PastExplorer

Dokumentacja projektowa

Autorzy:

Jakub Jaśkowiec
Bartłomiej Hyży
Michał Pieróg

Informatyka Stosowana
I rok II stopnia
WEAIiE, AGH

Kraków, 21.06.2012

Opis systemu

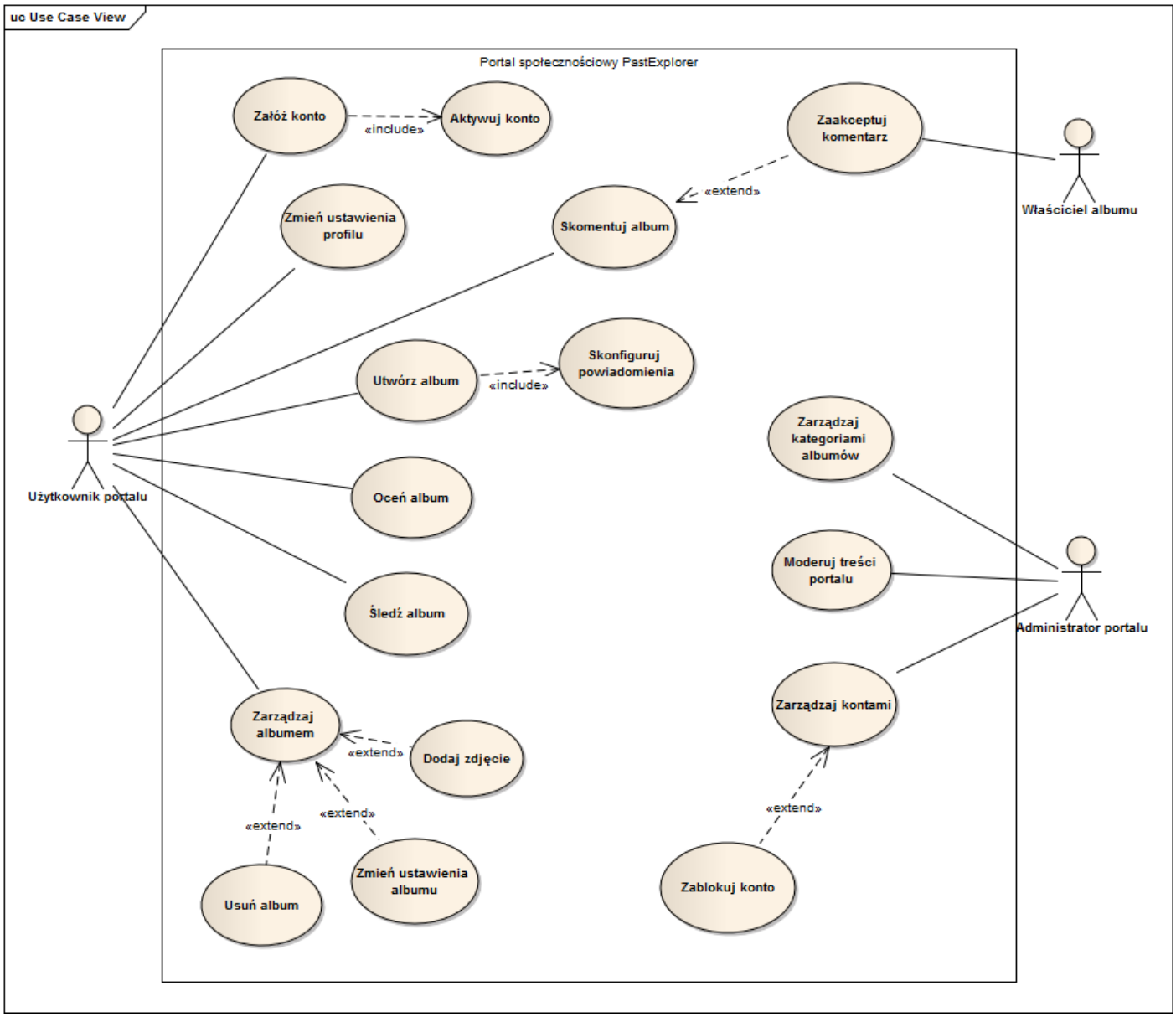
Aplikacja internetowa utworzona w ramach tego projektu pozwala za pomocą zdjęć umieszczanych przez użytkowników obserwować jak różnego rodzaju “rzeczy” (np. ludzie, budynki, krajobrazy) zmieniały się wraz z upływem czasu. Każdy użytkownik może zakładać albumy będące kolekcjami zdjęć zgodnych tematycznie 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 1: DIAGRAM PRZYPADKÓW UŻYCIA

Wykorzystane technologie

Poniżej zaprezentowana została lista technologii oraz narzędzi wykorzystanych do implementacji serwisu.

