Politechnika Poznańska

Wydział Elektryczny

Instytut Automatyki i Inżynierii Informatycznej

Marcin Wawrzyniak

Praca dyplomowa inżynierska

Tworzenie oraz udostępnianie zbiorów danych poprzez REST API

Promotor: dr Jerzy Bartoszek

Poznań, 2013

STRESZCZENIE

Zaprojektowano oraz zrealizowano aplikację internetową, która umożliwia użytkownikom obsługę płaskich baz danych. Obsługa ta może odbywać się za pomocą przeglądarki internetowej bądź interfejsu REST.

Oprogramowanie po stronie serwera zostało wykonane z wykorzystaniem języka PHP w wersji 5.3, frameworka CakePHP oraz baz danych MySQL i MongoDB.

Interfejs przeglądarkowy korzysta z biblioteki selektorów CSS Twitter Bootstrap oraz języka JavaScript wraz z frameworkiem jQuery.

ABSTRACT

This project assumed the design and creation of a web application which allows the users to create and share data sets in two ways - using web browser and using the REST interface.

Server side software was developed using PHP language (version 5.3), CakePHP framework and MySQL/MongoDB databases.

Web interface makes use of CSS selector library Twitter Bootstrap and JavaScript language (with use of jQuery framework).

Zawartość

1. WPROWADZENIE

Niniejsza praca dotyczy tematyki aplikacji oraz usług internetowych. Oprogramowanie opracowane w ramach pracy dyplomowej jest dostępne przez Internet, przy wykorzystaniu przeglądarki internetowej bądź dowolnego innego klienta HTTP (*ang. Hypertext Transfer Protocol*), np.:

- biblioteka cURL,
- Java: klasa java.net.HttpURLConnection,
- Python: klasa http.client,
- JavaScript: obiekt XMLHttpRequest.

Tym samym aplikacja klasyfikuje się jako aplikacja utrzymująca bazę danych oraz usługa sieciowa.

Celem wykonanego oprogramowania (przy wykorzystaniu metodyki programowania ekstremalnego), było udostępnienie dwóch głównych funkcjonalności:

1. Zarządzanie płaskimi zbiorami danych – poprzez przeglądarkę, bądź REST (*ang. Representational State Transfer*) API (*ang. Application Programming Interface*).
2. Udostępnianie płaskich zbiorów danych darmowo lub odpłatnie (wymagany zakup klucza) – innymi słowy funkcjonalność ta umożliwia dostęp do tworzonych przez użytkowników repozytoriów, za pomocą REST API, lub ich podgląd przez przeglądarkę.

Płaskie zbiory danych to inaczej płaskie struktury danych, składające się z wierszy oraz kolumn. Dane takie zazwyczaj cechują się redundancją. W dalszej części pracy ten typ danych będzie nazywany repozytoriami.

Jak wspomniano wcześniej, aplikacja udostępnia dwa rodzaje interfejsów: przeglądarkowy oraz REST.

Interfejs przeglądarkowy umożliwia rejestrację i logowanie użytkownika, zarządzanie repozytoriami (edycję, usunięcie, modyfikację wierszy) oraz ustalenie zasad upublicznienia repozytorium. Interfejs ten jest przeznaczony dla grupy użytkowników zaznajomionych z działaniem aplikacji internetowych.

Interfejs REST umożliwia odczyt, edycję, dodawanie oraz usuwanie wierszy z repozytorium po uprzedniej autoryzacji odbywającej się za pomocą nagłówka HTTP Authorization (bądź tylko odczyt w przypadku wykorzystania parametru klucza). Interfejs ten jest przeznaczony dla zaawansowanych użytkowników, docelowo programistów chcących wykorzystać udostępniane dane repozytorium jako źródło danych w swoich aplikacjach.

Istnieją przynajmniej dwa zbliżone, znane autorowi, rozwiązania dotyczące tematu pracy.

Pierwszym jest prosta aplikacja WWW APIfy [11] umożliwiająca przetwarzanie istniejących już danych (HTML) za pomocą wyrażeń XPath (*ang. XML Path Language*), oraz tworząca z tych danych repozytorium w formacie JSON, do którego dostęp jest możliwy za pomocą REST API (jedynie operacja odczytu). APIfy to rozwiązanie niekomercyjne, pozwalające na korzystanie z zasobów Internetu. Obrazowym przykładem wykorzystania APIfy jest ten podany przez samych autorów rozwiązania – przetworzenie dokumentu HTML portalu IMDB[1] na odpowiedź w formacie JSON zawierającą listę najpopularniejszych filmów.

Drugim rozwiązaniem jest usługa udostępniająca dane geolokalizacyjne Geo Autocomplete [19], która umożliwia nieodpłatne wykorzystanie Web API z danymi krajów, regionów oraz miast.

Opracowana aplikacja umożliwia zarządzanie (nie tylko odczyt, ale również tworzenie i modyfikację) dowolnymi repozytoriami (składającymi się z kolumn oraz wierszy) oraz udostępnia użytkownikowi możliwość upubliczniania dostępu do tych danych przez REST API (darmowo lub odpłatnie – przy pomocy zakupionego klucza).

Oprogramowanie zostało wykonane z wykorzystaniem języka PHP oraz JavaScript. Wykorzystano również frameworki CakePHP, jQuery oraz Codeception, bazy danych MySQL i MongoDB oraz serwer pamięci podręcznej Memcached. Docelowym systemem operacyjnym, na którym powinno zostać uruchomione oprogramowanie jest Linux bądź Windows. Aaplikację uruchomiono oraz testowano pod systemem Microsoft Windows 7 .

O wyborze tematu zdecydowała popularność oraz zamiłowanie autora do stylu REST tworzenia usług internetowych oraz zainteresowanie oprogramowaniem po stronie serwera.

Treść kolejnych działów omawianej pracy prezentuje się następująco:

Rozdział drugi, „Założenia projektowe” opisuje postawione wymagania, zastosowane standardy oraz podejście do ochrony danych. Rozdział ten opisuje również wzorce takie jak MVC, MVP, REST.

Rozdział trzeci, „Szczegóły projektu” przedstawia architekturę systemu.

Rozdział czwarty, „Wykorzystane narzędzia” przedstawia technologie wykorzystywane przy programowaniu aplikacji.

Rozdział piąty, „Implementacja” omawia aspekty implementacyjne..

[1] Imdb.com, portal udostępniający recenzje filmów wideo

Rozdział szósty, „Testy” opisuje przeprowadzone testy akceptacyjne oraz obciążeniowe.

Rozdział siódmy, „Wnioski” podsumowuje projekt oraz opisuje ewentualne kierunki rozwoju.

2. ZAŁOŻENIA PROJEKTOWE

Założeniem pracy jest wykonanie działającej aplikacji WWW umożliwiającej użytkownikowi tworzenie i zarządzanie repozytorium danych (wierszami, kolumnami), oraz udostępnienie go przez REST API oraz przeglądarkę internetową.

Aplikacja powinna działać w oparciu o popularne technologie, m.in. PHP, JavaScript, CakePHP, jQuery, Memcached MySQL oraz MongoDB.

Zakłada się istnienie trzech poziomów ról dostępu. Poziomy te zostały opisane szczegółowo w rozdziale „Szczegóły projektu", dlatego poniższy akapit zawiera jedynie nazwy ról z ich krótkim opisem:

1. Rola „Gość" – jest to użytkownik niezalogowany, posiadający ograniczony dostęp do interfejsu przeglądarkowego oraz dostęp do darmowych repozytoriów poprzez REST API
2. Rola „Użytkownik" – jest to użytkownik zalogowany (w przypadku przeglądarki), bądź identyfikujący się parametrem klucza (w przypadku dostępu do repozytorium płatnego)
3. Rola „Administrator" – jest to rozszerzenie roli „Użytkownik" o pełny dostęp (odczyt, modyfikację) do utworzonych przez „administratora" repozytoriów, zarówno w przypadku przeglądarki jak i REST API.

Zakłada się, że aplikacja będzie umożliwiała:

1. Rejestrację użytkownika oraz logowanie (zarówno rejestracja jak i logowanie odbywają się poprzez podanie adresu email oraz hasła).
2. Tworzenie repozytoriów oraz wierszy danych (darmowych, płatnych).
3. Import repozytoriów (pliki w formacie CSV).
4. Dokumentację użytkową dostępną z poziomu aplikacji.
5. Symulację płatności internetowych.
6. Dostęp do wierszy repozytorium utworzonych przez użytkownika za pomocą REST API (rola „użytkownik REST").
7. Modyfikowanie utworzonych danych za pomocą REST API (rola „administrator").
8. Dodanie repozytorium, do listy „ulubionych" repozytoriów poprzez przeglądarkę WWW.
9. Licznik wyświetleń „strony wizytówki", prezentującej dane o repozytorium takie jak nazwa, pierwsze 100 wierszy, informacje o typie repozytorium (darmowym lub płatnym).

Aplikacja jest przeznaczona dla dwóch grup odbiorców. Pierwszą grupą są użytkownicy chcący tworzyć repozytoria, zapewniać ich aktualność (poprzez zarządzanie wierszami) oraz udostępniający repozytoria odpłatnie lub darmowo. Drugą grupą są użytkownicy korzystający z utworzonych w ten sposób repozytoriów za pomocą REST API

(przykładowo z poziomu opracowanej przez nich aplikacji, poprzez wykorzystanie odpowiednich sposobów dostępu).

Przyjęto, że interfejs opracowanej aplikacji internetowej musi:

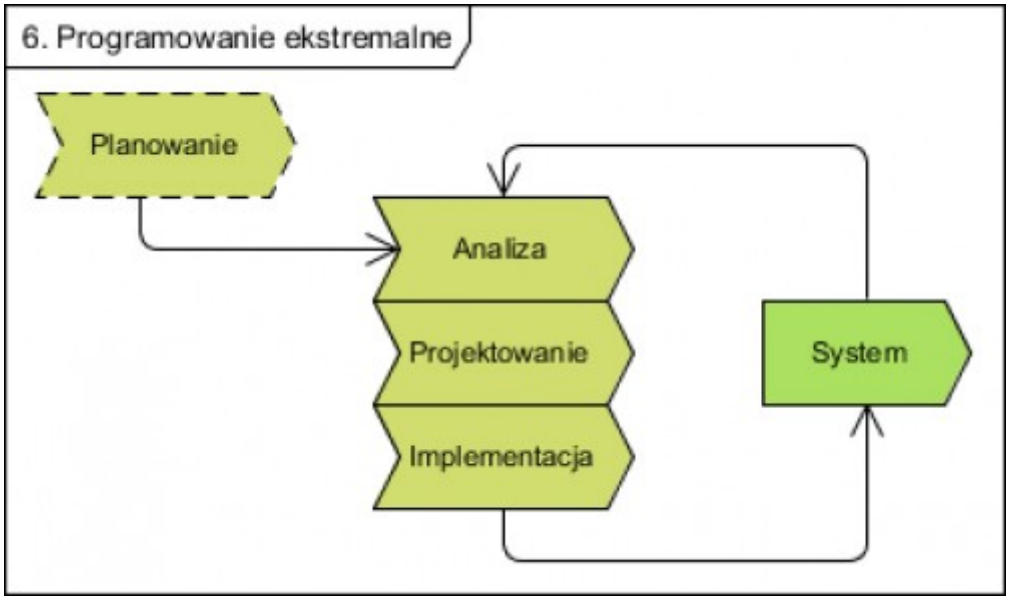
1. używać biblioteki CSS Twitter Bootstrap,
2. być używalny na urządzeniach mobilnych (tablety).

Motywowane było to tym, że urządzenia mobilne wciąż zyskują na popularności, natomiast biblioteka Twitter Bootstrap dostarcza elegancki oraz ujednolicony interfejs.

2.1. Metodyka

Przyjęto, że przy opracowywaniu aplikacji zastosowana zostanie metodyka programowania ekstremalnego (*ang. Extreme Programming*) [3].

Rysunek 2.1 prezentuje podstawowy cykl występujący w programowaniu ekstremalnym.



Rys. 2.1. Programowanie ekstremalne [25]

W rzeczywistości, prace nad oprogramowaniem miały też charakter iteracyjny, a architektura aplikacji została zmieniona w porównaniu z początkowymi założeniami:

- W pierwszej iteracji zaprojektowano później odrzucony schemat bazy danych MySQL - ponieważ zakładał użycie dodatkowych tabel do przechowywania danych oraz nakładał zbędną warstwę abstrakcji na płaski zbiór danych,
- Iteracja druga obejmowała instalację bazy MongoDB oraz zaprojektowanie najlepszego schematu składowania dokumentów,

- Iteracja trzecia polegała na opracowaniu modeli MongoDB oraz uproszczonym projekcie bazy MySQL wraz z modelami tabel bazy danych,
- Iteracja czwarta polegała na oprogramowaniu systemu oraz komponentu DataTables (UI),
- Iteracja piąta wprowadzała warstwę abstrakcji dla DataTables oraz zmianę architektoniczną - RestController w iteracji piątej jest generalizacją DatatablesController, ponieważ w iteracji czwartej były to niezależne od siebie klasy współdzielące zestaw funkcjonalności,
- Iteracja szósta polegała na zmianach w interfejsie,
- Iteracja siódma to głównie poprawki błędów oraz zmiana w nazwach kluczy memcached (zmieniono separator z kropki na ukośnik).

2.2. Wzorzec MVC oraz MVP

Przyjęto, że aplikacja będzie budowana zgodnie z wzorcem projektowym MVC, który implementuje framework CakePHP.

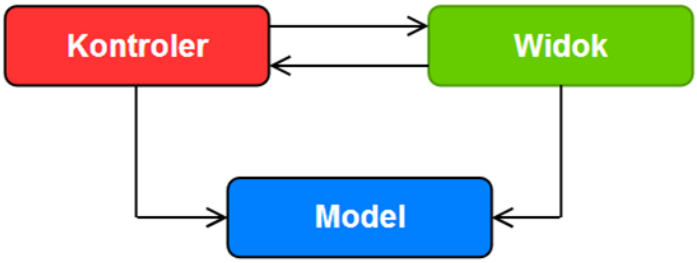
MVC (*ang. Model-View-Controller*) to wzorzec projektowy, którego korzenie sięgają języka Smalltalk (1979) [10]. Ze względu na specyfikę środowiska aplikacji WWW wzorzec MVC (pokazany na rys. 1) jest wzorcem popularnie stosowanym podczas tworzenia aplikacji WWW oraz GUI (*ang. Graphical User Interface*) aplikacji jednostanowiskowych.

