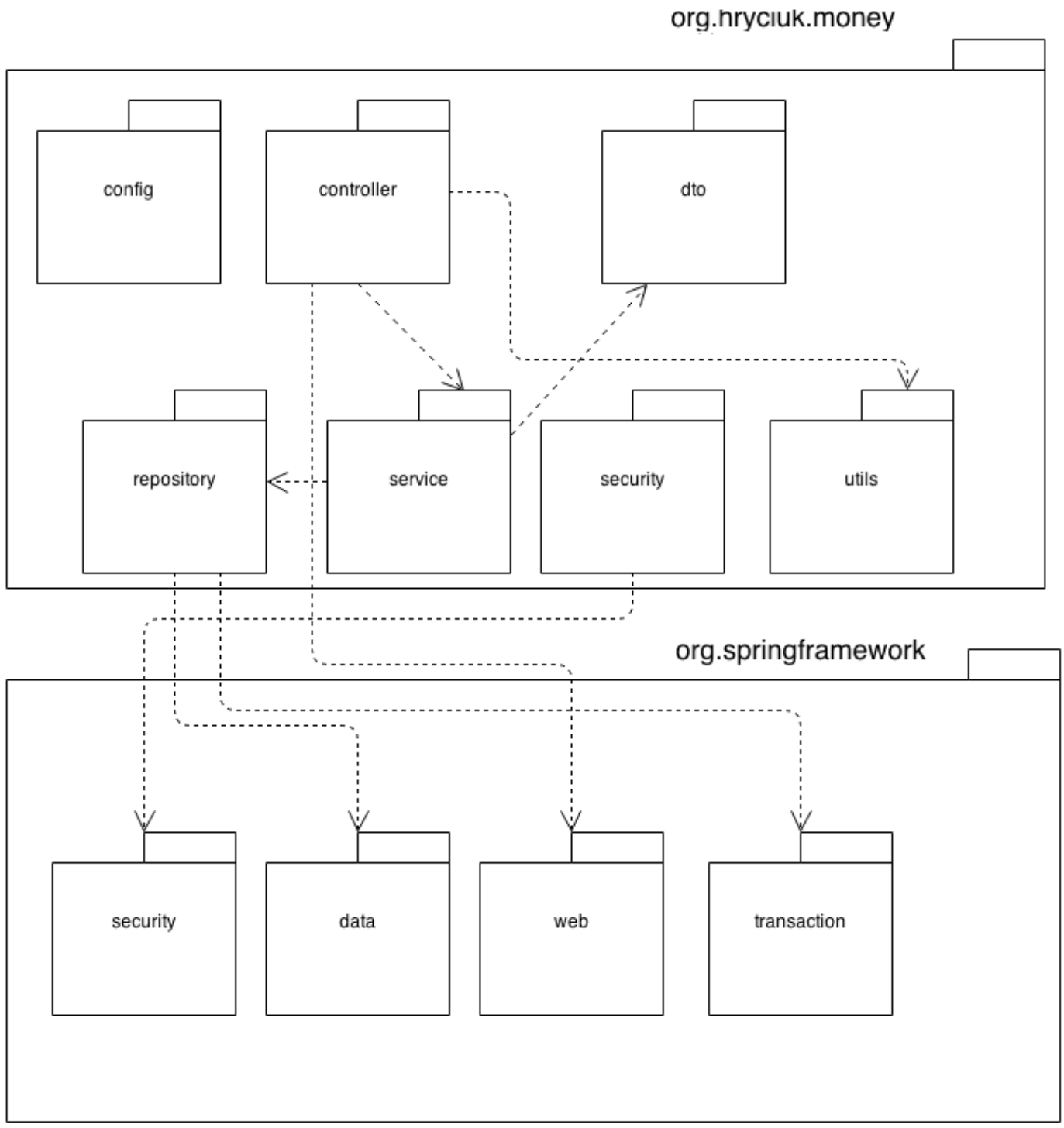
- wykres słupkowy,
- wykres kołowy,
- wykres liniowy,
- wykres punktowy,
- wykres obszarowy.

Wykresy są rysowane na podstawie danych dostarczonych do obiektów JS obsługujących wykres. Dane te mogą być ustalone odgórnie w kodzie AngularJS, a mogą być także dostarczane dynamicznie. Aplikacja Money dostarcza obiekty w formacie JSON otrzymane ze strony serwera, do kodu AngularJS, gdzie dane są przetwarzane i w rezultacie wykorzystywane przez bibliotekę Angular-charts.

Angular-charts wykorzystuje do swojego działania dwie inne biblioteki JavaScript tj. jQuery oraz d3.

