PastExplorer

Dokumentacja projektowa

# 

Autorzy:

Jakub Jaśkowiec

Bartłomiej Hyży

Michał Pieróg

Informatyka Stosowana

I rok II stopnia

WEAIiE, AGH

Kraków, 21.06.2012

# Opis systemu

Aplikacja internetowa utworzona w ramach tego projektu pozwala za pomocą zdjęć umieszczanych przez  
użytkowników obserwować jak różnego rodzaju “rzeczy” (np. ludzie, budynki, krajobrazy) zmieniały się wraz z upływem czasu. Każdy użytkownik może zakładać albumy będące kolekcjami zdjęć zgodnych tematycznie robionych w różnych odstępach czasu, np. codziennie, cotygodniowo, bądź corocznie.

Aplikacja jest mocno nakierowana “społecznościowo”, tj. zawiera elementy pozwalające na tworzenie wokół serwisu społeczności użytkowników aktywnie uczestniczących w jego rozwoju, np. ocenianie i komentowanie albumów.

Przykłady zastosowań:

* codzienne zdjęcia rozwoju naszego dziecka
* cotygodniowe zdjęcia stanu budowy pewnego obiektu, np. stadionu piłkarskiego
* coroczne zdjęcia rynku w naszym mieście

# Diagram przypadków użycia serwisu



Rysunek : diagram przypadków użycia

# Wykorzystane technologie

# Architektura systemu

Oprócz właściwego serwisu dostępnego dla użytkowników z poziomu przeglądarki internetowej, dodatkowo istnieje możliwość korzystania z części funkcjonalności portalu za pośrednictwem aplikacji mobilnej. W związku z tym konieczne okazało się zaprojektowanie architektury systemu w sposób pozwalający na korzystanie z niego w sposób uniezależniony od typu dostępnego urządzenia wyświetlającego, np. monitora lub ekranu telefonu komórkowego.

Portal zbudowany został w architekturze trójwarstwowej i składa się z warstwy danych, warsty logiki biznesowej oraz warstwy prezentacji.

## Warstwa danych

System korzysta z pojedynczej bazy danych, dostęp do której odbywa się za pomocą klas realizujących wzorzec DAO (Data Access Object). Dzięki zastosowaniu tego wzorca pozostałe części systemu są w pełni odseparowane od konkretnego motora bazodanowego, co teoretycznie pozwala na jego podmianę na dowolny inny przy zajściu takiej potrzeby, pod warunkiem, że interfejs dostępowy do encji bazodanowych pozostanie niezmieniony.

## Warstwa logiki biznesowej

Warstwa ta realizuje rzeczywiste funkcjonalności serwisu i stanowi jego rdzeń. Za operację na obiektach wchodzących w skład zbioru modeli system odpowiadają kontrolery, podzielone na dwie grupy:

* kontrolery aplikacji internetowej – korzysta z nich aplikacja internetowa dostępna poprzez przeglądarkę internetową
* kontrolery webserwisowe – korzysta z nich aplikacja mobilna

Obie grupy kontrolerów współdzielą pewne wspólne funkcjonalności, takie jak operacje na modelach danych czy logowanie.

Proponowana architektura systemu byłaby bardziej spójna i efektywna, gdyby zamiast stworzenia osobnych grup kontrolerów mających za zadanie obsługę żądań różnych typów aplikacji, stworzyć dodatkową warstwę udostępniającą zestaw wszystkich funkcjonalności realizowanych przez portal w formie webserwisu. Obie aplikacje (internetowa i mobilna) komunikowały by się wtedy bezpośrednio z tym webserwisem, formatując i przetwarzając otrzymane w wyniki w sposób dla nich specyficzny. Wyeliminowałoby to wadę, jaką posiada aktualna architektura, tj. niewielką duplikację logiki pomiędzy poszczególnymi grupami kontrolerów. Ponadto taka modyfikacja pozwoliłaby na efektywniejsze testy głównych funkcjonalności portalu, które aktualnie muszą być wykonywane osobno dla obu grup.