MVC określa separację trzech podstawowych komponentów:

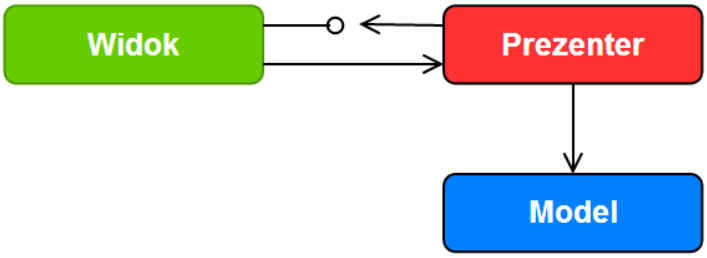
- Modelu danych – np. modelu bazy danych, usługi sieciowej,
- Widoku (warstwy prezentacji) – np. kod HTML, JSON, XML,
- Kontrolera (logiki aplikacji).

MVP (*ang. Model View Presenter)* to uogólnienie wzorca MVC opracowane w latach 90” dla systemu operacyjnego Taligent [45].

Główna różnica pomiędzy MVC a MVP (rys. 2.2, rys. 2.3) to fakt, że w MVP warstwa Widoku jest aktualizowana wyłącznie przez warstwę Prezentera (natomiast w MVC warstwa Widoku ma bezpośredni dostęp do danych Modelu, więc komunikacja pomiędzy warstwami Widoku oraz Kontrolera jest ograniczona) [28].



Rys. 2.2. Wzorzec MVC



Rys. 2.3. Wzorzec MVP

2.3. Wzorzec architektoniczny REST

REST jest często wybieraną – obok SOAP (*ang. Simple Object Access Protocol*) - metodą implementacji usług sieciowych (*ang. Web service*).

Roy Fielding, w swojej pracy doktorskiej [44] jako pierwszy użył terminu REST oraz opisał ten styl architektoniczny jako styl spełniający minimalnie poniższe wymogi:

- architektura klient–serwer (*ang. Client-Server model*),
- bezstanowość (*ang. Stateless*) - wszystkie dane potrzebne do zidentyfikowania sesji użytkownika powinny być zawarte w zapytaniu, a nie przechowywane po stronie serwera,
- buforowanie podręczne (*ang. Cache*) - oznacza, że dane zawarte w odpowiedzi na zapytanie muszą zostać jawnie, bądź nie jawnie oznaczone jako mogące podlegać zapisowi do pamięci podręcznej,
- jednolity interfejs dostępu (*ang. Uniform Interface*) – jednolity interfejs dostępu pomiędzy komponentami zaprojektowany dla sieci WWW,
- system warstwowy (*ang. Layered System*) – pozwala architekturze składać się z warstw ułożonych hierarchicznie, ze względu na co każdy komponent systemu „widzi” tylko jedną warstwę, a nie warstwy z którymi przeprowadzane są dalsze interakcje.

REST jest często wybieranym rozwiązaniem ze względu na jego implementację na szczycie protokołu HTTP, brak konieczności przetwarzania dokumentów XML oraz

mniejszy narzut obliczeniowy (nie wymaga jak w przypadku SOAP wykorzystania koperty [41] (*ang. Envelope*)) [1].

REST używa metod dostępu takich samych jak http (w pracy wykorzystano poniższe cztery) [40]:

- GET (pobranie wierszy repozytorium, bądź pojedynczego wiersza),
- PUT (aktualizacja bądź utworzenie wiersza repozytorium),
- POST (utworzenie wiersza repozytorium),
- DELETE (usunięcie wiersza repozytorium).

2.4. Ochrona danych

Identyfikatory użytkowników są tożsame z adresami e-mail użytkowników i są wykorzystywane jedynie w celu:

1. rejestracji,
2. uwierzytelnienia,
3. wysłania wiadomości w przypadku zakupienia klucza dostępu do repozytorium.

Klucze dostępu do repozytorium są generowane przy wykorzystaniu skrótu SHA1 oraz funkcji PHP uniqid() [32] i są w takiej postaci przechowywane w bazie danych MySQL. Dostęp do kluczy jest możliwy jedynie dla zalogowanych użytkowników z poziomu przeglądarki.

Hasła użytkowników są przechowywane w postaci skrótów obliczanych przez funkcję skrótu SHA-1 [37] z dodatkiem soli[2] (*ang. Salt)*.

2.5. Standardy PHP

Kod opracowanej aplikacji powinien spełniać standardy:

- PSR-0 – określający standardy automatycznego ładowania klas w PHP [21] [33],
- PSR-1 – określający standardy kodowania (plików) oraz nazewnictwa [22],
- PSR-2 – rozszerza standardy kodowania określone w PSR-1 (np. maksymalna długość linii) [23].

2.6. Środowisko uruchomieniowe

Zaleca się aby system został uruchomiony w środowisku LAMP (akronim od Linux, Apache, MySQL, PHP). Wymagana jest również instalacja Memcached w wersji co najmniej 1.4.15 oraz MongoDB w wersji minimum 2.2.1.

[2] Sól - inaczej ciąg zaburzający

Zalecane wersje oraz moduły oprogramowania prezentują się następująco:

1. Apache w wersji co najmniej 2.2.22 (mod_rewrite),
2. MySQL w wersji minimum 5.5.24,
3. PHP w wersji przynajmniej 5.3.13 (php_mongo, php_memcache).

3. SZCZEGÓŁY PROJEKTU

Rozdział ten opisuje elementy tworzonego systemu, bazy danych oraz użyte narzędzia w celu dostarczenia działającej aplikacji internetowej.

Ponieważ stan aplikacji jest determinowany przez sposób dostępu (przeglądarka, REST API), stan i typ repozytoriów (publiczne, prywatne; darmowe, płatne), stan użytkownika (zalogowany lub niezalogowany), konieczne było określenie ról dostępu dla interfejsu przeglądarkowego (tabela 3.1) oraz REST (tabela 3.2).

Tabela 3.1. Uprawnienia użytkowników interfejsu przeglądarkowego

Nazwa roli	Dostęp do interfejsu przeglądarkowego	Dostęp do repozytorium darmowego	Dostęp do repozytorium płatnego
Gość	Tak (ograniczony)	Tak	Nie
Użytkownik	Tak	Tak	Tak (przy wykorzystaniu klucza)
Administrator (właściciel repozytorium)	Tak	Tak	Tak (nagłówek Authorization)

„Gościem” nazwano użytkownika niezalogowanego. „Użytkownik” to użytkownik uwierzytelniony za pomocą hasła oraz adresu email. „Administrator”, to użytkownik zalogowany oraz właściciel określonego repozytorium.

W interfejsie REST uwzględniono różne uprawnienia użytkowników. Pokazano je w tabeli 2.

Tabela 3.2. Uprawnienia użytkowników interfejsu REST

Nazwa roli	Posiada dostęp do Interfejsu REST	Posiada dostęp do operacji odczytu REST	Posiada dostęp do operacji modyfikacji REST
Gość	Tak	Tak (tylko darmowe repozytorium)	Nie
Użytkownik	Tak	Tak (darmowe oraz płatne repozytorium za pomocą klucza)	Nie
Administrator (właściciel repozytorium)	Tak	Tak	Tak

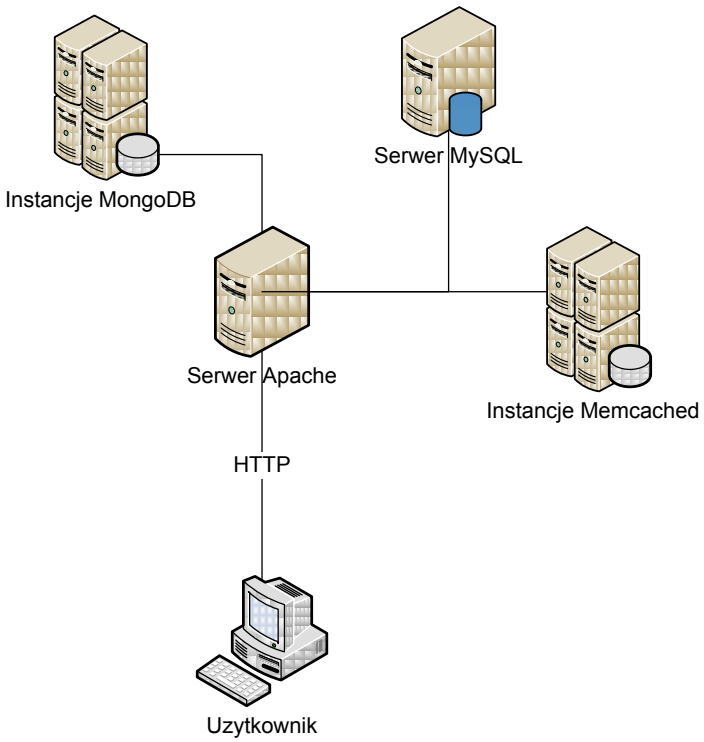
„Gościem” nazwano użytkownika, który przy pomocy interfejsu REST próbuje uzyskać dane z repozytorium płatnego bez klucza (bądź z błędnym kluczem) lub nagłówka Authorization.

„Użytkownikiem” nazwano użytkownika, który próbuje uzyskać dane z repozytorium płatnego z podanym poprawnym kluczem.

„Administratorem” nazwano użytkownika, który próbuje edytować bądź odczytać dane repozytorium przy pomocy poprawnego nagłówka Authorization.

3.1. Fizyczna architektura systemu

Na rysunku 3.3 przedstawiono fizyczną architekturę omawianego systemu, czyli połączenie serwera Apache z instancjami bazy MongoDB, jedną instancją bazy MySQL oraz instancjami pamięci podręcznej Memcached.

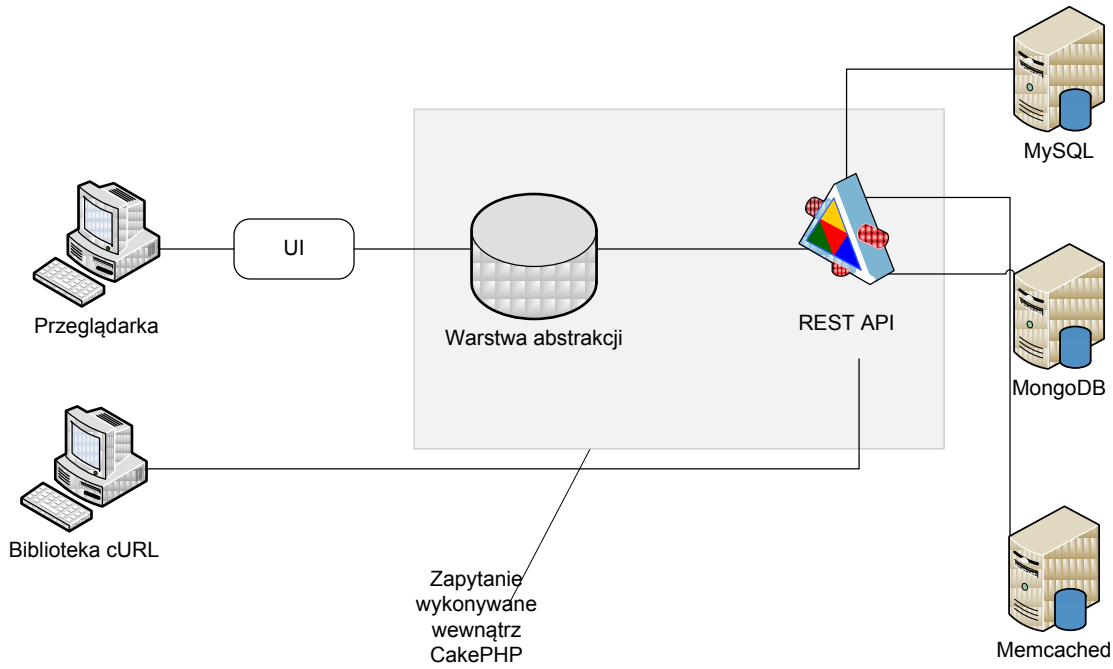


Rys. 3.3. Schemat fizyczny architektury

Na rysunku 3.4 przedstawiono elementy systemu uczestniczące w przetwarzaniu zapytania. W przypadku użycia przeglądarki, komunikuje się ona z warstwą abstrakcji REST API, która jest pośrednio (poprzez warstwę API) odpowiedzialna za połączenie z bazami danych oraz pamięcią podręczną.

Warstwa abstrakcji jest wymagana ze względu na specyficzne wymagania komponentu DataTables (parametry HTTP wysyłane oraz zwracane, np. parametr „sEcho”), co oznacza iż nie może on bezpośrednio korzystać z warstwy REST API.

W drugim przypadku, gdy zapytanie zostanie wykonane z poziomu klienta HTTP, nie używa się UI (*ang. User Interface*) - oraz warstwy abstrakcji (dla interfejsu), a więc zapytania są kierowane bezpośrednio do warstwy REST API.