Warstwa logiki biznesowej

- Microsoft IIS – serwer aplikacyjny
- ASP.NET MVC (C#) – framework do tworzenia aplikacji internetowych w modelu MVC
- Android SDK (Java) – użyty do implementacji aplikacji mobilnej

Warstwa danych

- PostgreSQL – silnik bazodowanowy
- NHibernate – mapowanie obiektowo-relacyjne (ORM)

Warstwa prezentacji

- HTML
- CSS
- JavaScript (biblioteka jQuery)
- framework Twitter Bootstrap – predefiniowane elementy interfejsu użytkownika
- API Google Maps – wyświetlanie mapy z zaznaczonymi miejscami wykonania zdjęć

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.

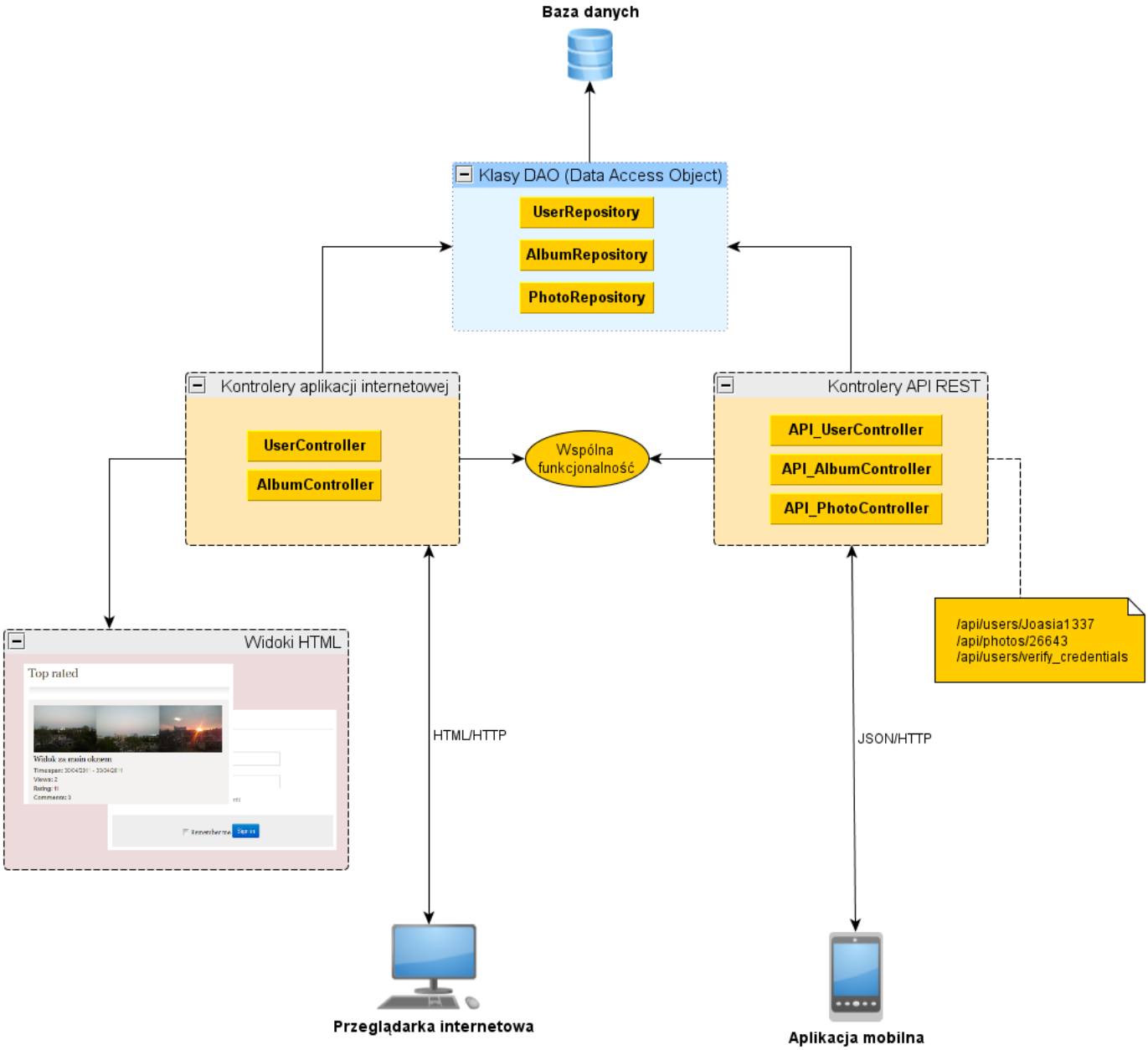
Proponowana architektura systemu byłaby bardziej spójna i efektywna, gdyby zamiast stworzenia osobnych grup kontrolerów mających za zadanie obsługę żądań różnych typów aplikacji, stworzyć dodatkową warstwę udostępniającą zestaw wszystkich funkcjonalności realizowanych przez portal w formie webserwisu. 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