## Warstwa prezentacji

Warstwa ta odpowiada za prezentację wyników przeprowadzanych operacji w sposób zależny od docelowego urządzenia wykonującego zapytanie:

* aplikacja internetowa – rezultaty działań użytkownika zwracane są w postaci widoków HTML, odpowiednich do bezpośredniego wyświetlenia przez jego przeglądarkę internetową
* aplikacja mobilna – rezultaty działań użytkownika zwracane są w postaci encji JSON, które są następnie interpretowane i formatowane do wyświetlenia w odpowiedni sposób

## *architektura.png*

Rysunek : architektura serwisu

# Encje

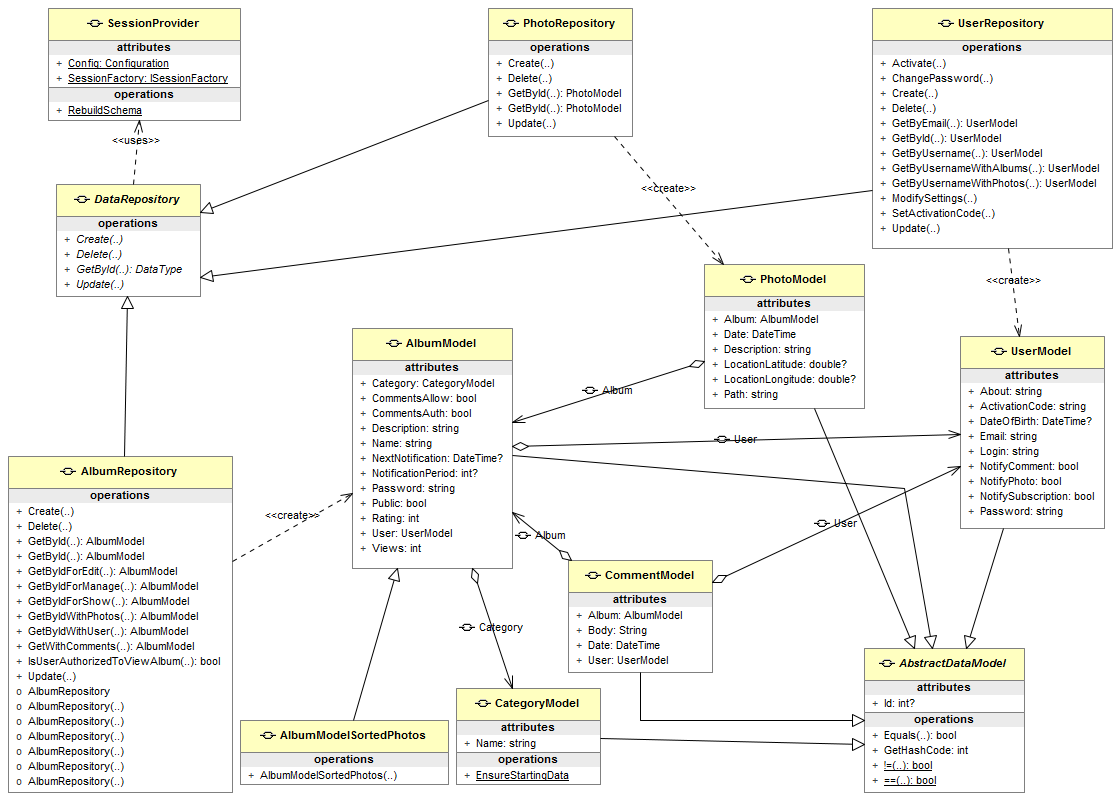
# Klasy

# Wykonanie akcji

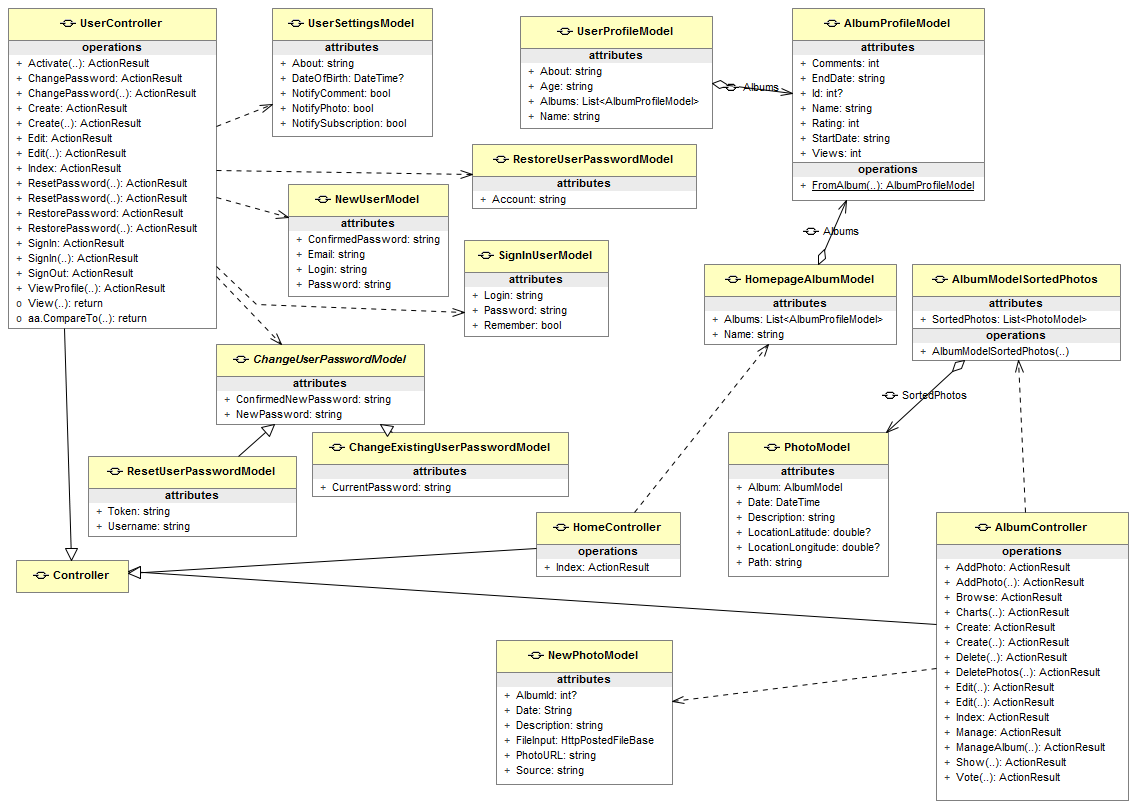
# Stany aplikacji internetowej

# Webservice – API w stylu REST

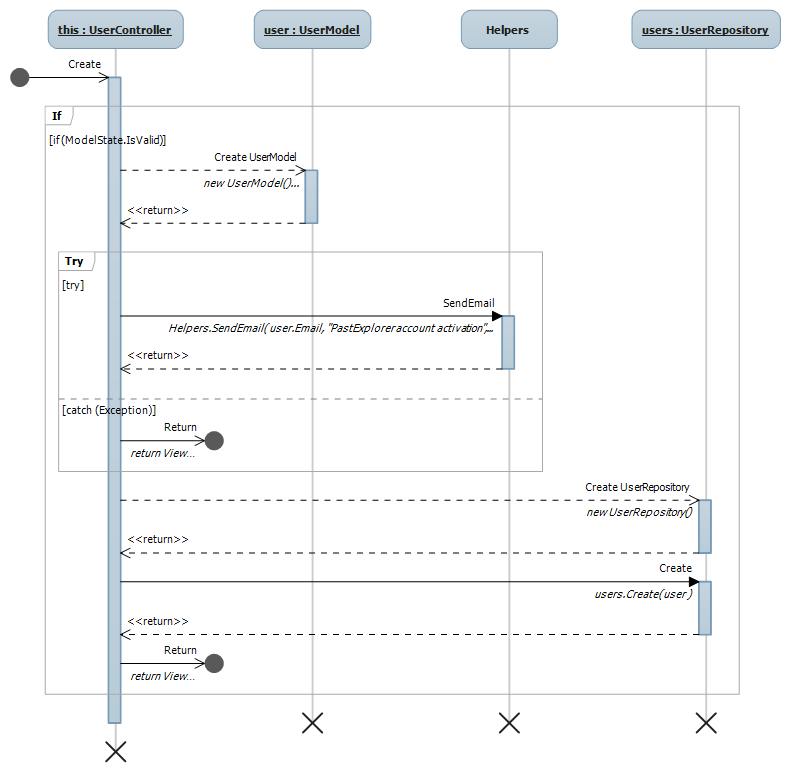
## Diagram klas: Warstwa dostępu do bazy danych