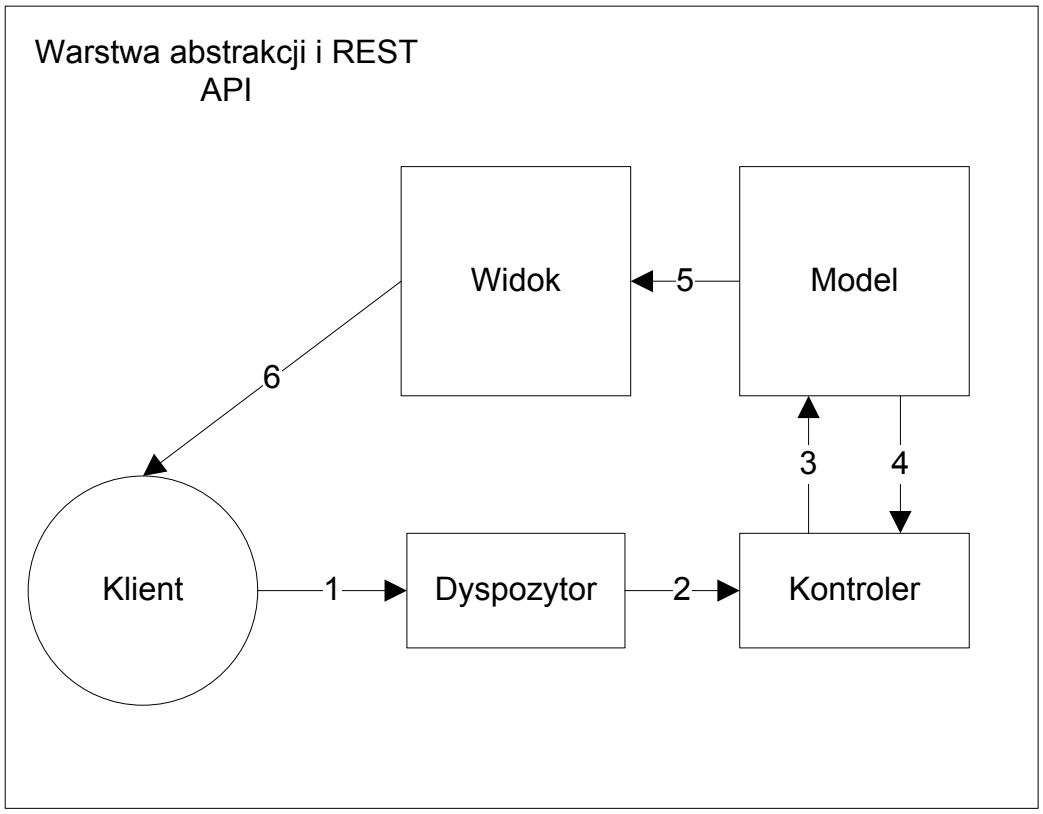


Rys. 3.4. Schemat logiczny architektury

Gdy serwer Apache otrzyma zapytanie oraz wywołany zostanie interpreter PHP, jest ono przetwarzane przez aplikację opracowaną z wykorzystaniem CakePHP.

Typowy przepływ zapytania we wnętrzu CakePHP przedstawiono na rysunku 3.5. Przedstawiono na nim poszczególne etapy jakie musi przejść zapytanie wysłane przez klienta, zanim odesłana zostanie odpowiedź.

Dyspozytor (*ang. Dispatcher)*, będący integralną częścią Framework CakePHP decyduje na podstawie trasowania (*ang. routing)* do jakiej akcji warstwy kontrolera wysłać zapytanie, następnie warstwa kontrolera komunikuje się z warstwą modelu (bazami danych, pamięcią podręczną) oraz przekazuje dane do warstwy widoku, który jest zwracany użytkownikowi w postaci dokumentu HTML/JSON.



Rys. 3.5. Przepływ zapytania (MVC) w CakePHP [12]

3.2. Pamięć podręczna Memcached

Memcached to serwer umożliwiający przechowywanie w pamięci podręcznej danych takich, jak rekordy bazy danych, fragmenty HTML czy obiekty. Dzięki korzystaniu z pamięci podręcznej można ograniczyć opóźnienia wynikające z długiego czasu dostępu do dysku [5].

3.3. Protokół HTTP

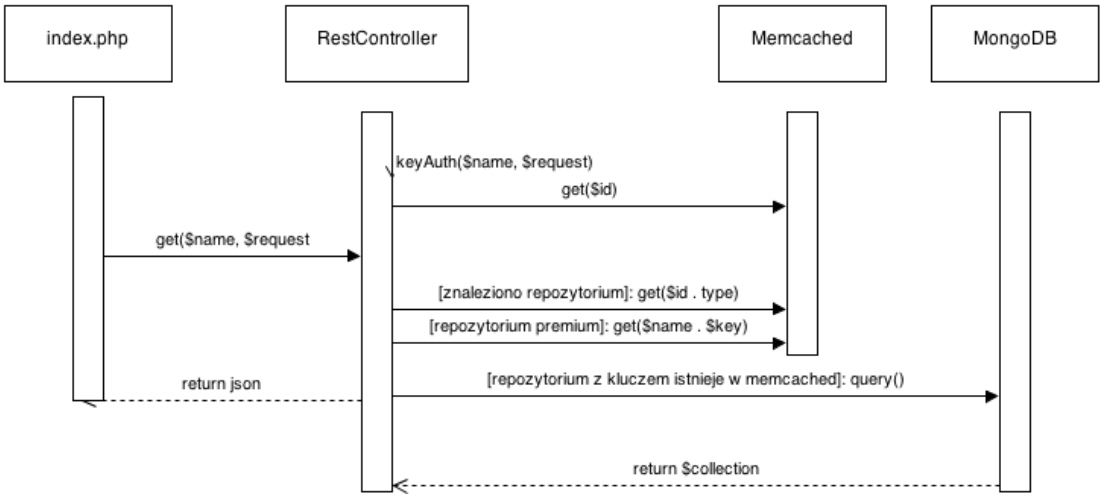
REST API zakłada użycie poniższych czterech metod:

1. GET,
2. POST,
3. PUT,
4. DELETE.

Najczęściej wykonywanym zapytaniem w omawianej aplikacji jest zapytanie typu GET, wysyłane na adres konkretnego repozytorium.

3.4. Diagram sekwencji zapytania GET

Rysunek 3.6 przepływ demonstruje przepływ zapytania GET w aplikacji. Zapytanie GET wysłane na adres konkretnego repozytorium jest kierowane do kontrolera (klasy) RestController, a konkretniej do jego akcji (metody) get().



Rys. 3.6. Diagram sekwencji RestController::get()

Wywołanie metody RestController::get() nie skutkuje nawiązaniem połączenia z serwerem MySQL, dzięki czemu zniwelowane zostało opóźnienie wynikłe z tytułu nawiązania połączenia, oraz przesyłu danych z MySQL do aplikacji. Zamiast tego nawiązywane jest połączenie z serwerem Memcached oraz MongoDB.

Dzięki wykorzystaniu pamięci podręcznej Memcached, możliwe jest bez połączenia z bazą MySQL odczytanie następujących danych:

- id (odpowiadające kluczowi głównemu tabeli feeds bazy MySQL) na podstawie nazwy repozytorium,
- typ repozytorium,
- czy zakończono edycję repozytorium,
- informacji na temat wykupionych kluczy.

3.5. Baza danych

Na rysunku 3.7 przedstawiono model bazy danych MySQL używany przez aplikację.

Utworzono pięć tabel:

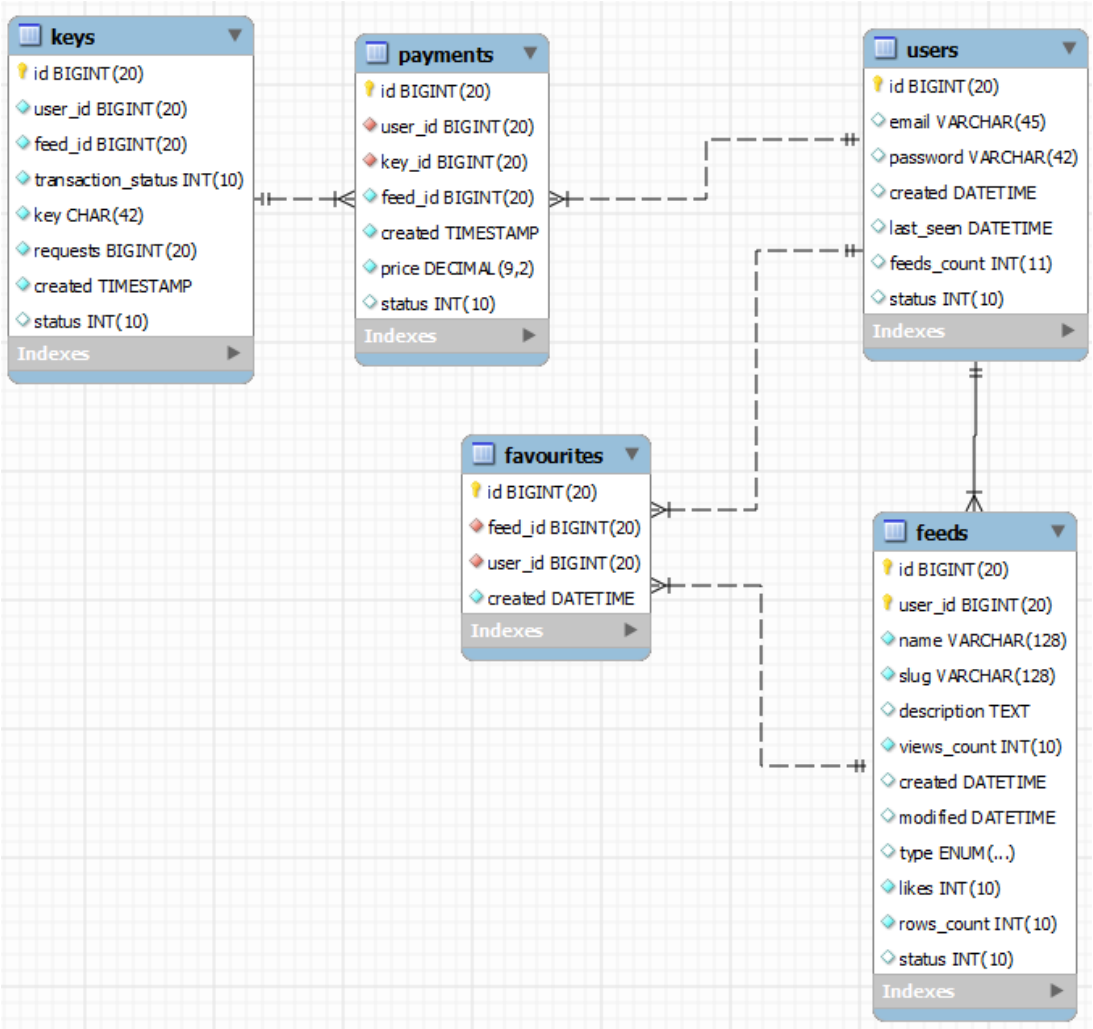
1. keys
   - id – klucz główny typu BIGINT
   - user_id - indeks, klucz obcy typu BIGINT,

- feed_id - indeks, klucz obcy typu BIGINT,
- transaction_status - typ INT
- key - typ CHAR
- requests - typ BIGINT
- created - typ TIMESTAMP
- status - typ INT

2. payments
   - id – klucz główny typu BIGINT,
   - user_id – indeks, klucz obcy typu BIGINT,
   - key_id - indeks, klucz obcy typu BIGINT,
   - feed_id - indeks, klucz obcy typu BIGINT,
   - created – typ TIMESTAMP
   - price – typ DECIMAL(9,2)
   - status – typ INT
3. users
   - id - klucz główny typu BIGINT,
   - email – typ VARCHAR,
   - password – typ VARCHAR,
   - created – typ DATETIME,
   - last_seen – typ DATETIME,
   - feeds_count – typ INT,
   - status – typ INT
4. favourites
   - id - klucz główny typu BIGINT,
   - feed_id - indeks, klucz obcy typu BIGINT,
   - user_id - indeks, klucz obcy typu BIGINT,
   - crated – typ DATETIME
5. feeds
   - id - klucz główny typu BIGINT,
   - user_id - indeks, klucz obcy typu BIGINT,
   - name – typ VARCHAR,
   - slug – typ VARCHAR,
   - description – typ TEXT,
   - views_count – typ INT,
   - created – typ DATETIME,
   - modified – typ DATETIME,
   - type – typ ENUM
   - likes – typ INT,
   - rows_count – typ INT,
   - status – typ INT

Jako typ danych klucza głównego zastosowano typ SQL BIGINT z uwagi na to, że może pomieścić dwukrotnie więcej bajtów w porównaniu do typu SQL INT − 8 bajtów zamiast 4 bajtów, a więc może pomieścić łącznie 18,446,744,073,709,551,615 wartości. Zastosowano również klucze obce pozwalające utrzymać spójność bazy danych.

Do przechowywania danej o wielkości płatności, użyto typu SQL DECIMAL(9,2) (precyzja do dwóch liczb po przecinku) zamiast FLOAT ponieważ, liczby typu SQL FLOAT reprezentują liczby zmiennoprzecinkowe w sposób binarny (liczby w systemie dwójkowym można zapisać ze stuprocentową dokładnością, w przeciwieństwie do liczb systemu dziesiętnego), przez co następuje zaokrąglenie oraz utrata dokładności wartości.



Rys. 3.7. Schemat bazy danych

3.6. Format danych JSON

JSON (ang. *JavaScript Object Notation*) [38] to kompaktowy, tekstowy format wymiany danych, związany z językiem JavaScript.

Format JSON może reprezentować cztery podstawowe typy danych:

- ciągi znaków (*ang. String)* - sekwencje zera, lub więcej znaków Unicode,
- liczby (*ang. Number)*,
- wartości Boolowskie (*ang. Boolean)* – wartości „true” lub „false”,
- typ null (*ang. Null),*

oraz dwa typy strukturalne:

- obiekty - nieuporządkowane kolekcje zera, lub więcej par nazwa/wartość, gdzie nazwa to ciąg znaków, a wartość to ciąg znaków, liczba, wartość boolowska, null, obiekt lub tablica,
- tablice - uporządkowane sekwencje zera lub więcej wartości.

Ponieważ JSON stanowi składniowy podzbiór języka JavaScript, przetwarzanie danych w tym formacie jest bardziej naturalne z poziomu języka JavaScript, niż danych w formacie XML.

Format JSON jest często używany w połączeniu z nowoczesnymi usługami sieciowymi, m.in. typu REST, oraz aplikacjami internetowymi używającymi techniki AJAX.

Standard ECMAScript 5 [18] wprowadza metodę JSON.parse(), która jest zalecaną metodą parsowania danych formatu JSON (obok odradzanej funkcji eval() podatnej na ataki wstrzyknięcia kodu).

Wydruk 3.1 ilustruje zależność między nazwami kolumn w bazie danych (oraz ich domyślnymi wartościami) i wierszami, a schematem dokumentu JSON (por. punkt 4.6). Założono, że użytkownik utworzył repozytorium składające się z wierszy o kolumnach:

- name
- year.

Na wydruku widać iż użytkownik pośrednio (poprzez przeglądarkę) definiuje schemat dokumentu JSON. Wydruk pokazuje dwie kolekcje dokumentów JSON: rows oraz schemas. Pierwsza kolekcja reprezentuje zawartość wierszy, natomiast druga ich schemat ustalony przez użytkownika.

```
rows: {
      _id: ObjectId,
      feed_id: Int32,
      name:“Ford Mustang”,
      year: 1967,
      type: “cabrio”
},
schemas: {
      _id: Int32,
      name: [],
      year: [],
      type: [„cabrio”,”sedan”]
}
```

Wydruk 3.1: Powtarzalna struktura dokumentów MongoDB

Klucze kolekcji „schemas” odpowiadają nazwom pól, definiowanym przez użytkownika, natomiast ich wartości reprezentują wartości domyślne danego pola, również podane przez użytkownika – zapis [1 … 10] oznacza tablicę maksymalnie 10 elementową (ograniczenie do 10 elementów jest czysto losowe).