4. Projekt aplikacji

4.1 Diagram pakietów

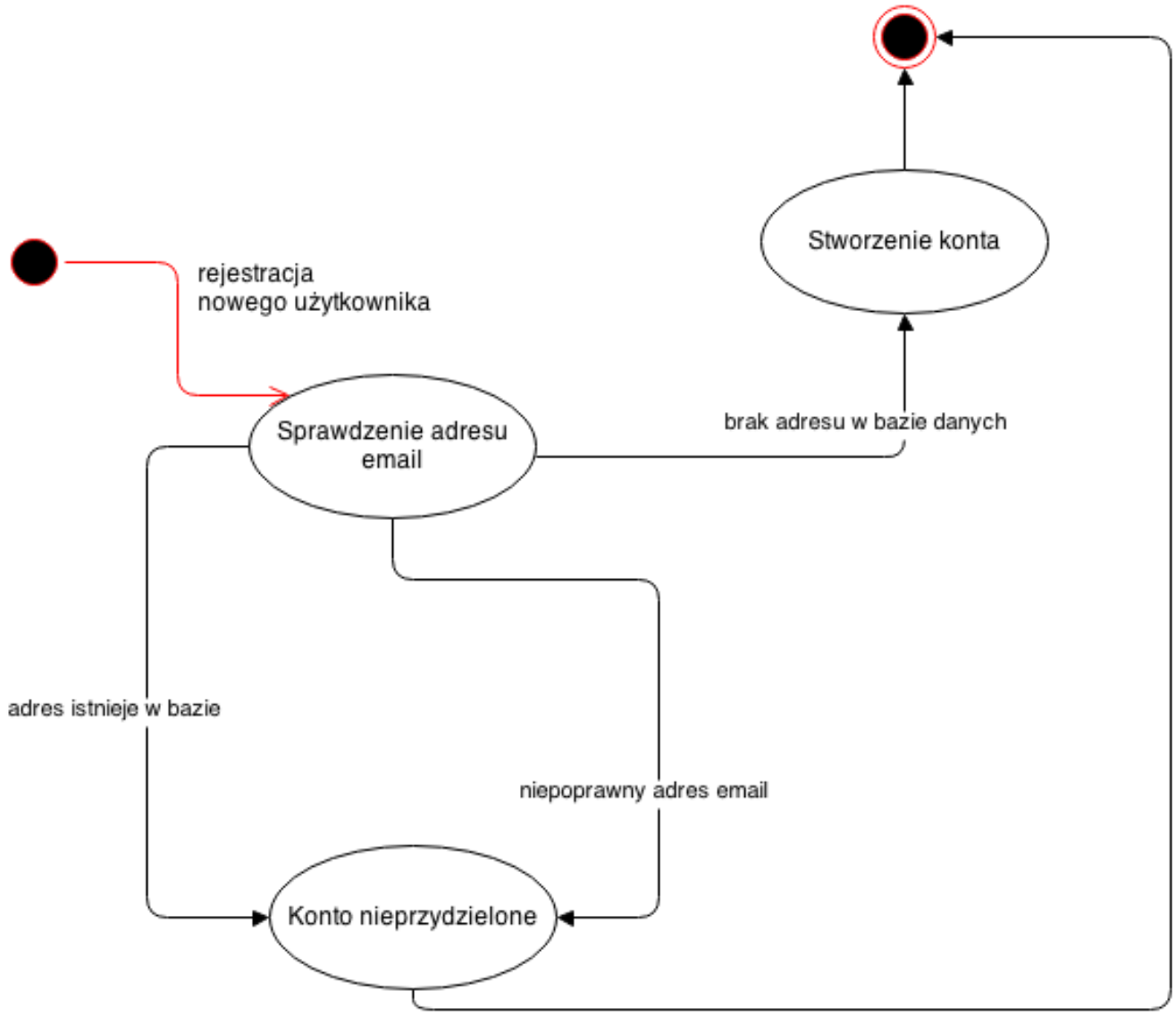


Rysunek 8. Diagram pakietów aplikacji Money

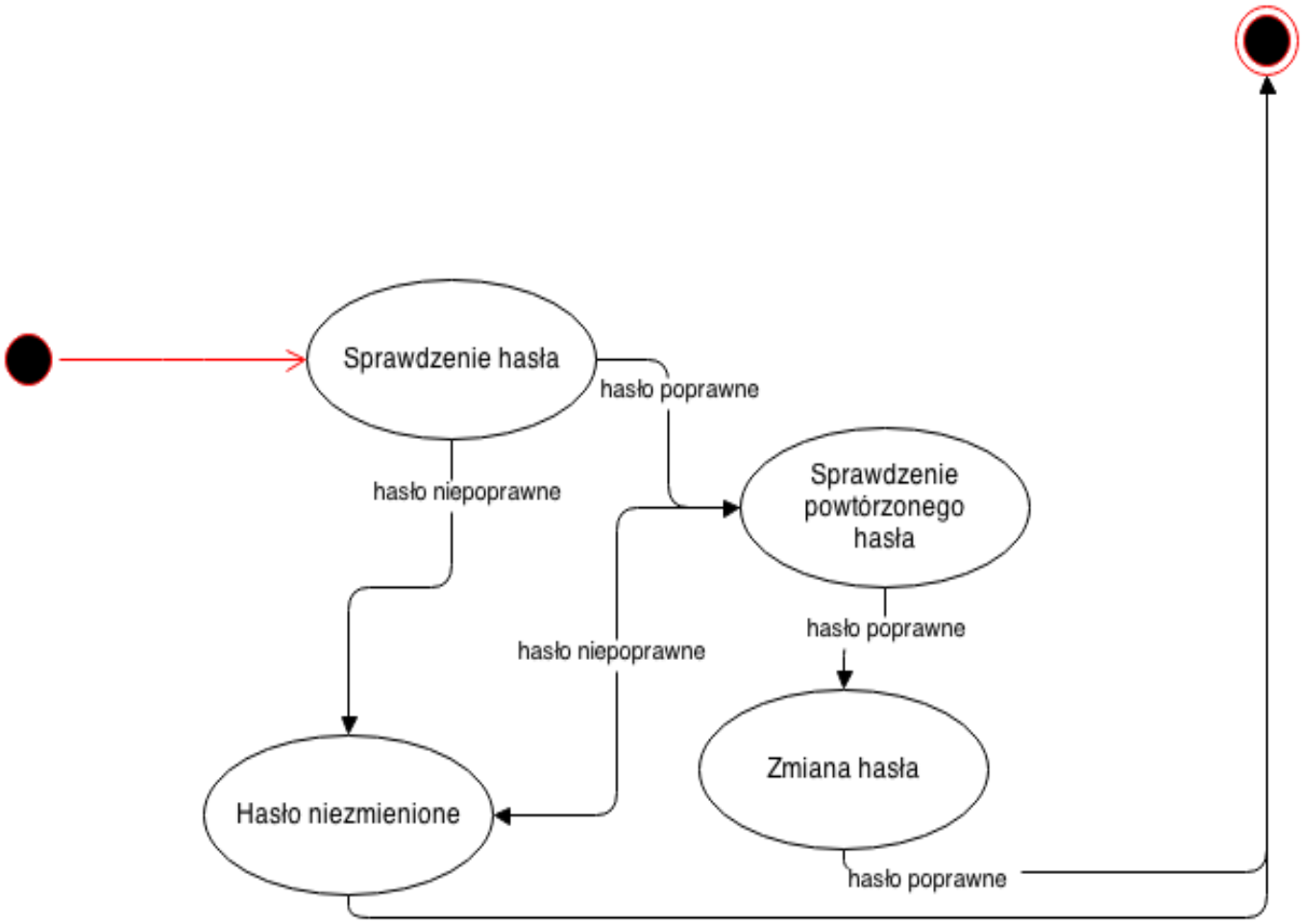
Na powyższym diagramie został celowo pominięty pakiet `org.hryciuk.model`, ze względu na poprawę czytelności rysunku. Jest to pakiet, który przechowuje klasy reprezentujące obiekty domenowe w aplikacji Money.

- Rysunek wizualizuje pakiety w aplikacji Money a także ich powiązanie z pakietami szkieletu Spring.
- Pakiet `org.hryciuk.money.config` zawiera pliki konfigurujące projekt. W pierwszych wersjach szkieletu Spring konfiguracji można było dokonać jedynie przy pomocy plików XML. Obecnie obydwie opcje są powszechnie używane.
- Pakiet `org.hryciuk.money.controller` zawiera pliki kontrolerów poszczególnych obiektów istniejących w aplikacji. W pakiecie tym są definiowane także adresy metod po których wywołaniu aplikacja AngularJS otrzymuje pliki JSON (stąd powiązanie z pakietem `org.springframework.web`)
- Pakiet `org.hryciuk.money.dto` zawiera klasy odpowiedające klasom z pakietu `org.hryciuk.money.model` ale z mniejszą ilością dostępnych pól. Obiekty te są używane do komunikacji przy pomocy standardu JSON. Powodem jest wydajność – obiekty DTO zajmują mniej pamięci
- Pakiet `org.hryciuk.money.repository` to pakiet przechowujący pliki dziedziczące po JPARepository. Odpowiedzialny jest za komunikację z bazą danych
- Pakiet `org.hryciuk.money.service` zawiera implementację metod serwisowych, wykonujących operacje na bazie danych.
- Pakiet `org.hryciuk.money.security` zawiera implementację klas związanych z bezpieczeństwem.
- Pakiet `org.hryciuk.money.utils` zawiera metody przydatne w pozostałych pakietach w dziedzinie aplikacji. Udostępnia głównie metody powiązane z wyznaczaniem przedziałów czasowych na podstawie dostarczanych danych wejściowych.

4.2 Diagramy maszyny stanowej

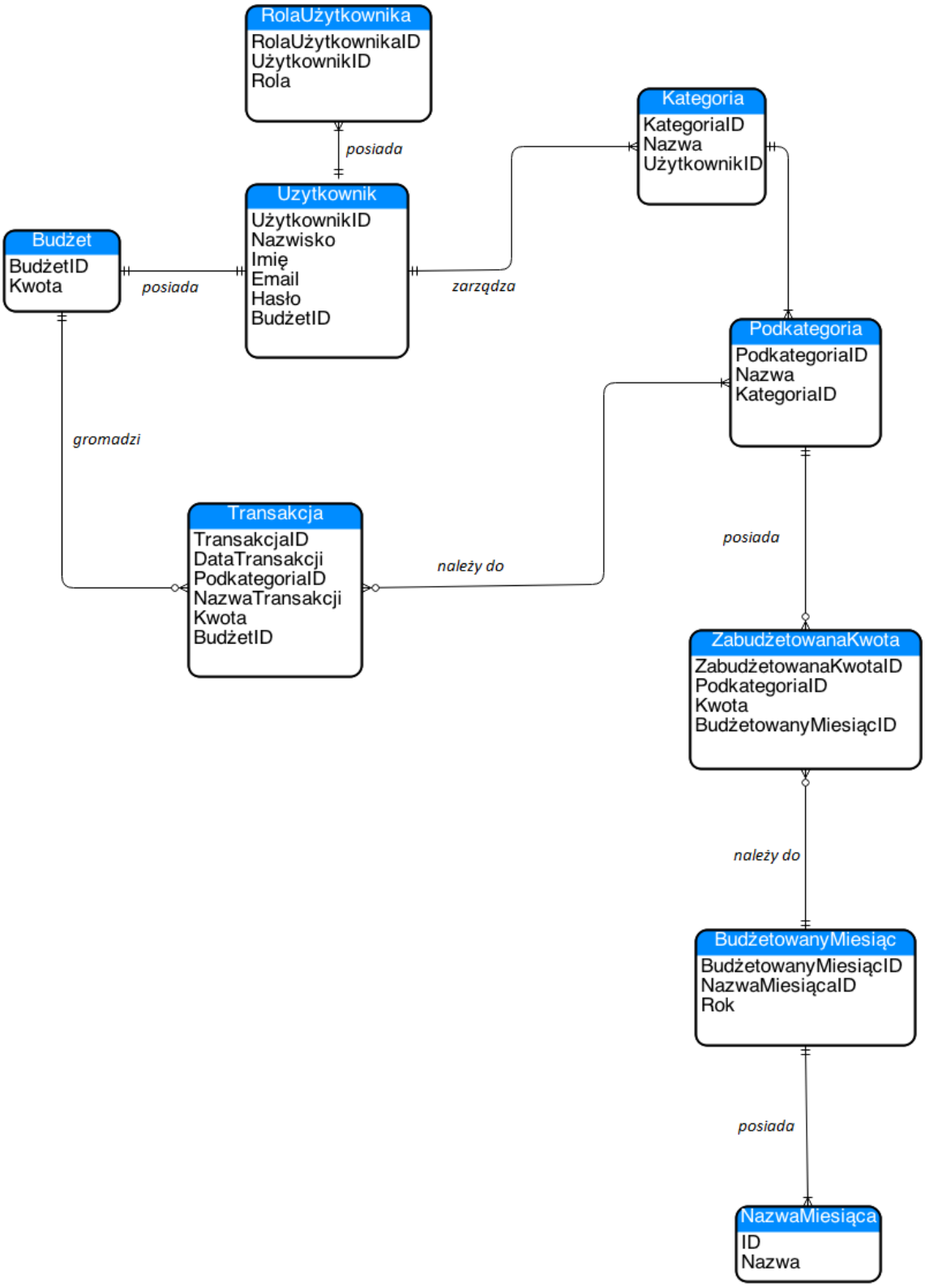


Rysunek 9. Dodanie nowego użytkownika – Diagram maszyny stanowej



Rysunek 10. Zmiana hasła – Diagram maszyny stanowej

4.3 Diagram encji



Rysunek 11. Diagram encji

Rola użytkownika

- Rola jest identyfikowana w systemie za pomocą ID.
- Każdy użytkownik może posiadać jedną lub więcej ról.
- Istniejące role to:
    - ROLE_USER – rola użytkownika
    - ROLE_ADMIN – rola administratora
- System może być rozszerzony o dodatkowe role.