## Diagram klas: Warstwa logiki



## Diagram sekwencji: Tworzenie konta



## Diagram sekwencji: Tworzenie albumu

## createAlbum-sequence.png

## Webservice (REST API)

W związku z tym, że implementacja właściwego serwisu internetowego obsługiwanego za pomocą przeglądarki internetowej zmierza ku końcowi, postanowiliśmy w ramach projektu wykonać aplikację mobilną, która pozwoli wykonywać część czynności dostępnych z poziomu serwisu za pośrednictwem smartfona z systemem Android.

Aby było to możliwe, konieczna była implementacja interfejsu dostępowego do funkcjonalności serwisu. Dotychczasowa implementacja dobrze nadawała się dla użytkownika łączącego się z portalem za pomocą przeglądarki internetowej. Rezultaty jego akcji zwracane były w postaci kodu HTML renderowanego przez przeglądarkę w sposób atrakcyjny wizualnie. Taka reprezentacja jest jednak mało efektywna i ergonomiczna dla aplikacji mobilnej.

Zdecydowaliśmy się więc na implementację interfejsu dostępowego dla aplikacji mobilnej w oparciu o model REST (ang. *Representational State Transfer*). Aktualnie zyskuje on coraz większą popularność i zaczyna wypierać bardziej standardowe rozwiązania oparte o SOAP/XML/WSDL, które charakteryzują się stosunkowo dużą złożonością konfiguracji i formalizmem.

**Funkcjonalność aplikacji mobilnej:**

* autentykacja użytkownika poprzez mechanizm HTTP Basic Authentication (HTTPS?)
* wyświetlanie listy albumów użytkownika
* wyświetlanie zdjęć w albumie użytkownika
* robienie zdjęcia z poprzednimi zdjęciami jako przezroczystymi makietami, wysyłanie do serwisu
* dołączanie danych geolokalizacyjnych do zdjęć

**Zasoby obsługiwane przez REST API:**

* użytkownicy (/api/users)
* albumy (/api/albums)
* zdjęcia (/api/photos)

**Pełny listing aktualnie zaimplementowanego API wraz z przykładowymi wywołaniami:**

* zwracanie informacji o użytkowniku

**GET /api/users/JanekKowalski**

{

"ok": true,

"data": {

"id": 3,

"username": "JanekKowalski",

"date\_of\_birth": {

"day": 1,

"month": 3,

"year": 1989

},

"about": "Jestem z Krakowa. Lubię jeździć na rowerze.",

"albums": [

"http://localhost:3518/api/albums/5",

"http://localhost:3518/api/albums/18"  
 ]

}

}

* zwracanie informacji o albumie

**GET /api/albums/5**

{

"ok": true,

"data": {

"id": 5,

"name": "Moja twarz",

"description": "Jak zmieniałem się w czasie",

"category": "People",

"owner": "JanekKowalski",

"is\_public": true,

"rating": 10,

"views": 1234,

"photos": [

"http://localhost:3518/api/photos/1",

"http://localhost:3518/api/photos/2",

"http://localhost:3518/api/photos/3",

"http://localhost:3518/api/photos/4",

"http://localhost:3518/api/photos/5",

"http://localhost:3518/api/photos/6",

"http://localhost:3518/api/photos/7"

],

"comments": []