RYSUNEK 2: ARCHITEKTURA SERWISU

Encje danych

Jednym z pierwszych etapów projektowania aplikacji było określenie encji, które miały modelować dane obecne w systemie oraz określenie relacji pomiędzy nimi. W wyniku analizy wydzielone zostały encje przedstawione w sekcji "Słownik danych". W sekcji "Diagram relacji encji" przedstawiono natomiast przełożenie tych encji na schemat bazodanowy oraz ustalone zostałe relacje pomiędzy poszczególnymi obiektami wraz z ich licznością.

Słownik danych

User - użytkownik

user_id, INT SERIAL PRIMARY KEY - *klucz główny, unikalne ID użytkownika*
login, varchar(255) NOT NULL - *login użytkownika w serwisie*
password, varchar(255) NOT NULL - *hasło zahashowane*
email, varchar(255) NOT NULL - *adres email użytkownika*
active, boolean NOT NULL - *czy konto jest aktywne*
date_of_birth, date - *data urodzenia*
about, TEXT - *informacje dodatkowe u użytkowniku*
notify_comment, boolean - *czy użytkownik ma być powiadamiany mailowo o komentarzach*
notify_photo, boolean - *czy wysyłane jest przypomnienie o zrobieniu nowego zdjęcia*
notify_subscr, boolean - *czy użytkownik ma być powiadamiany mailowo o nowych zdjęciach w obserwowanych* albumach
notification_period, smallint - *ilość dni, co jaką wysyłane jest w/w przypomnienie o zrobieniu zdjęcia*

Album – album ze zdjęciami

album_id, INT SERIAL PRIMARY KEY - *klucz główny, unikalne ID albumu*
user_id, INT NOT NULL - *klucz obcy do tabeli User, oznacza właściciela albumu*
category_id, INT NOT NULL - *klucz obcy do tabeli Category, oznacza kategorię, do której należy album*
name, varchar(255) NOT NULL - *nazwa albumu*
description, text - *opis albumu*
rating, smallint - *ocena albumu*
views, integer - *liczba odsłon albumu*
next_notification, timestamp - *kiedy ma być wysłane najbliższe powiadomienie do użytkownika o konieczności zrobienia zdjęcia do albumu*
public, boolean NOT NULL - *czy album jest publiczny, czy widoczny tylko dla użytkownika*
password, varchar(255) NOT NULL - *hasło zahashowane*
comments_allow, boolean NOT NULL - *czy możliwe jest dodawanie komentarzy do albumu*
comments_auth, boolean NOT NULL - *czy komentarze wymagają moderacji autora*

Category – kategoria albumu

category_id, INT SERIAL PRIMARY KEY - *klucz główny, unikalne ID kategorii*
name, varchar(255) NOT NULL - *nazwa kategorii*

Photo – zdjęcie

photo_id, INT SERIAL PRIMARY KEY - *klucz główny, unikalne ID fotografii*
album_id, INT NOT NULL - *klucz obcy do tabeli Album, oznacza album do którego należy zdjęcie*
date_taken, timestamp NOT NULL - *data zrobienia zdjęcia*
description, text - *opis zdjęcia*
file_path, text NOT NULL - *ścieżka do pliku ze zdjęciem*
loc_latitude, numeric(10,7) - *szerokość geograficzna, na której zostało zrobione zdjęcie*
loc_longitude, numeric(10,7) - *długość geograficzna, na której zostało zrobione zdjęcie*

TrustedUser – użytkownik zaufany, która ma dostęp do albumu o ograniczonym dostępie

album_id, INT PRIMARY KEY - *klucz główny, klucz obcy do tabeli Album, określa album do którego dostęp ma użytkownik*
user_id, INT PRIMARY KEY - *klucz główny, klucz obcy do tabeli User, określa użytkownika, który ma dostęp do albumu*