Użytkownik

- Użytkownik jest identyfikowany w systemie za pomocą ID.
- Każdy użytkownik ma unikalny adres email.
- Hasło użytkownika jest zapisywane w bazi danych jako efekt funkcji skrótu MD5.
- Użytkownik jest powiązany z dokładnie jednym budżetem. Dzięki temu może pozbyć się całej historii transakcji jednocześnie zachowując stworzoną hierarchię kategorii i podkategorii.

Budżet

- Budżet jest identyfikowany w systemie za pomocą ID.
- Jest powiązany z konkretnym użytkownikiem i nie może bez niego istnieć.
- Zawiera zsumowaną kwotę wydatków.

Kategoria

- Kategoria jest identyfikowana w systemie za pomocą ID.
- Użytkownik może posiadać wiele przypisanych do siebie kategorii.
- Może istnieć wiele kategorii o tej samej nazwie, powiązanych z różnymi użytkownikami.

Podkategoria

- Podkategoria jest identyfikowana w systemie za pomocą ID.
- Kategoria może posiadać wiele przypisanych do siebie kategorii.
- Może istnieć wiele podkategorii o tej samej nazwie w obrębie innych kategorii.

Transakcja

- Transakcja jest identyfikowana w systemie za pomocą ID.
- Transakcja jest powiązana z konkretnym budżetem, a więc i z użytkownikiem.
- Każda transakcja należy do jakiejś podkategorii, a więc i do kategorii.
- Każda transakcja musi mieć datę w której została wykonana.
- Każda transakcja musi mieć określoną kwotę.
- Każda transakcja musi mieć określoną nazwę.

Zabudżetowana Kwota

- Zabudżetowana kwota jest identyfikowana w systemie za pomocą ID.
- Zabudżetowana kwota jest powiązana z podkategorią danego użytkownika oraz z określonym budżetowanym miesiącem. Dzięki temu każdy użytkownik może mieć zabudżetowane różne kwoty dla tego samego miesiąca.

Budżetowany Miesiąc

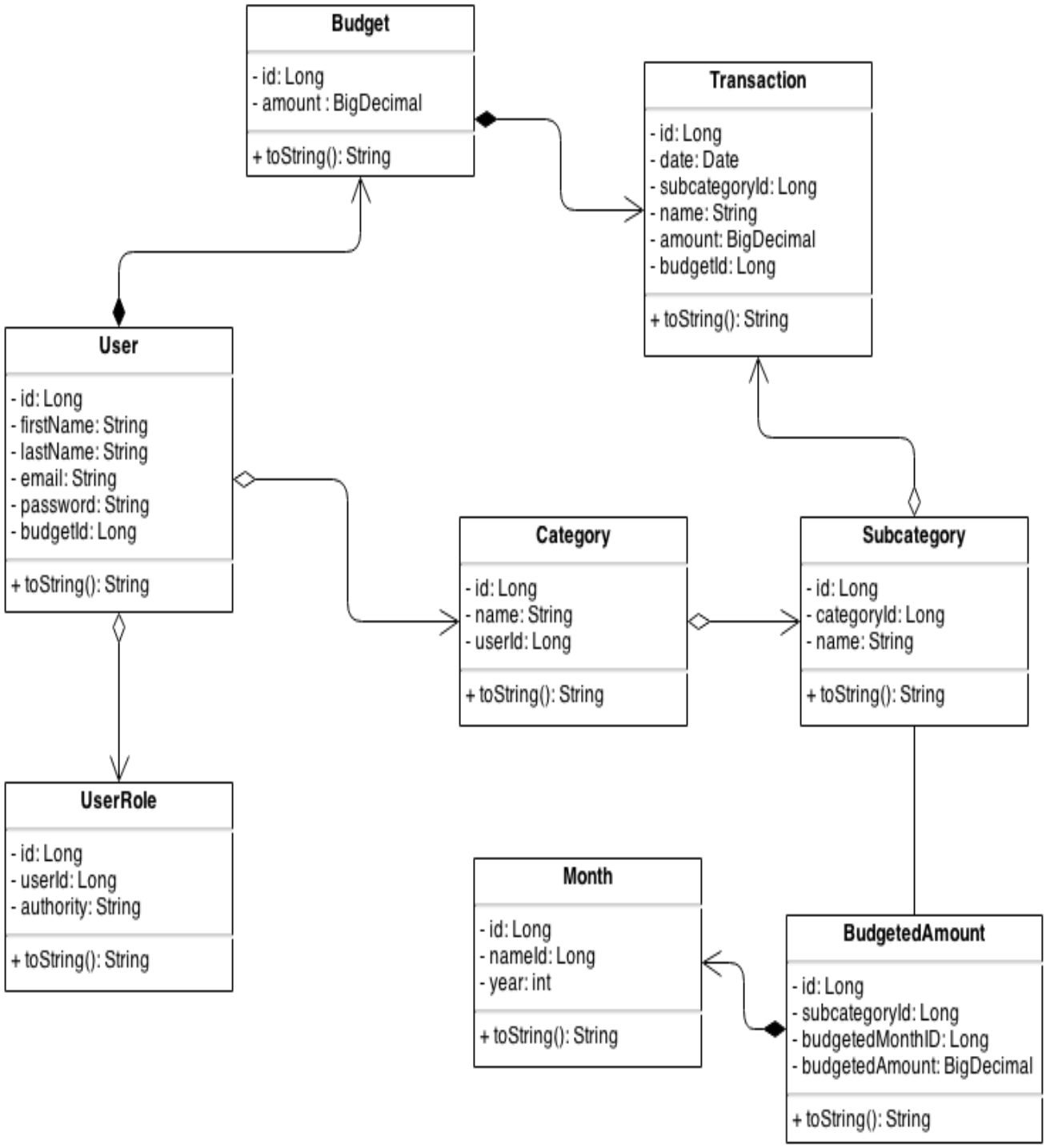
- Budżetowany miesiąc jest identyfikowany w systemie za pomocą ID.
- Budżetowany miesiąc musi mieć określony rok.
- Budżetowany miesiąc jest powiązany z nazwą miesiąca.

Nazwa miesiąca

- Nazwa miesiąca jest identyfikowana w systemie za pomocą ID.
- Przechowuje wszystkie nazwy miesięcy z odpowiednią dla nich numeracją.

4.4 Diagram klas

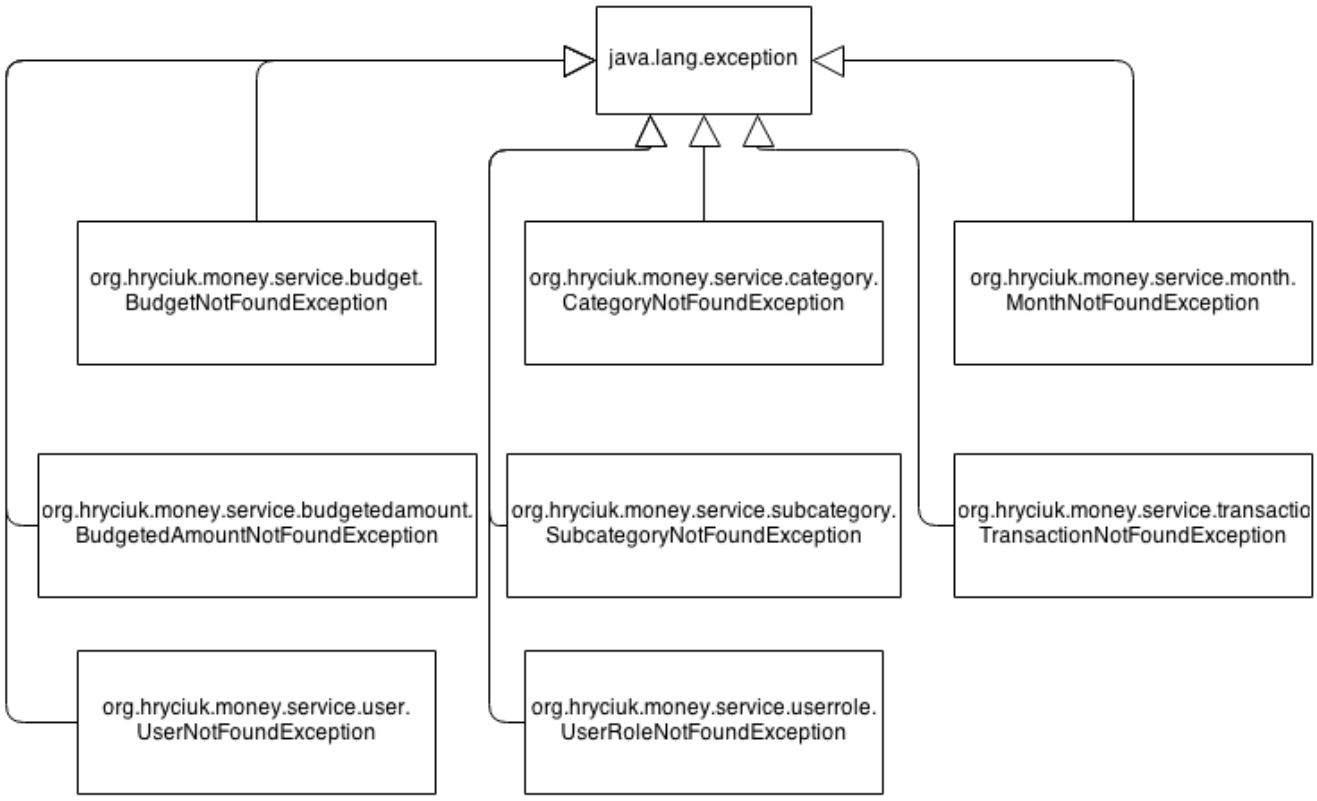
W klasach na poniższym wykresie pominięte zostały metody typu get() oraz set(). Klasy te pochodzą z pakiet `org.hryciuk.money.model`



Rysunek 12. Diagram klas

4.5 Wyjątki

W aplikacji Money zostały stworzone wyjątki mające na celu łatwe odnalezienie zidentyfikowanie problemu w przypadku wyjątkowego zachowania aplikacji.