}

}

* zwracanie informacji o zdjęciu

**GET /api/photos/2**

{

"ok": true,

"data": {

"id": 2,

"album": "http://localhost:3518/api/albums/5",

"date": {

"day": 30,

"month": 4,

"year": 2011

},

"description": "Oto ja",

"image": "http://localhost:3518/Static/photos/photo\_2012051022450267.jpg",

"thumbnail": "http://localhost:3518/Static/photos/photo\_2012051022450267\_mini.jpg",

"latitude": 25.21356,

"longitude": 34.12357

}

}

* wysyłanie zdjęcia

**POST /api/photos**

**(... HTTP Body: zdjęcie, metadane ...)**

## Autentykacja użytkownika (REST API)

Istnieje wiele rozwiązań problemu weryfikacji, czy użytkownik jest tym, za kogo się podaje (autentykacja) i czy posiada dostęp do określonego zasobu (autoryzacja). Przed przystąpieniem do implementacji API REST analizie poddane zostało pare rozwiązań, ostatecznie jednak zdecydowaliśmy się na wbudowany w HTTP mechanizm Basic Authentication z następujących powodów:

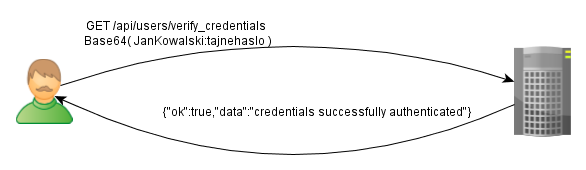
* prostota, autentykacja polega na dodaniu jednego nagłówka do zapytania HTTP z zakodowanymi w Base64 danymi użytkownika (login + hasło)
* brak konieczności przetrzymywania po stronie serwera żadnego stanu zalogowania użytkownika, jak w przypadku sesji, co naruszało by postulamy modelu REST
* jest to mechanizm wbudowany we wszystkie przeglądarki – duża uniwersalność

Metoda ta jednak posiada pewne wady:

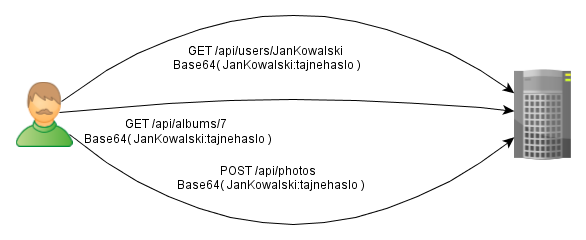
* dane użytkownika (w tym hasło) przesyłane są w zapytaniu HTTP tekstem jawnym, stąd konieczność komunikacji przez HTTPS (szyfrowanie SSL/TLS)
* dane użytkownika przesyłane są wraz z każdym zapytaniem, co może stwarzać niepotrzebne zagrożenie; bezpieczniejszym rozwiązaniem może być modyfikacja mechanizmu, polegająca na przesyłaniu skrótu hasła obliczonego funkcją hashująca, oraz odpowiednia interpretacja tego skrótu po stronie serwera (tracimy jednak w ten sposób możliwość współpracy z przeglądarkami internetowymi, które nie potrafią obsłużyć takiego mechanizmu; w naszym przypadku nie jest to problemem, z uwagi na to, że API przeznaczone jest głównie dla aplikacji mobilnej, którą sami zaimplementujemy)

**Autentykacja użytkownika odbywa się dwustopniowo:**

1. Weryfikacja danych logowania wprowadzonych przez użytkownika



1. Autentykacja przy każdym zapytaniu



W obu krokach uwierzytelnianie przeprowadzane jest w ten sam sposób. W pierwszym kroku następuje jedynie weryfikacja danych logowania wprowadzonych przez użytkownika, dzięki czemu aplikacja mobilna może od razu wykryć, że wprowadzone zostały nieprawidłowe dane, poinformować o tym i zablokować wykonywanie zapytań do momentu podania poprawnych danych.

Dodatkowo autentykacja odbywa przy każdym zapytaniu, gdyż nie można zakładać, że wstępna weryfikacja danych logowania zostanie w ogóle przeprowadzona. Powoduje to pewien narzut wydajnościowy, jednak jest on naszym zdaniem zdecydowanie akceptowalny, biorąc pod uwagę prostotę takiego mechanizmu oraz jego przenośność.