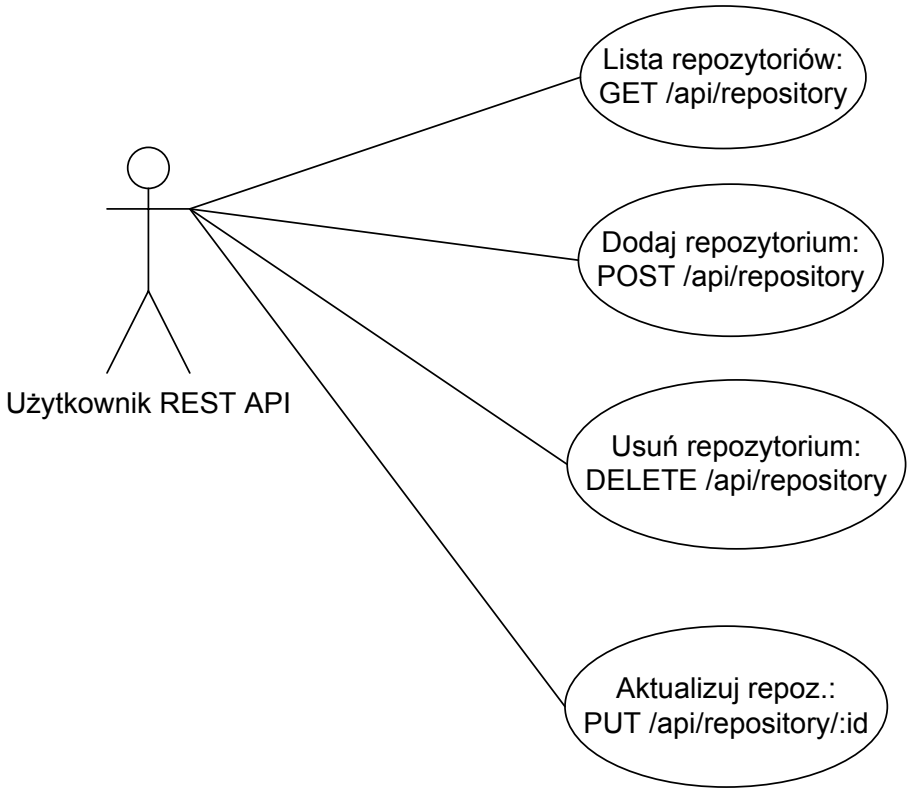
Kolekcja „rows”, oprócz standardowego pola _id oraz zawsze obecnego pola feed_id, zawiera również pola definiujące nazwy pól, wprowadzone przez użytkownika (co oznacza, że użytkownik pośrednio tworzy schemat dokumentu kolekcji MongoDB).

Kolekcja „rows” zawiera wiersze danych wprowadzonych przez użytkownika podczas edycji repozytorium bądź wywołania zapytania PUT/POST poprzez REST API.

Kolekcja „schemas” oprócz pola _id które odpowiada kluczowi głównemu tabeli Feeds w bazie MySQL, zawiera również pola tożsame z nazwami pól, zdefiniowanych przez użytkownika (jak w przypadku kolekcji „rows”), jednak w tym przypadku pola te, nie zawierają wartości, a 10 elementowe tablice z wartościami domyślnymi (użytkownik może podać 10 wartości domyślnych danego pola).

3.7. Diagramy przypadków użycia

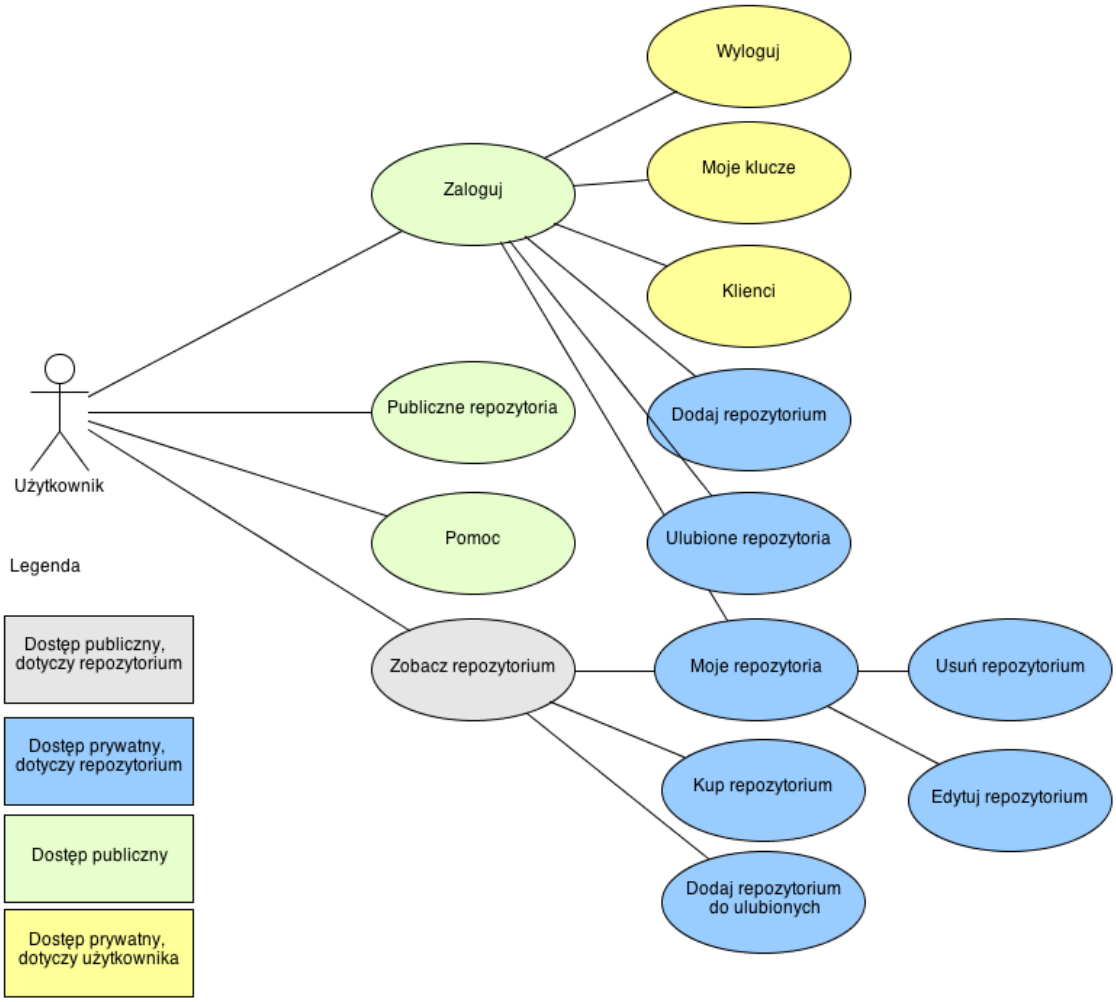
Przedstawione na rysunku 3.8 oraz 3.9 diagramy przypadków użycia mają za zadanie przedstawić możliwy sposób korzystania z omawianej aplikacji.



Rys. 3.8. Diagram przypadków użycia (użytkownik API)

Na rysunku 3.8 pokazano możliwości jakie posiada „użytkownik REST API” – dotyczą one jedynie korzystania z REST API.

Na rysunku 3.9 natomiast przedstawiono możliwości użytkownika korzystającego z przeglądarki.



Rys. 3.9. Diagram przypadków użycia (użytkownik)

4. WYKORZYSTANE NARZĘDZIA

4.1. Bazy danych MySQL oraz MongoDB

MySQL (*ang. My Structured Query Language*) został wybrany ze względu na wieloplatformowość oraz popularność.

Baza relacyjna MySQL zakłada przechowywanie w tabelach informacji o:

- użytkownikach (hasła, adresy email),
- repozytoriach (nazwa, opis),
- kluczach (wartość klucza),
- płatnościach (ID użytkownika, ID klucza, ID repozytorium),
- ulubionych repozytoriach (ID użytkownika, ID repozytorium).

Oprócz MySQL użyto nierelacyjnej bazy MongoDB typu NoSQL (*ang. Not Only SQL*) [29], w której przechowywane są dane o:

- wierszach repozytorium,
- nazwach kolumn repozytorium,
- wartościach domyślnych kolumn w repozytorium.

MongoDB to skalowalna, wysoko wydajna baza danych NoSQL o otwartym kodzie źródłowym. Napisana w C++, udostępnia szereg funkcjonalności, jak na przykład [27]:

1. składowanie dokumentów w formacie JSON ,
2. indeksowanie,
3. replikacja[3],

MongoDB nie jest bazą relacyjną, jest natomiast bazą zorientowaną na dokumenty JSON. Dokument JSON składa się z pól oraz wartości, a zbiór dokumentów JSON jest nazywany kolekcją. MongoDB wspiera skalowanie horyzontalne (zwiększanie mocy obliczeniowej poprzez dodawanie kolejnych serwerów) dzięki partycjonowaniu poziomemu (*ang. Sharding*) [4].

Bazy zwanymi NoSQL są często uważane za niewymagające schematu. Niemniej jednak schemat taki jest wartościową częścią dokumentu, ponieważ w przypadku omawianej pracy schemat stworzony przez użytkownika jest w pewnym stopniu powtarzalny.

[3] Jest to replikacja typu Master/Slave, Master wykonuje odczyty i zapisy, Slave kopiuje dane z instancji Master, i może wykonywać jedynie odczyty.

4.2. Komponent DataTables

Do prezentacji danych tabelarycznych oraz umożliwienia ich wygodnej edycji użyto komponentu DataTables [17]. Umożliwia on prezentację repozytoriów utworzonych przez użytkownika, sortowanie ich oraz edycję i filtrowanie danych.

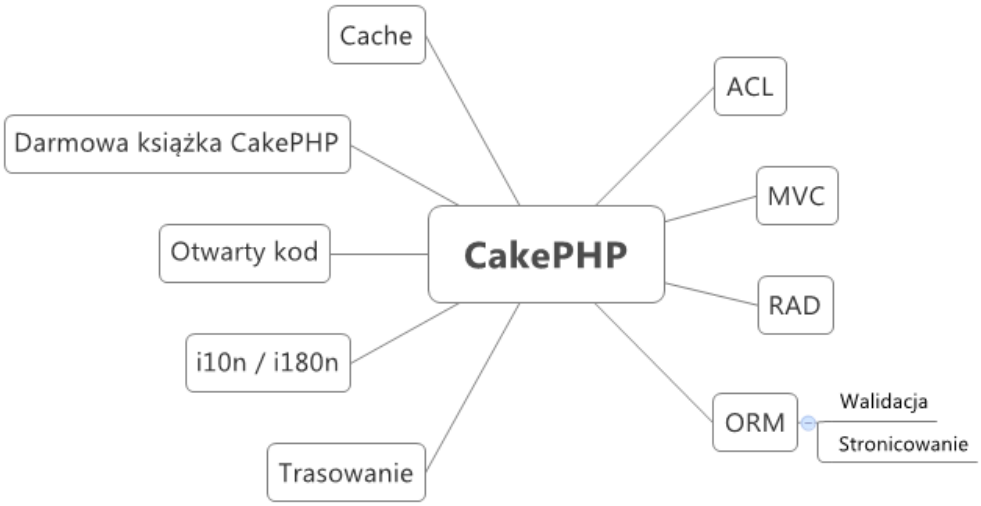
4.3. Frameworki

„Zadaniem frameworka jest dostarczenie szkieletu do tworzenia aplikacji (lub pewnej jej części), a także zbioru ogólnych funkcjonalności, które programista może rozszerzać, by dopasować je do konkretnych potrzeb. Framework może być rozumiany zatem jako szczególny przypadek biblioteki programistycznej” [34].

Część oprogramowania uruchamiana po stronie serwera Apache, została wykonana z użyciem frameworka CakePHP. Jest to framework implementujący wzorzec MVC.

4.3.1. Framework CakePHP

CakePHP to framework aplikacji internetowych dla języka PHP korzystający z wzorca MVC i przez podobny do frameworka Ruby on Rails [13]. Na rysunku 4.1, zademonstrowano główne mechanizmy CakePHP.



Rys. 4.1. Mechanizmy CakePHP

- RAD (*ang. Rapid Application Development*), to mechanizm umożliwiający szybkie pisanie oprogramowania,
- ACL (*ang. Access Control List*) to lista zezwoleń dostępu, przypisana do użytkownika oraz klasy kontrolera (a nawet akcji). Określa którzy użytkownicy systemu mają dostęp do danej akcji kontrolera, oraz jakiego typu jest to dostęp,

- L10n/I18n (skrót od *ang. Localization* oraz *ang. Internationalization*), możliwość obsługi komunikatów, dokumentacji, oraz danych (czas, wartości) w formatach odpowiednich dla danego obszaru kulturowego,
- ORM (*ang. Object-relational mapping*) – modelowanie obiektowo-relacyjne pozwala na użycie podejścia obiektowego podczas pracy na wierszach bazy danych, dzięki czemu nie ma potrzeby pisania zapytań SQL, oraz zapewniana jest ochrona przed atakami na bazę danych (np. SQL Injection [30])
- MVC – framework CakePHP został w całości oparty na wzorcu MVC,
- System szablonów – jako system szablonów CakePHP domyślnie wykorzystuje język PHP
- Trasowanie (*ang. Routing*) – to funkcjonalność dzięki której można elastycznie odwzorować adresy URL na akcje kontrolerów (w modelu MVC).

CakePHP został wybrany ze względu na posiadane doświadczenie autora przy pisaniu aplikacji internetowych z użyciem tego narzędzia.

4.3.2. Framework jQuery

Po stronie klienta został wdrożony framework jQuery (udział rynku 89.6%) [42]. jQuery to wydajna oraz spójna biblioteka, która ułatwia manipulację elementami drzewa DOM, obsługę zdarzeń (*ang. Events*), animacje oraz umożliwia wykonywanie zapytań AJAX (*ang. Asynchronous JavaScript and XML*) [26].