Rysunek 13. Diagram klas – stworzone wyjątki w aplikacji Money

- `BudgetNotFoundException` – wyjątek rzucany w przypadku próby wykonania operacji na obiekcie typu `Budget`, który nie istnieje w bazie danych. Np. usunięcie budżetu który nie istnieje.
- `BudgetedAmountNotFoundException` - wyjątek rzucany w przypadku próby wykonania operacji na obiekcie typu `BudgetedAmount`, który nie istnieje w bazie danych.
- `CategoryNotFoundException` - wyjątek rzucany w przypadku próby wykonania operacji na obiekcie typu `Category`, który nie istnieje w bazie danych.
- `SubcategoryNotFoundException` - wyjątek rzucany w przypadku próby wykonania operacji na obiekcie typu `Subcategory`, który nie istnieje w bazie danych.

- `MonthNotFoundException` - wyjątek rzucany w przypadku próby wykonania operacji na obiekcie typu `Month`, który nie istnieje w bazie danych.
- `TransactionNotFoundException` - wyjątek rzucany w przypadku próby wykonania operacji na obiekcie typu `Transaction`, który nie istnieje w bazie danych.
- `UserNotFoundException` - wyjątek rzucany w przypadku próby wykonania operacji na obiekcie typu `User`, który nie istnieje w bazie danych.
- `UserRoleNotFoundException` - wyjątek rzucany w przypadku próby wykonania operacji na obiekcie typu `UserRole`, który nie istnieje w bazie danych.

4.6 Adresowanie

Adresowanie podstron jest istotnym elementem aplikacji, który pozwala na uporządkowanie zarządzania dostępu do stron, a także na kategoryzację dostępnych metod. Zastosowanie takiego samego prefiksu dla powiązanych ze sobą stron, wpływa na skuteczne zwiększenie bezpieczeństwa (reguły w Spring Security) . Ponadto, adresowanie podstron wedle przyjętej konwencji oraz przesyłanie do nich parametrów przy pomocy HTTP GET pozwala na przesyłanie tych adresów we wszelkiego rodzaju wiadomościach elektronicznych.

4.6.1 Adresowanie w aplikacji Money

Transakcje

- Wszystkie adresy funkcji serwisowych dotyczących transakcji zaczynają się prefiksem `/transactions`
- Funkcje zwracające sumy transakcji na przestrzeni danego czasu są adresowane wg konwencji:
  `transactions/{budgetId}/sum/{week, month, year}`
- Podobna konwencja jest użyta przy próbie uzyskanie informacji na temat sumy transakcji w określonym przedziale czasu dla określonej kategorii:
  `/transactions/{budgetId}/category/{categoryId}/sum/{week, month, year}`

Kategorie

- Wszystkie adresy funkcji serwisowych dotyczących kategorii zaczynają się prefiksem /categories

Podkategorie

- Wszystkie adresy funkcji serwisowych dotyczących podkategorii zaczynają się prefiksem /subcategories

Uwierzytelnianie

- Wszystkie adresy zaczynające się od prefiksu `/api` dotyczą uwierzytelniania użytkownika
- `/api/login` dotyczy logowania użytkownika oraz utworzenia pliku cookie
- `/api/logout` dotyczy wylogowania użytkownika i usunięcia pliku cookie.

4.6.2 Zabezpieczanie adresów po stronie serwera

Zabezpieczenie dostępu do adresów osobom niepożądanym odbywa się poprzez określenie zasad w konfiguracyjnym pliku XML o nazwie `applicationContext-security.xml` w folderze `/src/main/resources`. Przykładowy fragment kodu:

```
<security:intercept-url pattern="/api/user/" access="permitAll()"/>
<security:intercept-url pattern="/transactions/**" access="hasRole('ROLE_USER')"/>
```

Rysunek 14. Przykład konfiguracji zabezpieczania stron

W powyższym przykładzie należy zwrócić szczególną uwagę na atrybut `access` W przypadku, gdy strona nie wymaga tego by użytkownik był zalogowany, używamy opcji `permitAll()`. W przypadku gdy chcemy udostępnić dane adresy użytkownikom o zadanej roli używamy opcji `hasRole('NAZWA_ROLI')`.

4.6.3 Zabezpieczanie adresów po stronie klienta

W przypadku aplikacji Money równie istotne jak zabezpieczenia po stronie serwera są zabezpieczenia adresów po stronie klienta w AngularJS. W przypadku gdy, użytkownik próbuje dostać się do strony, której nie ma prawa oglądać, AngularJS przekierowuje go do strony głównej aplikacji (strony logowania). Odbywa się to poprzez użycie *serwisu* AngularJS, który przez cały czas trwania sesji utrzymuje informacje o użytkowniku (tj. czy jest on zalogowany) a także jest dostępny z poziomu każdej podstrony aplikacji. Oto deklaracja serwisu SharedDate z aplikacji Money:

```
moneyApp.factory('SharedData', function() {
    return {
        isLogged: false,
        budgetId: '',
        userId: ''
    };
});
```

Rysunek 15. Implementacja serwisu AngularJS

Przykładowy fragment kodu definiującego, czy użytkownik musi być zalogowany aby przeglądać docelową stronę:

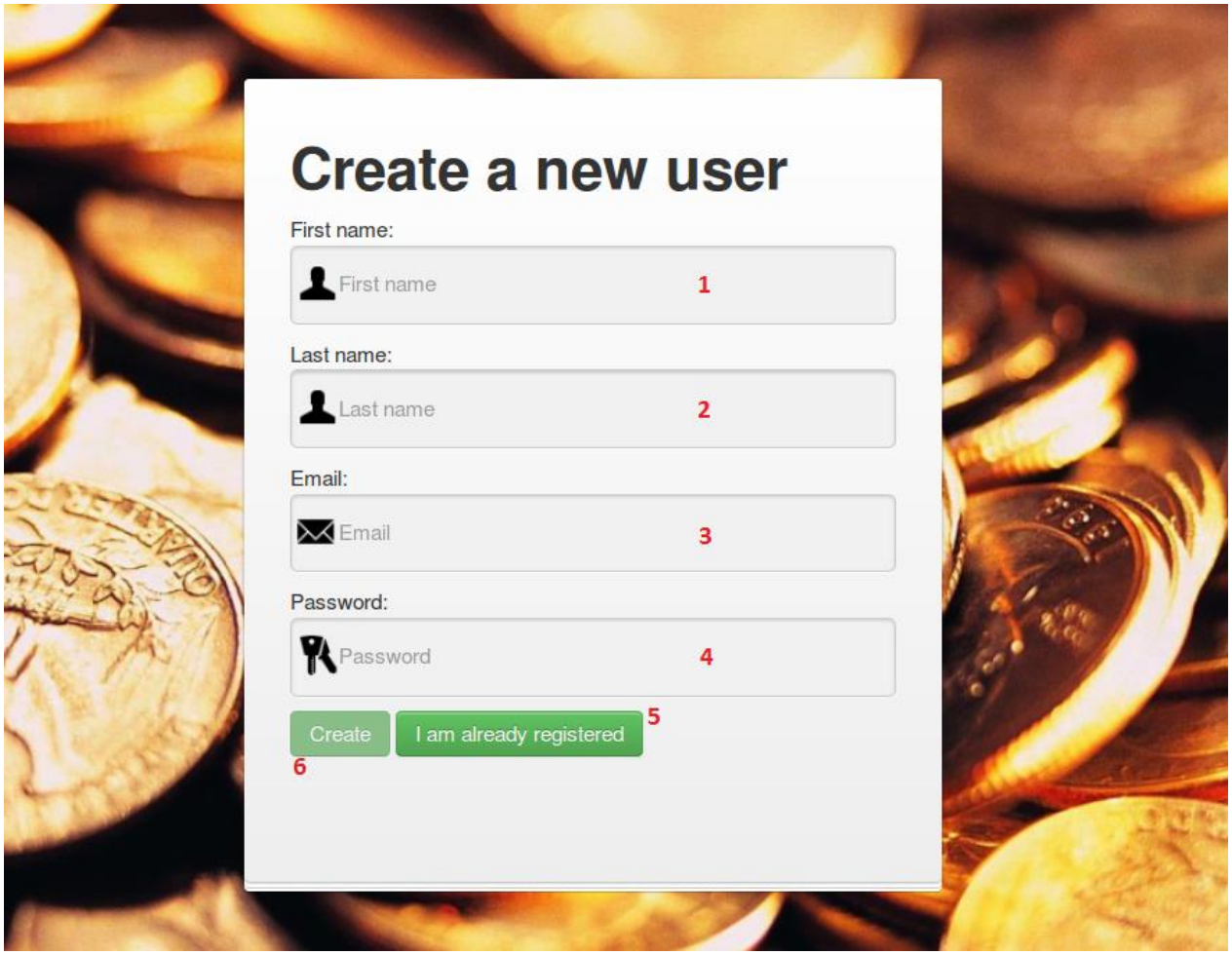
```
$routeProvider.when('/login', {templateUrl: 'partials/login.html',
                    controller: 'LoginFormController',
                    requiresLogin: false});
$routeProvider.when('/categories/:userId', {templateUrl: 'partials/categories.html',
                    controller: 'CategoryController',
                    requiresLogin: true});
```