Subscription – subskrypcja albumu

album_id, INT PRIMARY KEY - *klucz główny, klucz obcy do tabeli Album, określa subskrybowany album*
user_id, INT PRIMARY KEY - *klucz główny, klucz obcy do tabeli User, określa użytkownika, który subskrybuje album*

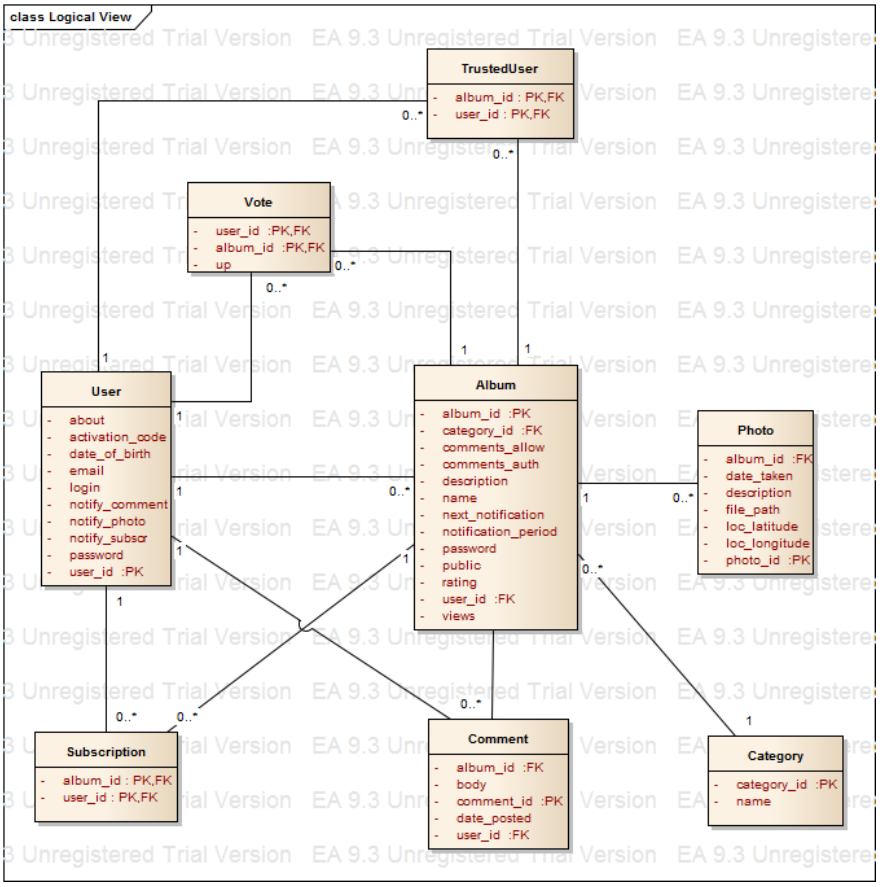
Comment - komentarz

comment_id, INT SERIAL PRIMARY KEY - *klucz główny, unikalne ID komentarza*
album_id, INT NOT NULL - *klucz obcy do tabeli Album, oznacza album do którego należy komentarz*
user_id, INT NOT NULL - *klucz obcy do tabeli User, oznacza autora komentarza*
date_posted, timestamp NOT NULL - *data opublikowania komentarza*
body, varchar(1000) NOT NULL - *treść komentarza*
accepted, boolean NOT NULL - *czy komentarz został zaakceptowany i może być wyświetlony*

Vote – głos oddany na album

album_id, INT PRIMARY KEY - *klucz główny, klucz obcy do tabeli Album, określa na który album został oddany głos*
user_id, INT PRIMARY KEY - *klucz główny, klucz obcy do tabeli User, określa użytkownika, który głosował na album*
up, boolean NOT NULL - *określa czy głos był pozytywny (true) czy negatywny (false)*