4.3.3. Framework Codeception

Aby zautomatyzować proces testowania oprogramowania użyto testów akceptacyjnych, oraz frameworka Codeception. Uzasadnieniem wyboru Codeception jest czytelny (dla osób nie związanych z programowaniem) sposób pisania testów (jak pokazano na wydruku 4.1). Oprócz testów akceptacyjnych, framework ten umożliwia programowanie testów funkcjonalnych oraz jednostkowych [16].

```
$I = new ApiGuy($scenario);

$I->wantTo('fetch single car');

$I->sendGET('/rest/get/cars/21');

$I->seeResponseCodeIs(200);

$I->seeResponseIsJson();
```

Wydruk 4.1. Kod testu Codeception

4.3.4. Framework Twitter Bootstrap

Twitter Bootstrap to popularna biblioteka znaczników CSS definiująca wygląd aplikacji internetowej [39].

4.4. System kontroli wersji Subversion

Subversion to darmowy system kontroli wersji. Pozwala na wygodne śledzenie zmian w kodzie, umożliwia przejrzenie historii edytowanego kodu wraz z adnotacjami o tym, kto dokonał edycji.

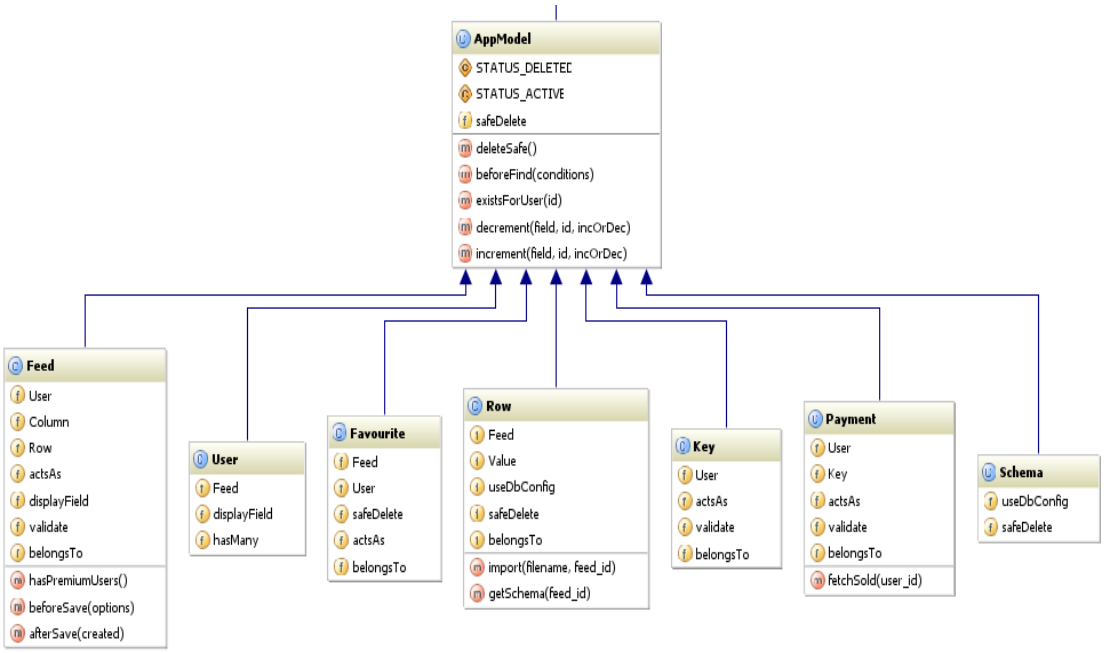
5. IMPLEMENTACJA

Rozdział ten opisuje implementację opracowanego systemu. Rozdział kończy instrukcja instalacji oprogramowania oraz podsumowanie.

5.1. Diagram klas

Rysunek 5.1 przedstawia diagram klas (tabel bazy danych) oraz kontrolerów (logiki aplikacji) oraz opisuje jego charakterystykę.

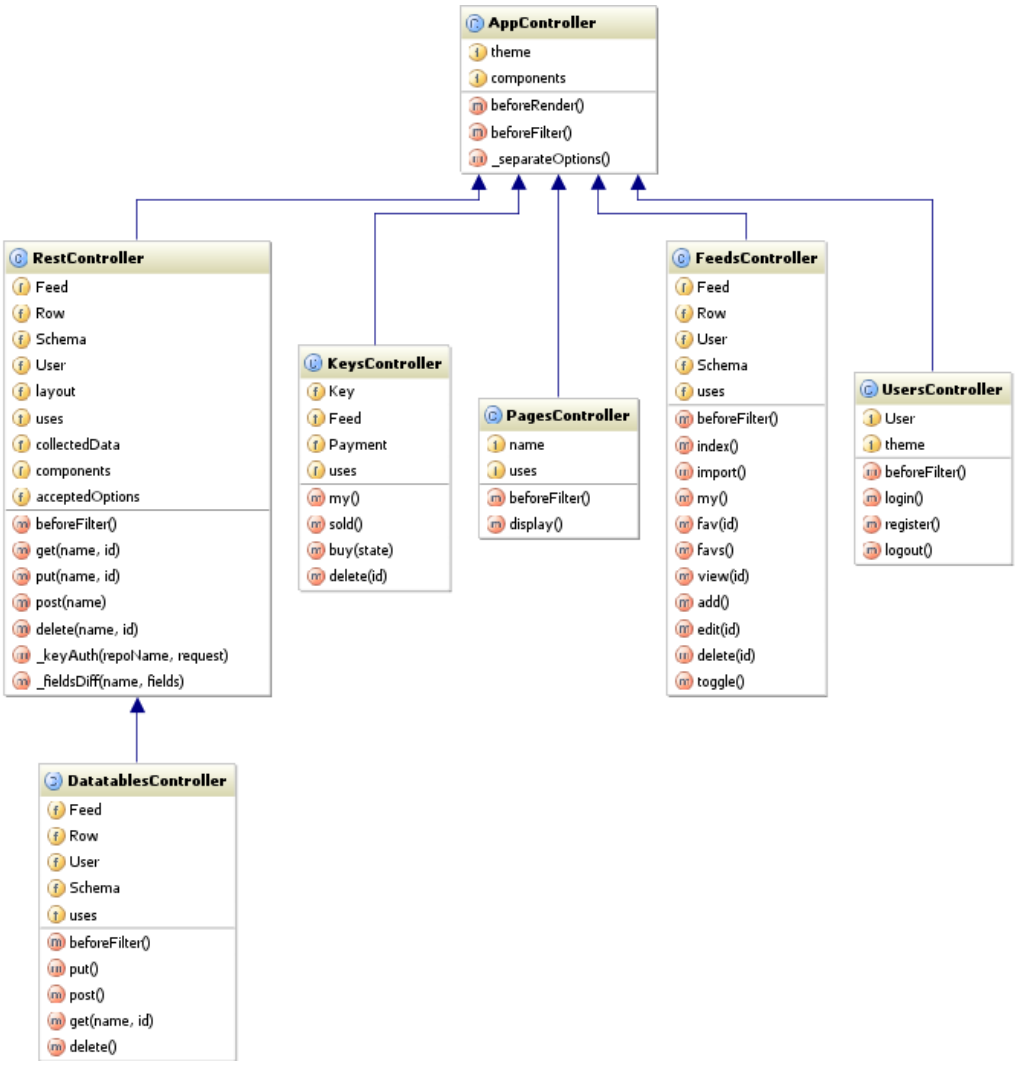
Modele korzystające ze zmiennej $useDbConfig modelują komunikację z bazą MongoDB.



Rys. 5.1. Diagram klas

- Klasa Feed jest reprezentacją repozytorium, zawiera informacje o jego nazwie, opisie oraz użytkowniku (klasa User),
- Klasa Favourite odpowiada za ulubione repozytoria użytkownika,
- Klasy Key oraz Payment są reprezentacją kluczy oraz płatności,
- Klasy Row oraz Schema zawierają dane wprowadzone przez użytkownika do repozytorium oraz ich format.

Rysunek 5.2 prezentuje schemat klas kontrolerów. Widać tutaj zależność kontrolera DatatablesController od RestController, zależność ta wynika z faktu, że biblioteka DataTables użyta po stronie interfejsu użytkownika posiada specyficzne wymagania co do odpowiedzi oraz formatu zwracanych danych.



Rys. 5.2. Diagram klas kontrolerów

5.2. Najważniejsze fragmenty kodu źródłowego

5.2.1. Analiza kodu

Poniższy podpunkt dotyczy zapytania GET (klasa RestController, metoda get). Opis kodu przeprowadzono linia po linii, ignorując nieistotny kod.

Opisano trzy metody klasy RestController:

- beforeFilter,
- get,
- _keyAuth().

Szczegóły uwierzytelnienia użytkownika zawarto w akapicie „Autoryzacja”.

Gdy klasa dyspozytora zadecyduje jaką metodę kontrolera wywołać, następuje wywołanie metody beforeFilter():

```
1. public function beforeFilter() {
2.     parent::beforeFilter();
3.     $this->Auth->allow('get');
```

która w przypadku metody GET,

```
4. if ('get' == $this->request->params['action']) {
```

sprawdza czy użytkownik jest uwierzytelniony, za pomocą nagłówka Authorization: Basic,

```
5. if (!$this->Auth->user()) {
```

oraz parametru (klucza – *key*) i typu repozytorium.

```
6. $this->_keyAuth( // sprawdź klucz
7.     $this->request->params['pass'][0], // $repoName
8.     $this->request // $request
9. );
```

Poniższy fragment przedstawia funkcję _keyAuth. Jeśli nazwa repozytorium jest pusta, zostanie zwrócony kod 403, który oznacza że dostęp jest zabroniony:

```
11.   if ($repoName === null) throw new ForbiddenException();
```

W przypadku, gdyby repozytorium nie istniało w instancjach Memcached, zostanie zwrócony kod 404, który oznacza że repozytorium nie zostało znalezione.

```
12.   if (Memcached::getInstance()->get($repoName) === false)
13.       throw new NotFoundException();
```

Gdyby repozytorium było typu „Premium” oraz użytkownik nie podał klucza, zwrócony zostanie kod 401, który oznacza brak autoryzacji.

```
14.   if (Memcached::getInstance()-
   >get($repoName . '/type') == 'premium'){
15.       $this->collectedData['isPremium'] = true;
16.
17.       if (!isset($request->query['key']))
18.           throw new UnauthorizedException();
19.
20.     $cacheKey = $repoName . '/' . $request->query['key'];
21.     $requestsLeft = Memcached::getInstance()->get($cacheKey);
```

Gdy klucz zostanie podany, ale nie będzie istnieć w żadnej z uruchomionych instancji serwera pamięci podręcznej memcached, zwrócony zostanie kod 403.

```
22.   if ($requestsLeft === false)
23.       throw new ForbiddenException();
24.
```

```
25.   $this->collectedData['key'] = $request->query['key'];
26.   $this->collectedData['keyOk'] = true;
27.   $this->collectedData['requestsLeft'] = $requestsLeft;
```

W przypadku gdy klucz jest poprawny, ale liczba wywołań została zużyta, zostanie zwrócony również kod 403.

```
28.   if ($requestsLeft == 0)
29.       throw new ForbiddenException();
```

Wracając do kodu RestController – w przypadku wysłania zapytania, innego niż GET,

```
30.   } else {
```

Gdy użytkownik nie jest uwierzytelniony,

```
31.   if (!$this->Auth->user()) {
32.       $this->Auth->autoRedirect = false;
```

oraz nie przeprowadził uwierzytelnienia za pomocą nagłówka Authorization: Basic,

```
33.   if (!$this->Auth->login()) {
```

uwierzytelnienie kończy się błędem 401 (brak dostępu).

```
34.   die(header("HTTP/1.0 401 Unauthorized"));
```

W przypadku, gdy autoryzacja przebiegła poprawnie, wywoływana jest metoda get.

```
35.   public function get($name = null, $id = null) {
```

Najpierw następuje sprawdzenie za pomocą obiektu CakeRequest, czy zapytanie jest typu GET (w praktyce oznacza to sprawdzenie poprzez m.in. poprzez nagłówek REQUEST_METHOD). Jeśli nie jest to metoda GET, zwracany jest wyjątek.

```
36.   if (!$this->request->is('get'))
37.       throw new BadRequestException("Only GET is allowed here.");
```

Kolejnym krokiem jest zapytanie bazy Memcached, czy klucz świadczący o widoczności publicznej repozytorium, istnieje. W przeciwnym przypadku zwracany jest wyjątek.

```
38.   if (Memcached::getInstance()->get($name . '/completed') != 1)
39.       throw new ForbiddenException();
```

Jeżeli repozytorium jest typu Premium oraz klucz dostępu jest poprawny, zmniejszany jest w Memcached limit zapytań, jaki pozostał użytkownikowi do zrealizowania w ramach klucza (odpowiednia Komenda co określony czas aktualizuje tę liczbę w MySQL).

```
40.   if ($this->collectedData['isPremium'] &&
41.       $this->collectedData['keyOk']) {
42.           $cacheKey = $name . '/' . $this->request->query['key'];
43.           Memcached::getInstance()->decrement($cacheKey);
44.   }
```

Następnie formowane jest odpowiednie zapytanie.

```
45.   $options = array(
46.       'conditions' => array(
47.           'feed_id' => (int) Memcached::getInstance()->get($name)
48.       ),
49.   );
```

W przypadku, gdy klucz świadczący od ID repozytorium, nie istnieje w instancji (instancjach) Memcached, rzucany jest wyjątek.

```
50.   if ($options['conditions']['feed_id'] === 0)
51.     throw new NotFoundException("Not found, Repository doesn't
   exist.");
```

Następnie oddzielane są parametry obsługiwane przez warstwę REST API (takie jak limit, offset, order, fields) od zbędnych parametrów, które mógłby potencjalnie wysłać użytkownik.

```
52.   $queryOptions = $this->_separateOptions();
```

W przypadku, gdy „użytkownik API" (bądź „administrator API") wprowadził parametr "fields", obliczana jest różnica nazw pól zawartych w tym parametrze oraz pól danego repozytorium, ponieważ baza danych MongoDB wymaga parametru wykluczającego lub włączającego pola (nie można mieszać obydwu typów tak jak w MySQL) – wyjaśnienie w kolejnym akapicie.