Rysunek 16. Konfiguracja dostępu do stron

W powyższym przykładzie parametr `requiresLogin` przyjmuje wartości funkcji boolowskiej determinując czy docelowa strona jest dostępna z poziomu niezalogowanego użytkownika.

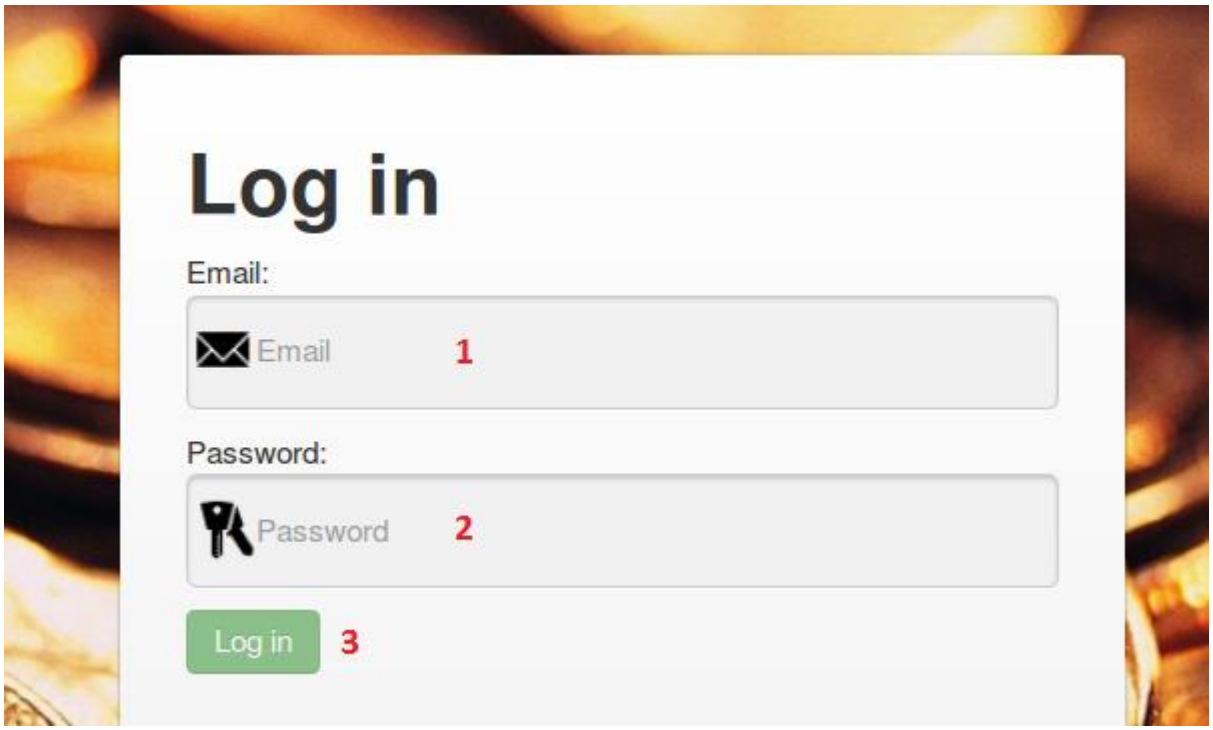
4.7 Interfejs graficzny

Interfejs graficzny jest wykonany przy użyciu biblioteki AngularUI oraz angular-charts. Dodatkowo używane są style kaskadowe CSS.



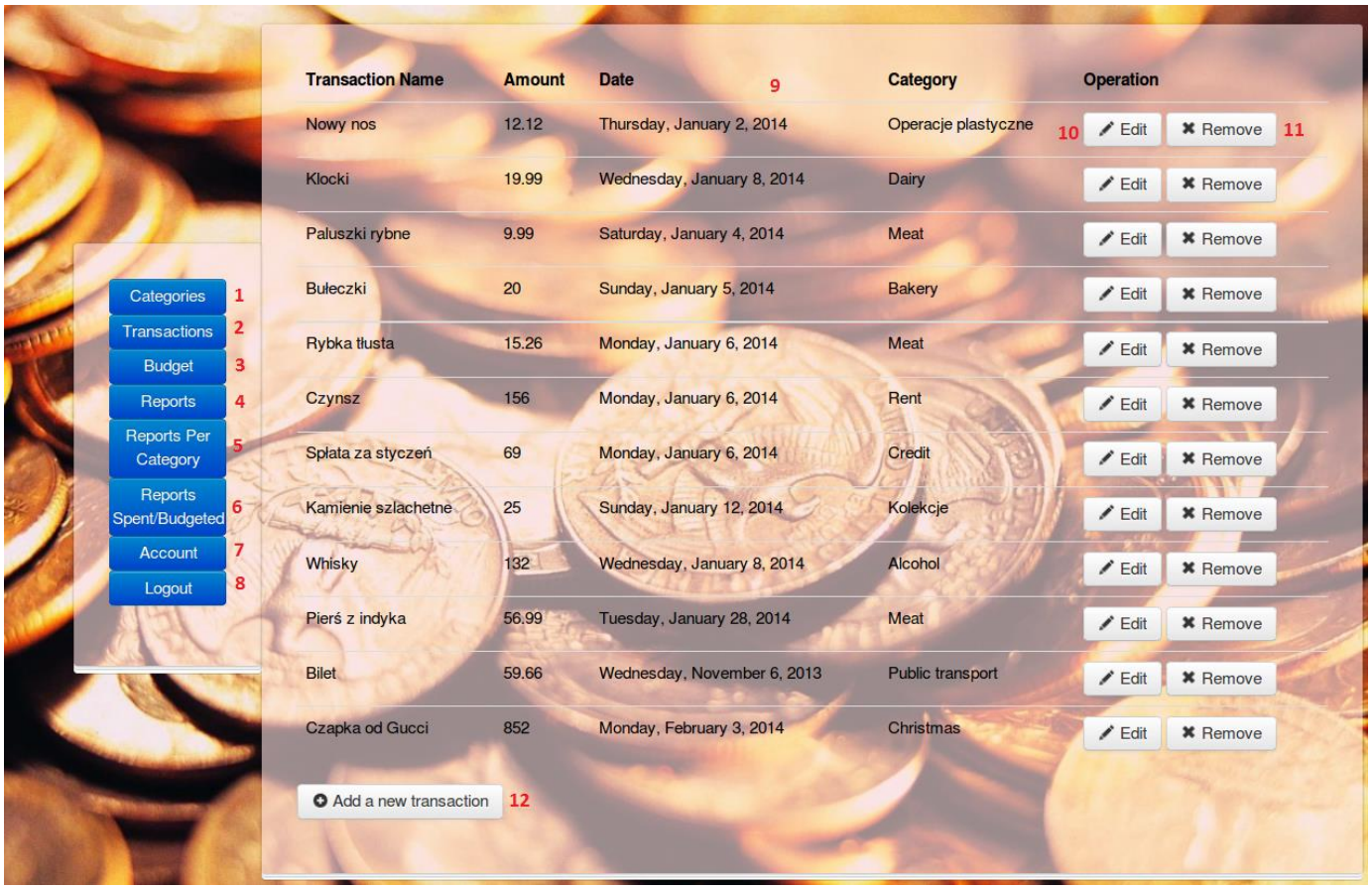
Rysunek 17. Interfejs graficzny – formularz rejestracji

1. Pole tekstowe do wprowadzenia imienia nowego użytkownika
2. Pole tekstowe do wprowadzenia nazwiska nowego użytkownika
3. Pole tekstowe do wprowadzenia adresu email nowego użytkownika
4. Pole tekstowe do wprowadzenia hasła nowego użytkownika
5. Przycisk przekierowujący do formularzu logowania
6. Przycisk wysyłające dane z formularza do serwera, do metody tworzącej nowego użytkownika



Rysunek 18. Interfejs graficzny – formularz logowania

1. Pole tekstowe do wprowadzenia adresu email użytkownika
2. Pole tekstowe do wprowadzenia hasła użytkownika
3. Przycisk logowania, aktywowany dopiero po wprowadzeniu danych.