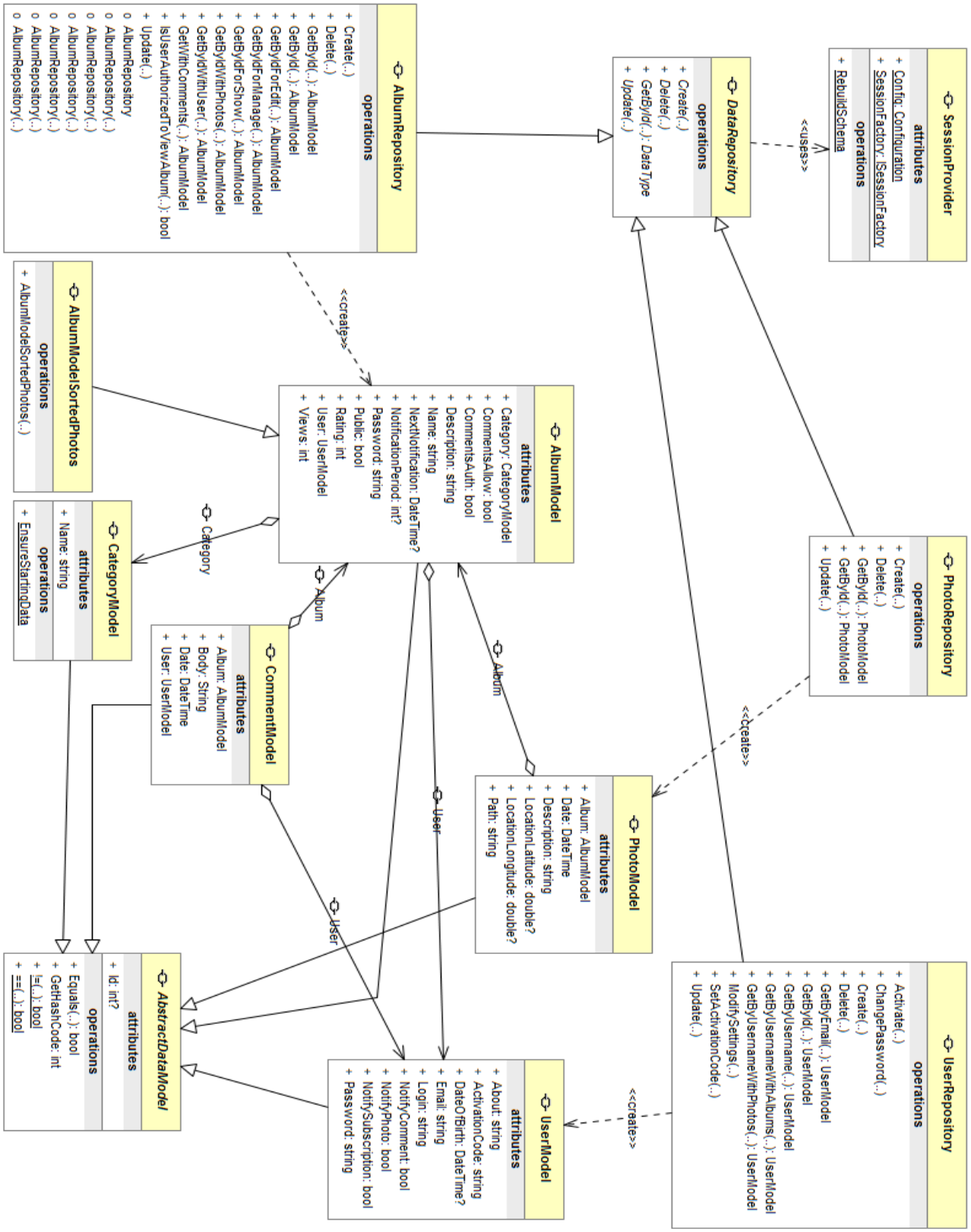
Diagram relacji encji (ERD)