Baza danych MySQL umożliwia podanie kolumn, które mają zostać zwrócone w zapytaniu (w typ przypadku, z wszystkich pól zostały wybrane jedynie id oraz name), np.:

```
SELECT id, name FROM table WHERE feed_id = 1
```

W przypadku MongoDB, domyślnie pobierane są wszystkie pola co oznacza, że użytkownik może je jedynie wykluczyć, bądź włączyć − czyli nie jest dozwolone zapytanie typu (ze względu na mieszanie pól wykluczających oraz włączających w parametrze drugim):

```
db.collection.find({feed_id: 1}, {id:1, name:1, created:0,
updated:0})
```

Dlatego też, gdy trzeba ograniczyć wynik zapytania otrzymanego od MongoDB, poprzez zawężenie otrzymanych pól oraz podanie nazw zwróconych pól, należy wykluczyć wszystkie pozostałe pola, oprócz potrzebnych. Odpowiedzialna za to jest _fieldsDiff().

```
53.   if (isset($queryOptions['fields'])) {
54.       $options['fields'] =
55.           $this>_fieldsDiff($name, $queryOptions['fields']);
56.           unset($queryOptions['fields']);
57.       } else {
58.           $options['fields']['_id'] = 0;
59.       }
```

Jeśli podane zostało ID konkretnego rekordu, dodawane jest ono do zapytania ograniczając w ten sposób jego zakres.

```
60.   if (!empty($id)) $options['conditions']['id'] =  $id;
```

Jeśli został podany parametr “order”, tworzona jest z niego tablica (różniąca się od domyślnego format CakePHP, ponieważ wykorzystano klasę MongodbSource) [20].

```
61.   if (isset($queryOptions['order'])) {
62.       $tmp = explode(",", $queryOptions['order']);
63.       $options['order'] = array($tmp[0] => $tmp[1]);
64.       unset($queryOptions['order']);
65.   }
```

Wartości limit oraz offset są rzutowane na typ int.

```
66.   foreach($queryOptions as $key => $value){
67.       $options[$key] = (int) $value;
68.   }
69.
70.   $options['fields']['feed_id'] = 0;
```

Ostatnim krokiem jest wysłanie zapytania do instancji MongoDB oraz odpowiednie formatowanie otrzymanych danych wraz z odpowiedzią.

```
71.   $data = $this->Row->find('all', $options);
72.   $data = Set::extract('/Row/.', $data);
73.   $this->set('data', $data);
```

5.3. Autoryzacja użytkownika

Przy korzystaniu z REST API wymagana jest autoryzacja użytkownika (nie dotyczy to operacji odczytu z darmowego repozytorium). Autoryzacja może odbywać się na dwa sposoby - poprzez podanie parametru „key”, czyli klucza otrzymanego w wyniku zakupienia repozytorium (możliwy jedynie odczyt danych z repozytorium) lub poprzez nagłówek „Authorization: Basic” (możliwa również modyfikacja danych repozytorium).

Użycie parametru klucza „key” za pomocą aplikacji curl zostało pokazane na wydruku 5.1.

```
curl -I -X GET http://apigeum.com/api/statistics-2012/?key=bWFyY2lu
```

Wydruk 5.1: Użycie zakupionego klucza

Wydruk 5.2 przedstawia wykorzystanie nagłówka „Authorization: Basic" umożliwiającego dodatkowo modyfikowanie repozytorium danych (komendy PUT, POST, DELETE)

```
    curl -I \
    -X DELETE \
    -H "Authoriztion: Basic d2hvYTpuaWNl"
http://apigeum.com/api/statistics-2012/
```

Wydruk 5.2: Nagłówek Authorization: Basic[4]

Punkt 5.4 zawiera opis zwracanych kodów, w przypadku poszczególnych wywołań metod.

5.4. REST API i zwracane kody błędów

W przypadku zapytania REST API możemy otrzymać poniższe kody odpowiedzi.

PUT /api/<Your:API>

200 OK
Gdy zasób został zaktualizowany
201 Created
Gdy zasób został utworzony
400 Bad request
Jeśli zapytanie nie jest typu PUT
401 Unauthorized
Jeśli użytkownik nie ma praw dostępu do repozytorium
500 Internal terror
Gdy po stronie serwera wystąpił błąd

POST /api/<Your:API>

201 Created
Gdy zasób został utworzony
400 Bad request
Jeśli zapytanie nie jest typu POST
Jeśli nie wysłano danych JSON
401 Unauthorized
Jeśli użytkownik nie ma praw dostępu do repozytorium
500 Internal terror
Gdy po stronie serwera wystąpił błąd

DELETE /api/<Your:API>

204 No Content
Kiedy zasób został usunięty

[4] Wartość nagłówka Authorization: Basic to BASE64(email:hasło)

400 Bad request
Jeśli zapytanie nie jest typu DELETE
401 Unauthorized
Jeśli użytkownik nie ma praw dostępu do repozytorium
403 Forbidden
Gdy nie podano nazwy repozytorium w URL
Gdy nie podano informacji na temat klucza REST API
Gdy klucz REST API stracił ważność
404 Not found
Gdy zasób nie został znaleziony
500 Internal terror
Gdy po stronie serwera wystąpił błąd

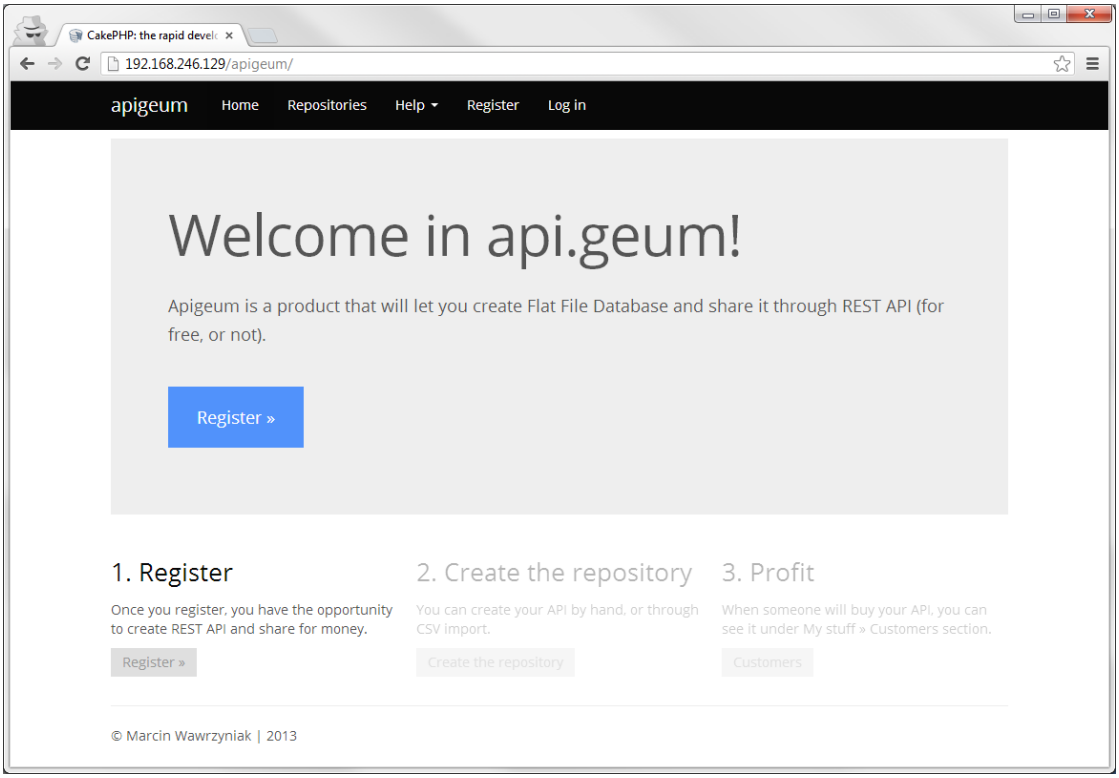
GET /api/<Your:API><:id>
200 OK
Gdy zasób został znaleziony oraz zwrócony
400 Bad request
Gdy zapytanie nie jest typu GET
401 Unauthorized
Gdy użytkownik nie ma praw dostępu do repozytorium
403 Forbidden
Gdy nie ma nazwy repozytorium w URL
Gdy nie podano informacji na temat klucza API
Gdy klucz REST API stracił ważność
404 Not found
Gdy zasób nie został znaleziony
500 Internal terror
Gdy po stronie serwera wystąpił błąd

Tabela 5.1 podsumowuje kody odpowiedzi zwracane przez warstwę REST API.

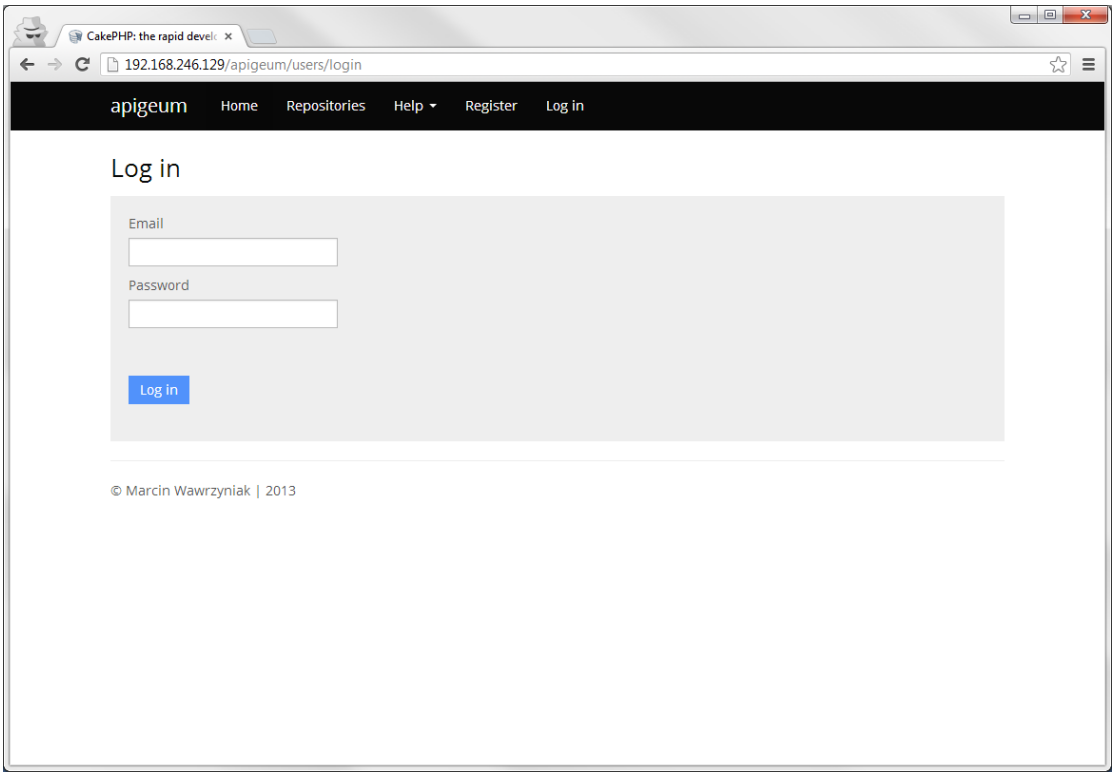
Tabela 5.1: Kody odpowiedzi

Kod odpowiedzi	GET	PUT	POST	DELETE
200 OK.		X	X	
201 Created		X	X	
204 No Content				X
400 Bad Request		X	X	X
401 Unauthorized	X	X	X	X
403 Forbidden	X	X	X	X
404 Not Found	X	X	X	X
500 Internal terror	X	X	X	X

Na rysunkach 5.3 – 5.9 zademonstrowano wygląd aplikacji - stronę główną, logowanie, lista repozytoriów, dodawanie repozytorium, oraz podgląd kluczy.