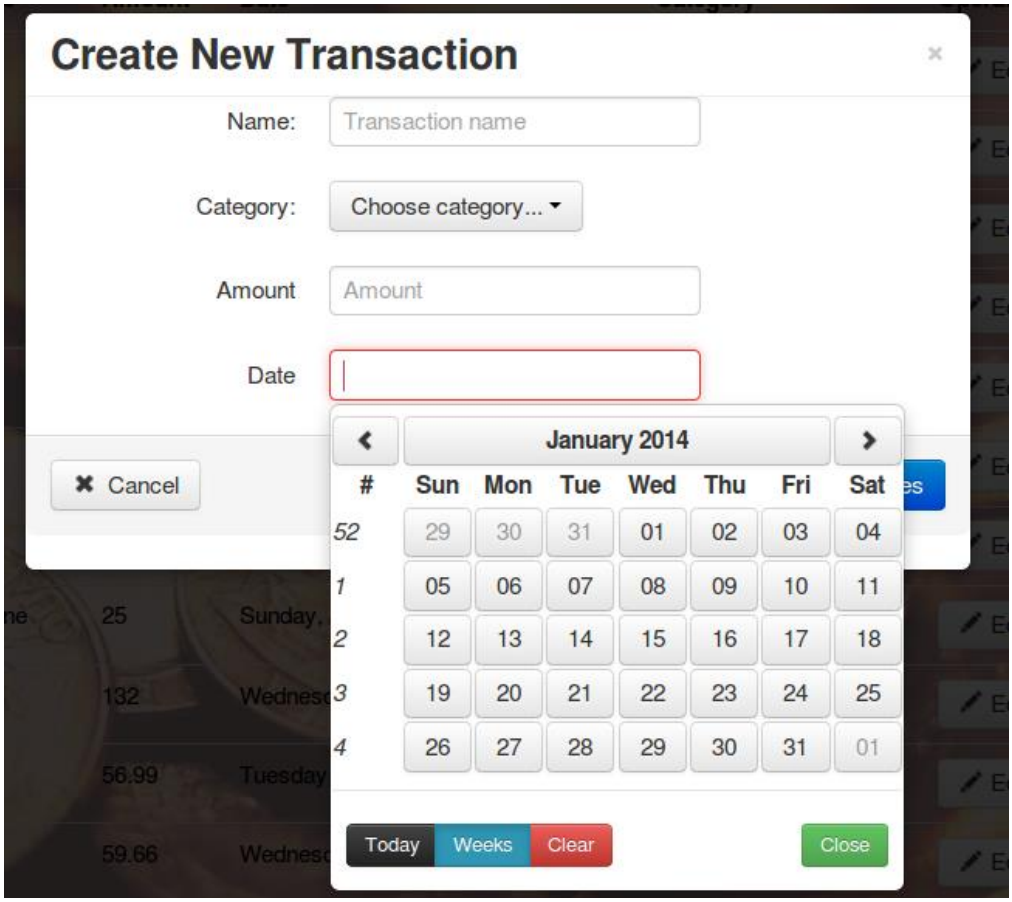


Rysunek 19. Interfejs graficzny – główny widok / widok transakcji

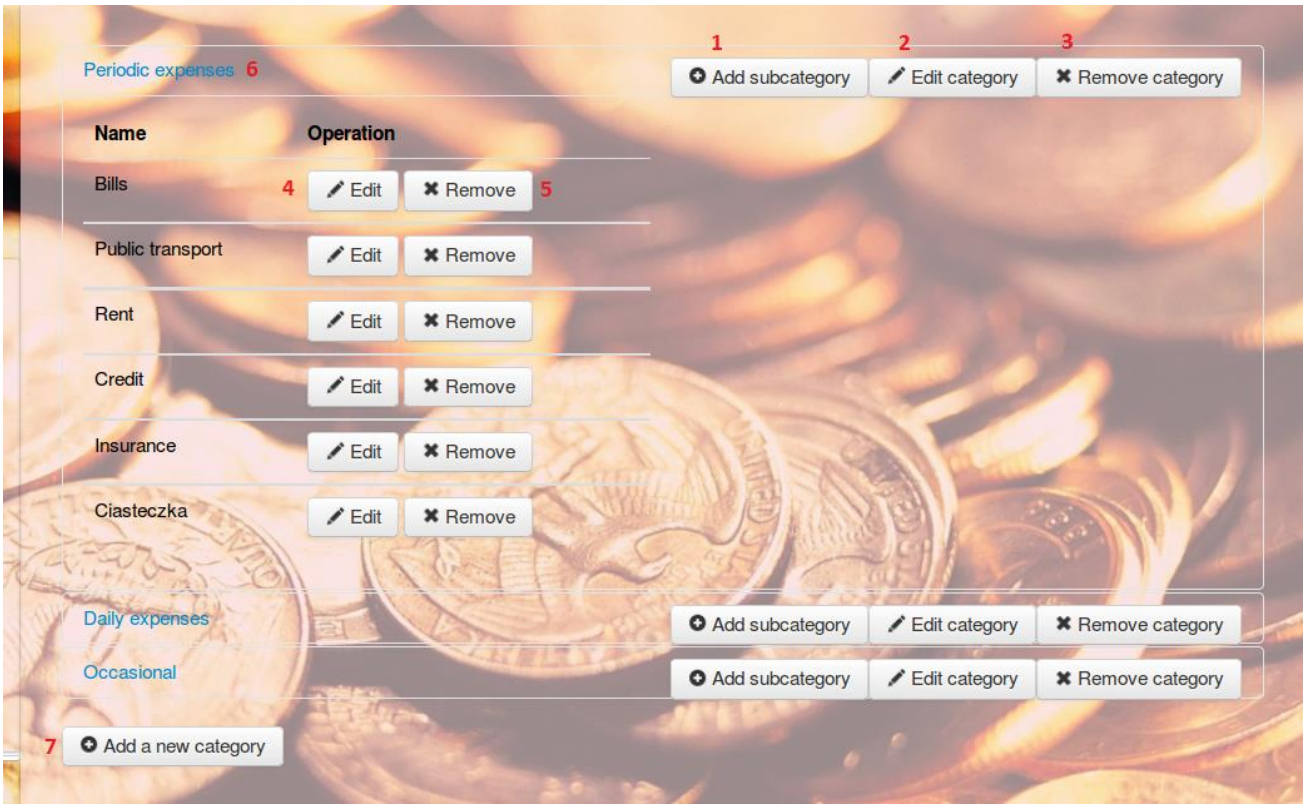
Uwaga. Elementy od 1-8 z poniższej listy to menu pojawiające się na każdym z od tego momentu omawianych widoków.

1. Przycisk przekierowujący do widoku kategorii.
2. Przycisk przekierowujący do głównego widoku / widoku transakcji.
3. Przycisk przekierowujący do widoku budżetowania.
4. Przycisk przekierowujący do widoku ogólnych raportów.
5. Przycisk przekierowujący do widoku raportów z uwzględnieniem wybranej kategorii.
6. Przycisk przekierowujący do raportów porównujących kwoty wydane do zabudżetowanych.
7. Przycisk przekierowujący do widoku z informacjami osobistymi użytkownika.
8. Przycisk wylogowania. Przekierowuje na stronę logowania.
9. Lista transakcji przypisanych do budżetu zalogowanego użytkownika.

10. Przycisk otwierający okienko modalne do edycji informacji o transakcji.
11. Przycisk usuwający daną transakcję
12. Przycisk otwierający okienko modalne z formularzem do dodawania nowej transakcji.

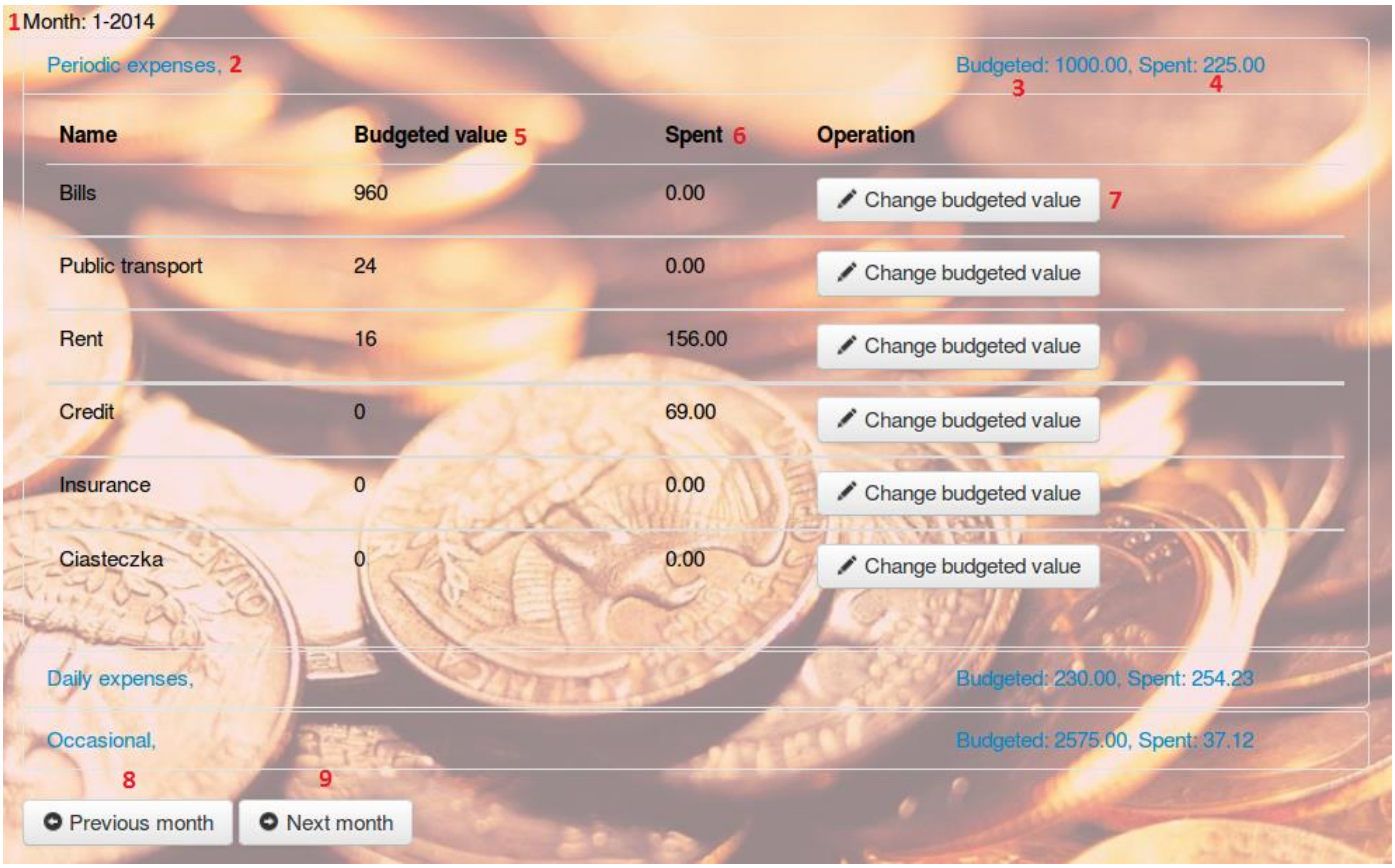


Rysunek 20. Interfejs graficzny – Okienko modalne. Formularz dodawania nowej transakcji



Rysunek 21. Interfejs graficzny – widok kategorii i podkategorii

1. Przycisk otwierający okienko modalne do dodawania nowej podkategorii do kategorii.
2. Przycisk otwierający okienko modalne do edycji wybranej kategorii.
3. Przycisk usuwający wybraną kategorię
4. Przycisk otwierający okienko modalne do edycji wybranej podkategorii.
5. Przycisk usuwający wybraną podkategorię.
6. Nazwa kategorii a jednocześnie przycisk rozwijający/zwijający listę podkategorii
7. Przycisk otwierający okienko modalne z formularzem dodawania nowej kategorii



Rysunek 22. Interfejs graficzny – widok budżetowania

1. Informacja o obecnie oglądanym miesiącu
2. Nazwa budżetowanej kategorii, a jednocześnie przycisk zwijający/rozwijający podkategorie.
3. Suma zabudżetowanych kwot na podkategorie w obrębie danej kategorii.
4. Suma wydanych kwot na podkategorie w obrębie danej podkategorii.
5. Kwota zabudżetowana na daną podkategorię w wybranym miesiącu.
6. Kwota wydana (suma transakcji) w obrębie danej podkategorii w wybranym miesiącu.
7. Przycisk otwierający okienko modalne z formularzem pozwalającym zmienić zabudżetowaną kwotę.
8. Przycisk powodujący zmianę danych w widoku na dane z miesiąca poprzedzającego obecnie wybrany.
9. Przycisk powodujący zmianę danych w widoku na dane z miesiąca następującego po obecnie wybranym.

Rysunek 23. Interfejs graficzny – widok raportowania ogólnego

Uwaga. Elementy od 1-5 z poniższej listy to przyciski nawigacji pojawiające się na każdym z od tego momentu omawianych widoków. Widok porównujący kwotę zabudżetowaną do wydanej przedstawia dane tylko w obrębie miesięcy (budżetujemy finanse na wybrany miesiąc, więc jest to jedyna opcja).

1. Przycisk wyboru przedziału czasowego jako tygodnia.
2. Przycisk wyboru przedziału czasowego jako miesiąca.
3. Przycisk wyboru przedziału czasowego jako roku.
4. Przycisk zmieniający dane na dane z poprzedniego okresu (tydzień, miesiąc, rok)
5. Przycisk zmieniający dane na dane z kolejnego okresu (tydzień, miesiąc, rok)
6. Wykres liniowy wizualizujący wydatki:
    a. Dla tygodnia – dla poszczególnych dni

b. Dla miesiąca – dla poszczególnych dni
c. Dla roku – dla poszczególnych miesięcy

7. Wykres kołowy wizualizujący wydatki:
   a. Dla tygodnia – po podkategoriach
   b. Dla miesiąca – po podkategoriach
   c. Dla roku – po kategoriach
8. Po najechaniu kursorem na wybrany dzień pojawia się suma wydatków w tym dniu.

Rysunek 24. Interfejs graficzny – widok raportowania po kategoriach

1. Informacje o przeglądanym okresie.
2. Wykres liniowy przedstawiający sumę wydatków w obrębie wybranej kategorii (4).
3. Wykres kołowy przedstawiający sumę wydatków z podziałem na podkategorie w obrębie wybranej kategorii (4).

4. Lista rozwijalna pozwalająca wybrać przeglądaną kategorię.

Rysunek 25. Interfejs graficzny – widok raportowania z uwzględnieniem zabudżetowanych środków

1. Informacja o przeglądanym miesiącu
2. Lista rozwijalna pozwalająca wybrać kategorię. Wybór ten ma wpływ na to co jest wizualizowane w wykresie szczegółowym (4).
3. Wykres słupkowy przedstawiający porównanie kwoty zabudżetowanej do sumy wydatków w obrębach kategorii dla wybranego okresu.
4. Wykres słupkowy przedstawiający porównanie kwoty zabudżetowanej do sumy wydatków w obrębie podkategorii dla wybranego okresu.

4.8 Zaprezentowanie warstw systemu na przykładzie implementacji prostej funkcji systemu

Funkcja wydobycia informacji o użytkowniku jest jedną z podstawowych funkcjonalności systemu Money. Przedstawię teraz kod odpowiedzialny za wydobycie informacji o użytkowniku, ukazując złożoność warstw systemu.

```
$scope.getTransactions = function() {
        $http({method: 'GET', url: 'transactions/'
        + $routeParams.budgetId +
        '/all/'}).success(function(data) {
            $scope.transactions = data; // response data
            angular.forEach($scope.transactions, function(i) {
                i.subcategoryName = '';
                $http({method: 'GET', url: 'subcategories/'
                + i.subcategoryId +
                '/'}).success(function(data) {
                    i.subcategoryName = data.name;
                });
            });
        });
    };
```

Rysunek 26. Wydobywanie transakcji - AngularJS

Powyższa funkcja ma za zadanie wywołanie funkcji serwisowej z kodu serwerowego Javy zwracającej obiekt JSON z danymi użytkownika.

```
/**
 * Retrieves a user with specified budgetID
 * @param budgetId - ID of a user's budget
 * @return User JSON object
 */
@RequestMapping(value = "/users/{budgetId}", method = RequestMethod.GET)
public @ResponseBody
User getUser(@PathVariable Long budgetId) {
    User user = userService.findByBudget(budgetId);
    LOGGER.debug(
            "User retrieved using budgetID: {}. Fields." +
            "First name: {}, Last name: {}, Email: {}   ",
            budgetId, user.getFirstName(), user.getLastName(),
            user.getEmail());
    return user;
}
```

Rysunek 27. Wydobywanie transakcji – kontroler użytkownika

Pierwsza warstwa po stronie kodu serwerowego to kontroler użytkownika z odpowiednimi adnotacjami. Adnotacja `@RequestMapping` określa adres po którym można odwołać się do danej funkcji na serwerze gdzie aplikacja się znajduje. Określona jest także metoda http, która jest obsługiwana. Adnotacja `@ResponseBody` oznacza, że obiekt User będzie zwracany jako odpowiedź tej funkcji. Z kolei adnotacja `@PathVariable` oznacza zmienną używaną w adresie – w tym wypadku jest to `{budgetId}`.

```
@Override
public User findByBudget(Long budgetId) {
        LOGGER.debug("Finding user by budgetId: {}", budgetId);
        return userRepository.findByBudget(budgetId);
}
```

Rysunek 28. Wydobywanie transakcji – serwis repozytorium użytkownika

Metoda serwisu repozytorium użytkownika przekierowuje zapytanie do repozytorium użytkownika.

```
@Query("SELECT u FROM User u WHERE LOWER(u.budgetId) = LOWER(:budgetId)")
public User findByBudget(@Param("budgetId") Long budgetId);
```

Rysunek 29. Wydobywanie transakcji – repozytorium użytkownika

Metoda repozytorium odwołująca się do skonfigurowanej bazy danych. Adnotacja `@Query` określa zapytanie, które zostanie wysłane do bazy danych.