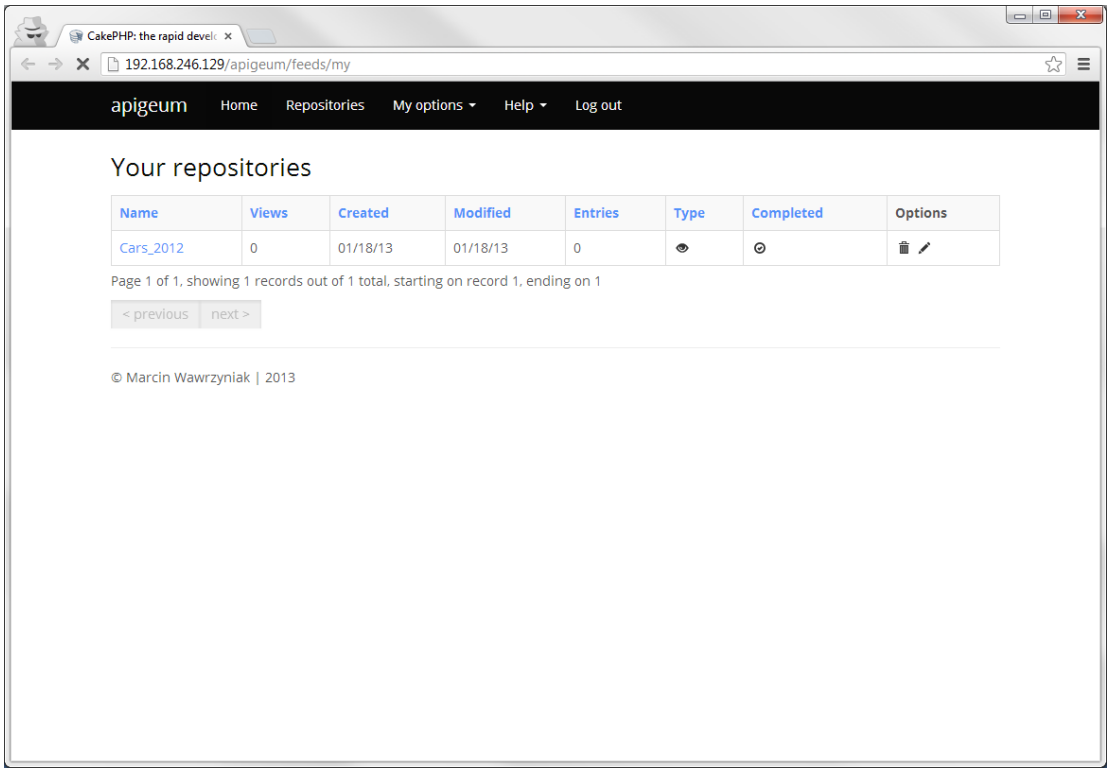


Rys. 5.3. Strona główna aplikacji WWW



Rys. 5.4. Logowanie w aplikacji WWW

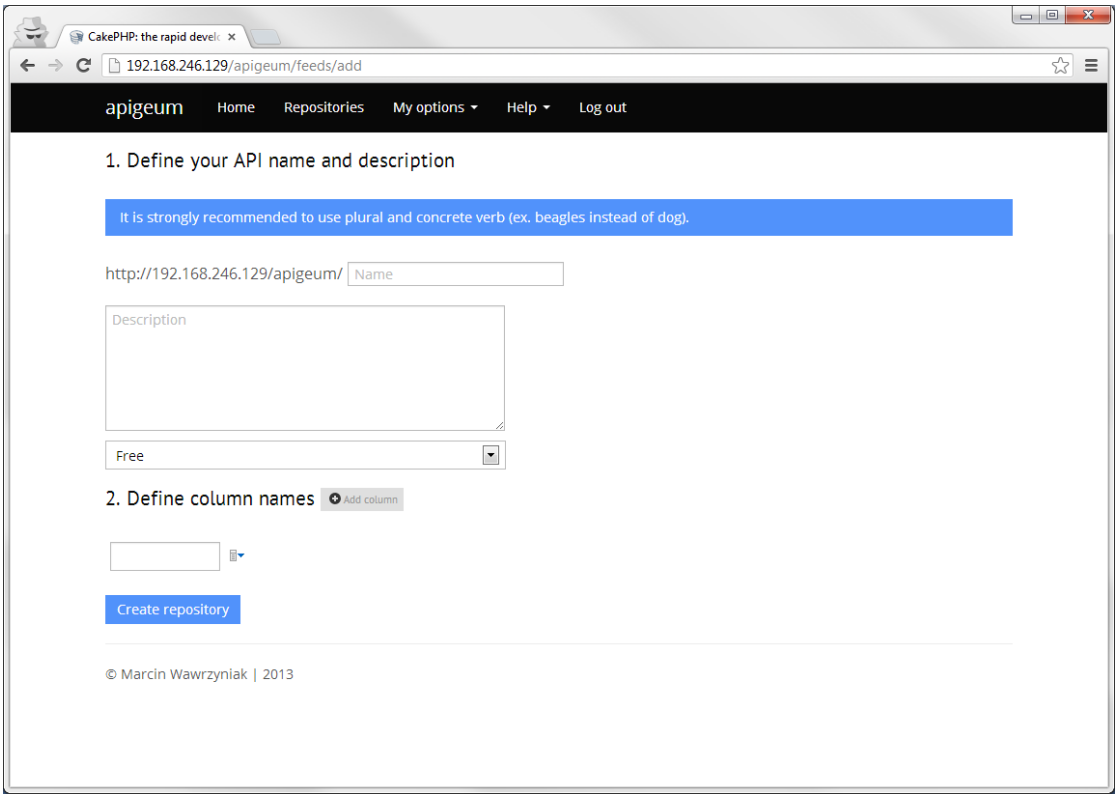
Rys. 5.4 prezentuje ekran logowania. Do zalogowania wymagany jest email (służy jako nazwa użytkownika) oraz wcześniej podane hasło (przechowywane w postaci skrótu funkcji mieszającej). W przypadku rejestracji wymagany jest również jedynie email oraz hasło.



Rys. 5.5. Lista repozytoriów użytkownika

Widok listy repozytoriów użytkownika, składa się z ośmiu kolumn:

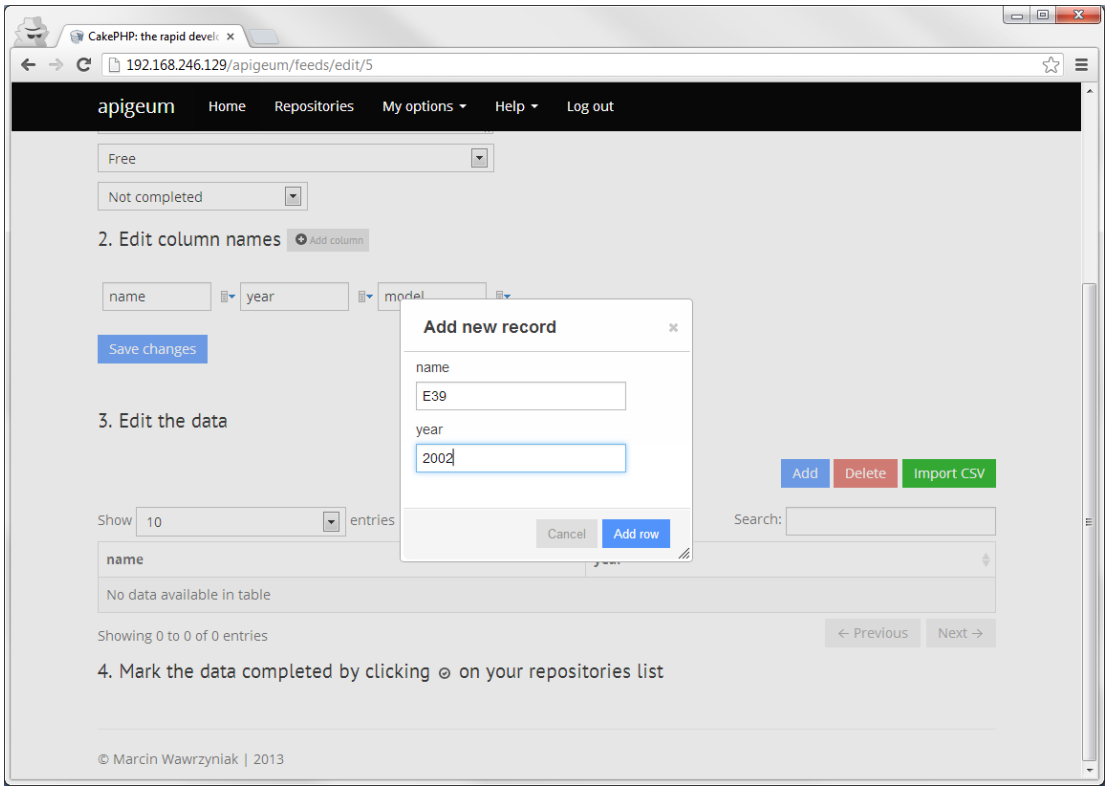
1. Name – zawiera nazwę repozytorium,
2. Views – określa ile razy „strona wizytówka" repozytorium została wyświetlona,
3. Created – określa datę utworzenia repozytorium,
4. Modified – określa datę ostatniej modyfikacji repozytorium,
5. Entries – określa liczbę wierszy z których składa się repozytorium,
6. Type – określa typ repozytorium (darmowy bądź płatny),
7. Completed – określa stan repozytorium – czy proces edycji repozytorium (m.in. dodawanie wierszy) przez użytkownika zakończyła się,
8. Options – umożliwia usunięcie bądź edycję repozytorium.



Rys. 5.6. Tworzenie nowego repozytorium

Podczas tworzenia repozytorium, należy podać następujące informacje:

1. Name – nazwa repozytorium, powinna być w formacie kompatybilnym ze standardem URL [35] - jeśli nie będzie kompatybilna, zostanie przekonwertowana (pole wymagane),
2. Description – opis repozytorium widoczny m.in. na „stronie wizytówce” (pole opcjonalne),
3. Type – typ repozytorium (może być „Free” lub „Premium”), określa czy repozytorium ma być darmowe, czy płatne,
4. Column names – są to nazwy kolumn z których składać się będą wiersze repozytorium, pole wymagane (należy podać przynajmniej jedną kolumnę).

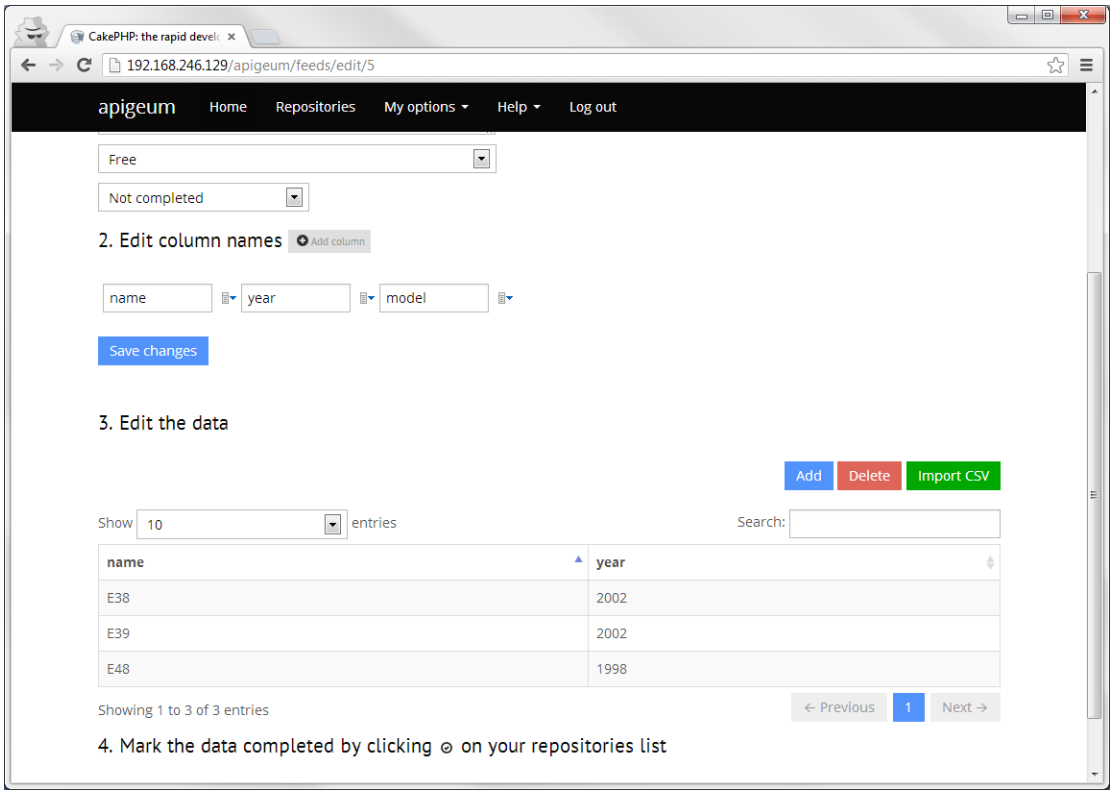


Rys. 5.7. Edycja repozytorium (dodawanie wiersza)

Widok edycji, różni się od widoku dodawania repozytorium niemożliwością edycji nazwy repozytorium, oraz dodatkową możliwością dodania wierszy repozytorium.

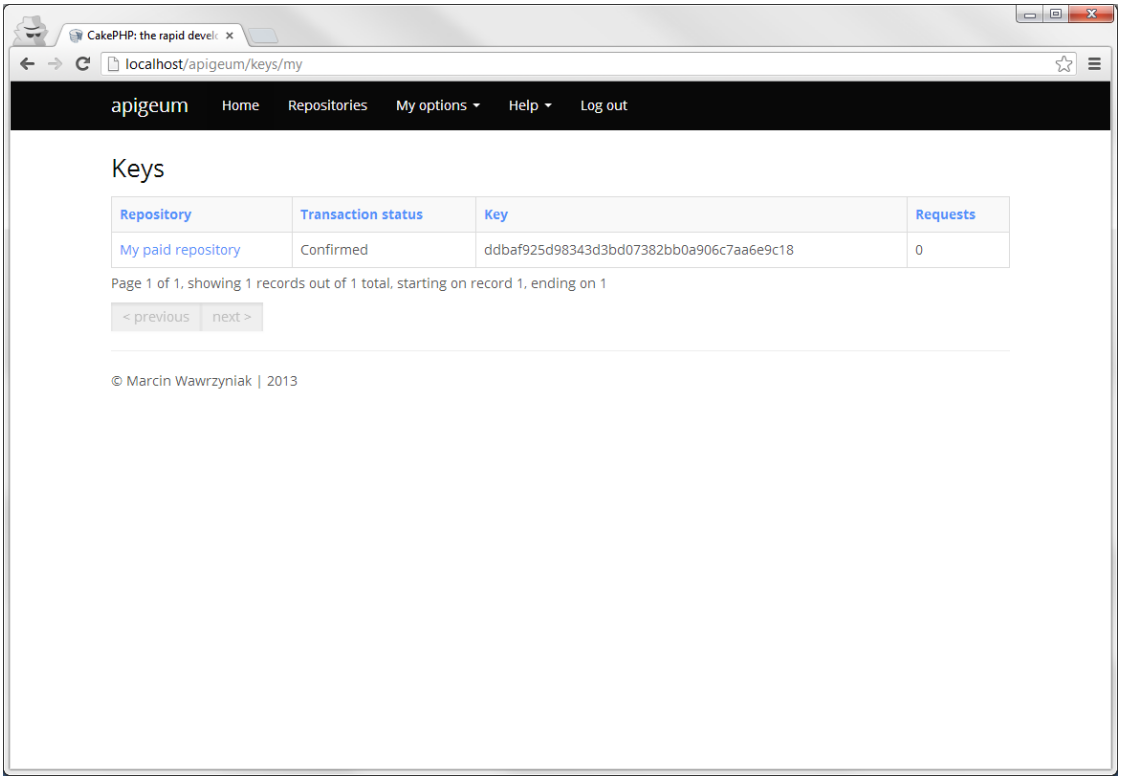
Proces dodawania wierszy odbywa się asynchronicznie (techniką AJAX) i następuje po aktywowaniu przycisku „Add”.

Możliwy jest również import plików CSV, proces ten rozpoczyna się po aktywowaniu zielonego przycisku „Import CSV”. W przypadku gdy nazwy kolumn pliku CSV nie będą pasowały wcześniej utworzonym nazwom w repozytorium, zostaną one pominięte.



Rys. 5.8. Lista dodanych wierszy

Na rysunku 5.8 przedstawiono dodane przez użytkownika przykładowe dane.



Rys. 5.9. Lista kluczy użytkownika

Na rysunku 5.9 pokazano klucze zakupione przez użytkownika.

Tabela prezentująca zakupione klucze, umożliwiające dostęp do płatnych repozytoriów, składa się z czterech kolumn:

1. Repository – nazwa repozytorium, do którego uprawnia klucz,
2. Transaction status – status płatności,
3. Key – wartość parametru klucza,
4. Requests – liczba pozostałych zapytań do wykorzystania w ramach klucza.

5.5. Instalacja

Omawiany system opiera się na zestawie rozwiązań LAMP (akronim od Linux, Apache, MySQL, PHP).

Kolejne podpunkty opisują instalację omawianego oprogramowania, przy założeniu, że zainstalowany został zestaw LAMP.

1. Zainstalować zestaw LAMP [38] [38] (zalecane jest użycie modułu Apache: mod_rewrite).
2. Edytować plik php.ini, dodać:

```
extension=php_mysql.dll
extension=pdo_mysql.dll
extension=php_mongo-[version].dll
extension=php_memcache.dll
```

3. Zainstalować MongoDB (skopiować plik .dll do katalogu php/ext).
4. Zainstalować memcached (skopiować plik .dll do katalogu php/ext).
5. Uruchomić domyślną instancję memcached.
6. Uruchomić domyślną instancję MongoDB.
7. Uruchomić komendę

   mysql << %appdir%\app\Config\Schema\db.sql

6. TESTY

Rozdział ten opisuje metody testowania wykonanego oprogramowania, zaczynając od testów akceptacyjnych a kończąc na testach obciążenia.

6.1. Testy akceptacyjne

Celem testów akceptacyjnych jest zbadanie zachowania kluczowych elementów systemu. Do testowania aplikacji wykorzystano Framework Codeception.

Testy objęły następujące przypadki użycia:

- Administrator API:
  - Pobranie rekordów bazy danych ,
  - Dodanie rekordu (nagłówek Authorization: Basic),
  - Aktualizacja rekordów bazy danych(nagłówek Authorization: Basic),
  - Pobranie konkretnego rekordu,
  - Usunięcie rekordu (nagłówek Authorization: Basic),
- Użytkownik API:
  - Brak dostępu: usunięcie rekordu,
  - Pobranie listy darmowych wierszy repozytorium,
  - Pobranie pojedynczego darmowego wiersza repozytorium,
  - Brak dostępu: pobranie nie zakończonego repozytorium,
  - Brak dostępu: pobranie wierszy płatnego repozytorium, bez klucza,
  - Brak dostępu: pobranie wierszy płatnego repozytorium, błędny klucz,
  - Pobranie wierszy płatnego repozytorium, klucz poprawny,
- Użytkownik WWW:
  - Logowanie,
  - Rejestracja.

Wydruk 6.1 przedstawia jeden z testów - pobranie rekordów poprzez użytkownika API.

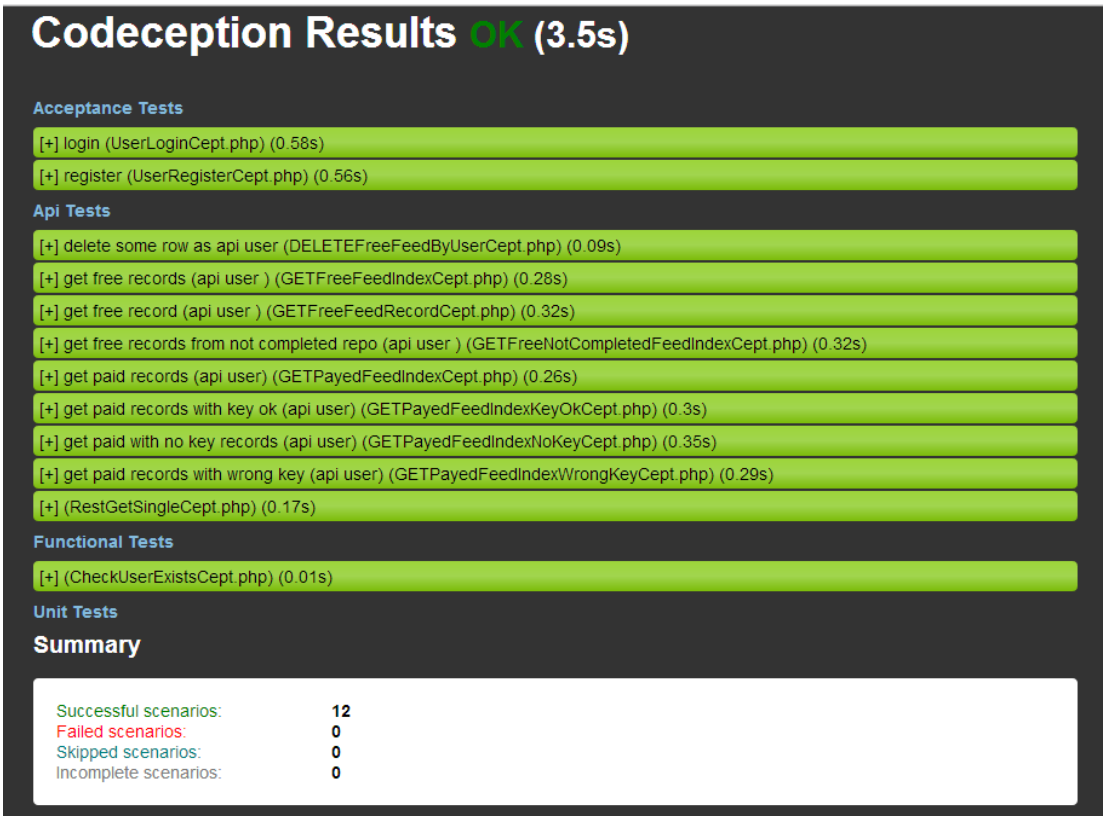
```
<?php

use \Codeception\Module\Db;

$I = new ApiGuy($scenario);

$I->wantTo('GET rows list');

$I->sendGET('/api/my-repository');

$I->seeResponseCodeIs(200);

$I->seeResponseIsJson();

$I->seeResponseContainsJson(array(array('field' => 'value')));
```

Wydruk 6.1. Przykładowy test frameworka Codeception

Wydruk 6.2 zawiera komendę, która wykonuje wszystkie napisane testy, czego rezultatem jest plik raportu pokazany na rys. 6.1.

```
php codecept.phar run
```

Wydruk 6.2. Uruchomienie testów z linii komend



Rys. 6.1. Raport w formacie HTML Codeception

6.2. Testy obciążeniowe

Do przeprowadzenia testów obciążeniowych wykorzystano program ApacheBench (ab.exe) [24].

Specyfikacja maszyny testowej prezentuje się następująco:

- Intel Pentium Dual Core T4300 2.10GHz,
- 3 GB pamięci RAM ,
- Dysk SSD Crucial M4.

Na fizycznej maszynie testowej uruchomiono maszynę wirtualną zawierającą skonfigurowaną aplikację gotową do użytku, a następnie przeprowadzono testy obciążeniowe.

6.2.1. Test pierwszy

W celu przeprowadzenia pierwszego testu użyto polecenia:

```
ab.exe -n 10000 -c 50 http://192.168.246.129/apigeum/api/repository
```

Parametr –n oznacza całkowitą ilość zapytań jakie mają zostać wysłane, natomiast parametr –c określa ilość jednoczesnych zapytań.

Adres podany jako trzeci parametr, to adres darmowego repozytorium zawierającego zaledwie 5 wierszy danych.

Wydruk 6.2 przedstawia wyjście programu ApacheBench po przeprowadzeniu testu:

```
Concurrency Level:      50
Time taken for tests:   576.455 seconds
Complete requests:      10000
Failed requests:        0
Write errors:           0
Keep-Alive requests:    9933
Total transferred:      5566788 bytes
HTML transferred:       1870000 bytes
Requests per second:    17.35 [#/sec] (mean)
Time per request:       2882.276 [ms] (mean)
Time per request:     57.646 [ms] (across all concurrent requests)
Transfer rate:          9.43 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   9.4      0     934
Processing:  1083 2871 806.0   2928   12284
Waiting:     1083 2871 806.0   2928   12284
Total:       1084 2872 805.9   2928   12284

Percentage of the requests served within a certain time (ms)
  50%   2928
  66%   3230
  75%   3401
  80%   3517
  90%   3805
  95%   4018
  98%   4322
  99%   4613
 100%  12284 (longest request)
```

Wydruk 6.3. Wyjście programu ApacheBench

Z wydruku tego wynikają następujące fakty:

- Testy zajęły łącznie 576 sekund,
- Wysłano 10.000 zapytań,
- Osiągnięto wydajność 17 zapytań na sekundę,
- Jedno zapytanie zajmuje średnio 2,8s.

6.2.2. Test drugi

W celu przeprowadzenia drugiego testu użyto polecenia:

```
ab.exe -t 300 -c 10 http://192.168.246.129/apigeum/api/repository
```

```
Benchmarking 192.168.246.129 (be patient)
Finished 1762 requests


Server Software:        Apache/2.2.16
Server Hostname:        192.168.246.129
Server Port:            80

Document Path:          /apigeum/api/bmw-cars
Document Length:        63 bytes

Concurrency Level:      100
Time taken for tests:   300.049 seconds
Complete requests:      1762
Failed requests:        0
Write errors:           0
Non-2xx responses:      1762
Keep-Alive requests:    1762
Total transferred:      773618 bytes
HTML transferred:       111006 bytes
Requests per second:    5.87 [#/sec] (mean)
Time per request:       17028.897 [ms] (mean)
Time per request:       170.289 [ms] (mean, across all concurrent
requests)
Transfer rate:          2.52 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0  11.6      0     463
Processing:   228 15738 15133.6   9617   89657
Waiting:      228 15738 15133.6   9617   89656
Total:        228 15738 15133.3   9617   89657
```

Wydruk 6.4. Wyjście programu ApacheBench

Z drugiego testu wynika ilość zapytań, jaką serwer jest w stanie obsłużyć w czasie 5 minut (przy maksymalnej liczbie jednoczesnych połączeń nie większej niż 100) – jest to wynik równy 1762 zapytaniom.

Należy mieć na uwadze iż wydajność w rzeczywistych warunkach będzie się różnić.

7. WNIOSKI

W perspektywie długoterminowego rozwoju aplikacji poniższe funkcjonalności mogłyby zostać zaimplementowane:

- Obsługa rzeczywistych płatności internetowych (np. DotPay, Payu).
- Możliwość importu innych formatów danych niż CSV (np. XLS).
- Rozbudowa mechanizmu powiadomień poprzez email .
- Możliwość eksportu danych.
- SSL.
- Implementacja standardu OAuth 2.0.
- Moduł odpowiedzialny za aktualność danych memcached w przypadku, gdy serwer pamięci podręcznej zostanie zrestartowany.
- Panel podglądu statystyk używanych kluczy oraz zarządzania repozytoriami.

Celem niniejszej pracy było zbudowanie działającej aplikacji WWW służącej tworzeniu oraz udostępnianiu repozytoriów. Udało się to zrealizować.

Napotkane problemy udało się wyeliminować za pomocą zmian projektowych, takich jak częściowe wykorzystanie serwera baz danych MongoDB zamiast bazy MySQL.

Kluczowe elementy systemu zostały poddane testom.

Aplikacja może zostać użyta jako główne (np. kolekcja muzyczna) lub pomocnicze (np. lista producentów samochodów na portalu motoryzacyjnym) źródło danych dowolnej treści dla innych programów internetowych poprzez interfejs REST.

Podsumowując, praca ta wpłynęła na wiedzę autora w zakresie danych usług sieciowych oraz wykorzystanych narzędzi. Autor doszedł do wniosku, iż priorytetem powinno być wybranie odpowiedniego narzędzia do wykonywanego zadania oraz systematyczna praca w środowisku, które nie rozprasza.

8. WYKAZ LITERATURY

[1] Abeysinghe S., RESTful PHP Web Services, Packt, 2008

[2] Balbo B., Fuecks H., Shafik D., Turmelle L., Weler M., The PHP Anthology, Collingwood: Sitepoint, 2007

[3] Beck K., Extreme Programming Explained: Embrace Change, Addison-Wesley, 2004

[4] Chodorow K., Dirolf M., MongoDB: The Definitive Guide, Sebastopol: O'Reilly, 2010

[5] Finsel J., Using memcached, The Pragmatic Bookshelf, 2008

[6] Kunze M., „German software magazine," 1998

[7] McArthur K., Pro PHP, Apress, 2008

[8] Shiflett C., Essential PHP Security, O'Reilly, 2005

[9] Ulman L., Visual QuickPro Guide PHP 6 and MySQL 5 For Dynamic Web Sites, Peachpit Press, 2007

[10] Weisfeld M., The Object-Oriented Thought Process, Sams Publishing, 2004, p. 239

[11] http://apify.heroku.com/resources

[12] http://book.cakephp.org

[13] http://book.cakephp.org/1.1/view/307/Introduction-to-CakePHP

[14] http://book.cakephp.org/2.0/en/getting-started/cakephp-conventions.html

[15] http://bsonspec.org

[16] http://codeception.com/

[17] http://datatables.net/

[18] http://ecma-international.org/publications/standards/Ecma-262.htm

[19] http://geo-autocomplete.com/

[20] http://github.com/ichikaway/

[21] http://github.com/php-fig/fig-standards/blob/master/accepted/PSR-0.md

[22] http://github.com/php-fig/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md

[23] http://github.com/php-fig/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md

[24] http://httpd.apache.org/docs/2.2/programs/ab.html

[25] http://it-consulting.pl

[26] http://jquery.com/

[27] http://mongodb.org/

[28] http://msdn.microsoft.com/en-us/library/ff647543.aspx

[29] http://nosql-database.org/

[30] http://owasp.org/index.php/SQL_Injection

[31] http://php.net, PHP Manual, php.net

[32] http://php.net/manual/en/function.uniqid.php

[33] http://php.net/manual/en/language.oop5.autoload.php

[34] http://pl.wikibooks.org/wiki/PHP/Czym_jest_framework%3F

[35] http://tools.ietf.org/html/rfc1738

[36] http://tools.ietf.org/html/rfc2616

[37] http://tools.ietf.org/html/rfc3174

[38] http://tools.ietf.org/html/rfc4627

[39] http://trends.builtwith.com/docinfo/Twitter-Bootstrap

[40] http://w3.org/Protocols/rfc2616/rfc2616.txt

[41] http://w3.org/TR/soap12-part1/#soapenvelope

[42] http://w3techs.com/technologies/overview/javascriptlibrary/all

[43] http://www.google.com/trends/explore#q=REST&cmpt=q

[44] http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[45] http://www.wildcrest.com/Potel/Portfolio/mvp.pdf