W odpowiedzi kod Angular otrzymuje gotowy obiekt JSON. Po czym dane te są wizualizowane w pliku HTML.

```
<div class="container" id="mainView">
    Name: {{user.firstName}}<br>
    Surname: {{user.lastName}}<br>
    Email: {{user.email}}<br>
    <button class="btn" ng-click="changePassword()" style="float: left;">
        <i class="icon-pencil"></i> Change password
    </button>
</div>
```

Rysunek 30. Użycie danych JSON w pliku HTML.

5. Wykorzystane narzędzie programistyczne

W tym rozdziale przedstawiono narzędzia programistyczne, które były wykorzystywane do prac nad projektem w trakcie jego realizacji. Dobry dobór narzędzi i ich znajomość pozwala znacznie przyspieszyć pracę nad projektem, oraz istotnie wpływa na jakoś wytwarzanego oprogramowania. Podobnie jak w przypadku technologii, autor kierował się kryterium cenowym.

5.1 Eclipse

Eclipse to darmowe zintegrowane środowisko programistyczne. Powstało jako szkielet aplikacyjny napisany w języku Java do tworzenia aplikacji typu rich client w 2004 roku. Dopiero później Eclipse przekształcił się w zintegrowane środowisko programistyczne do tworzenia programów w Javie. Projekt został utworzony przez firmę IBM, a następnie udostępniony na zasadach otwartego oprogramowania. Eclipse jest najpopularniejszym darmowym środowiskiem do tworzenia aplikacji w technologiach opartych na platformie Java. Program ten obsługuje także wtyczki, które pozwalają rozszerzyć funkcjonalność o wsparcie dla innych języków programowania takich jak C++.

Cały kod aplikacji Money w języku Java powstał właśnie przy użyciu Eclipse. Za szczególnie cenne i przydatne funkcjonalności środowiska Eclipse przy tworzeniu aplikacji autor uznał wtyczkę obsługującą narzędzie do budowania Maven – m2eclipse – oraz wbudowane narzędzie służące do debuggowania kodu.

5.2 Webstorm

Webstorm to komercyjne zintegrowane środowisko programistyczne do tworzenia kodu w JavaScript, CSS oraz HTML. Program został stworzony przez firmę JetBrains i jest podzbiorem kluczowego produktu firmy, środowiska IntelliJ IDEA. Webstorm jest obecnie jedynym środowiskiem programistycznym wspierającym szkielet AngularJS poprzez darmową wtyczkę.

Do kluczowych cech produktu należy zaliczyć:

- integrację z systemem kontroli wersji,
- uzupełnianie kodu JavaScript,
- debugger JavaScript,
- wsparcie dla HTML5,
- natywne wsparcie dla szkieletu JavaScript Node.js
- integrację z systemami kontroli zadań.

Kod AngularJS używany w aplikacji Money powstał przy użyciu ww. środowiska. Za szczególnie cenne i przydatne funkcjonalności tego IDE, autor uznał integrację z systemem kontroli wersji oraz uzupełnianie kodu JavaScript.

5.3 Apache Maven

Apache Maven jest narzędziem automatyzującym budowę oprogramowania na platformę Java. Poszczególne funkcjonalności Mavena realizowane są poprzez wtyczki, które są automatycznie pobierane przy ich pierwszym wykorzystaniu. Każdy projekt jest opisywany poprzez specjalny plik XML o nazwie pom.xml.

Rozwinięciem skrótu POM jest z angielskiego Project Object Model. Plik ten zawiera informacje o sposobie budowy aplikacji. Narzędzie Maven automatycznie spełnia zależności określone w pliku POM tj. pobiera wyspecyfikowane biblioteki z publicznych repozytoriów. Każdy taki plik zawiera specyfikację dotyczącą jednego projektu. Projekty złożone są z reguły dzielone na podprojekty z odrębnymi plikami pom.xml.

Typowy układ folderów w projekcie Maven to:

Nazwa katalogu	Opis
katalog projektu	Zawiera plik pom.xml oraz wszystkie podfoldery
src/main/java	Zawiera kod źródłowy projektu Java
src/main/resources	Zawiera pliki zasobów np. skrypty sql, czy pliki konfiguracyjne np. loggerów.

Maven jest rozpowszechniany na licencji Apache License.

W aplikacji Money Maven w sposób znaczący ułatwił proces budowania poprzez zwolnienie programisty z obowiązku dbania o zależności używanych bibliotek. Ponadto, spowodował uniezależnienie od środowiska programistycznego, poprzez zastosowanie typowej struktury katalogów i plików typowej dla Mavena, a nie jakby to miało miejsce bez jego użycia, dla IDE.

5.4 Git

Git jest rozproszonym systemem kontroli wersji. Został stworzony przez twórcę jądra systemu Linux, Linusa Torvaldsa w 2005 roku.

Git z założenia miał być systemem kontroli wersji nadającym się do utrzymywania wielkich rozproszonych projektów programistycznych, takich jak chociażby sam Linux. Oto jego najważniejsze cechy:

- Silne wsparcie dla programowania nieliniowego – Git wspiera rozgałęzianie (ang. branching) i łączenie (ang. merging) kodu, poprzez wizualizowanie i nawigowanie nieliniowej historii.
- Programowani rozproszone – Git dostarcza każdemu programiście pracującemu nad projektem lokalną kopię całej historii rozwoju aplikacji, a zmiany są kopiowane z jednego takiego repozytorium do kolejnego. Zmiany stanowią odrębne gałęzie i mogą być później przyłączane do głównej gałęzi aplikacji.
- Zgodność z istniejącymi systemami i protokołami – Repozytoria mogą być publikowana przy pomocy HTTP, FTP, rsync etc.

Git jest dystrybuowany na zasadach wolnego oprogramowania na licencji GNU GPL w wersji drugiej.

6. Podsumowanie

6.1 Efekt końcowy pracy

Jako rezultat wielomiesięcznej pracy nad projektem powstał program spełniający wymagania użytkownika, biznesowe oraz systemowe wymienione w rozdziale drugim. Gotowa aplikacja nadaje się do śledzenia finansów na przestrzeni określonego czasu a także pozwala na sprawne budżetowanie i planowanie wydatków. Przyczynia się to do zwiększenia świadomości finansowej użytkownika.

Implementacja pozwoliła mi dokładnie poznać wiele nowoczesnych technologii. Dała też doświadczenie, które będę mógł wykorzystać w kolejnych projektach w trakcie mojej dalszej edukacji i kariery zawodowej. Czas poświęcony na tę pracę uświadomił mi, jak istotny jest sam proces projektowy i z pewnością wpłynie na lepsze decyzje projektowe podejmowane w przyszłości. Efekt końcowy uznaję za bardzo zadowalający, a samo doświadczenie oceniam jako bezcenne.

6.2 Możliwe kierunki rozwoju aplikacji

Możliwe kierunki rozwoju programu to:

- Stworzenie odpowiednika programu w postaci aplikacji mobilnej na system Android oraz iOS
- Wczytywanie danych z paragonów na podstawie zdjęć i automatyczna kategoryzacja
- Podpowiadanie cen na podstawie ostatnio wprowadzonych danych
- Podpowiadanie nazw transakcji
- Wyszukiwanie transakcji

7. Spis rysunków

8. Bibliografia

[1] Walls C., *Spring in Action, Third Edition*, Manning, Greenwich, USA, 2011

[2] Dokumentacja szkieletu Spring, http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/

[3] Green B., Seshadri S. *AngularJS*, O'Reilly Media, Sebastopol, USA, 2013

[4] Dokumentacja szkieletu AngularJS, http://docs.angularjs.org/api

[5] Dokumentacja Spring Data JPA, http://static.springsource.org/spring-data/data-jpa/docs/current/reference/html

[6] Sacha K., *Inżynieria oprogramowania*, PWN, Warszawa, 2010

[7] Kainulainen P., *Spring Data*, Packt Publishing, Birmingham, Wielka Brytania, 2012

[8] Wheeler W., White J. *Spring in Practice*, Apress, Nowy Jork, USA, 2013

[9] Farrel J., *Java Programming, Sixth Edition*, Course Technology, Boston, USA, 2011

[10] Basham B., Sierra K., Bates B. *Head First Servlets and JSP, Second Edition*, O'Reilly Media, Sebastopol, USA, 2011

[11] Martin R. C., *Clean Code: A Handbook of Agile Software Craftsmanship*, Prentice Hall, 2008

[12] Kainulainen P., *Spring Data JPA Tutorial*, http://www.petrikainulainen.net/spring-data-jpa-tutorial/, 2012

[13] Dokumentacja szkieletu Bootstrap, http://bootstrapdocs.com/v2.3.1/docs/

[14] Dokumentacja biblioteki angular-charts, https://github.com/chinmaymk/angular-charts