RYSUNEK 3: DIAGRAM ERD DLA APLIKACJI

Klasy i ich odpowiedzialności

Warstwa modelu danych

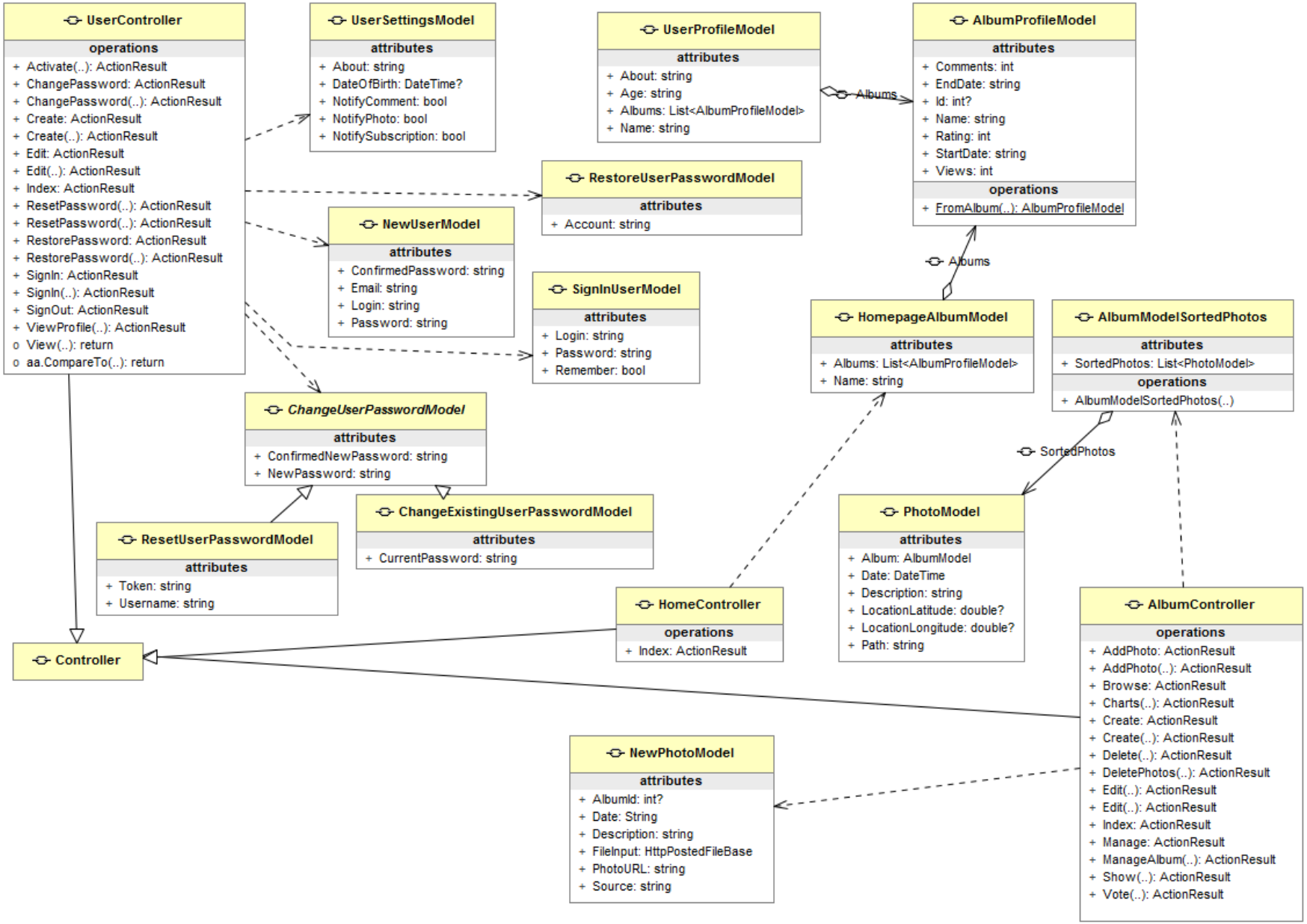


Wszystkie encje bazodanowe przedstawione w rozdziale „Encje danych” mapowane są na odpowiednie klasy modelu danych, dostępne z poziomu kodu źródłowego aplikacji. Oprócz tych klas w skład klas warstwy danych wchodzą również te realizujące wzorzec **Repository**, sterujący dostępem do instancji encji.

Nazwa klasy	Odpowiedzialność
SessionProvider	Implementuje wzorzec Singleton. Zapewnia dostęp do fabryki (interfejs ISessionFactory) pozwalającej na tworzenie obiektów kontrolujących sesję bazodanową.
AbstractDataModel	Abstrakcyjna klasa nadrzędna dla wszystkich klas modelu danych. Dla każdej klasy dziedziczącej zapewnia klucz surogatowy pozwalający na jednoznaczną identyfikację unikalnej encji. Ponadto zapewnia operacje pozwalające na poprawne przechowywanie encji w standardowych kontenerach frameworka .NET (identyfikacja na podstawie unikalnego klucza, generacja hasha, porównywanie encji).
DataRepository	Abstrakcyjna klasa nadrzędna dla wszystkich klas realizujących wzorzec Repository. Deklaruje zestaw podstawowych operacji na encjach (wstawianie, wyszukiwanie, modyfikacja oraz usuwanie), które każda klasa dziedzicząca musi implementować.
UserModel	Model encji użytkownika.
PhotoModel	Model encji zdjęcia.
CommentModel	Model encji komentarza.
AlbumModel	Model encji albumu.
CategoryModel	Model encji kategorii albumu.
UserRepository	Repozytorium udostępniające zestaw operacji na encjach użytkowników.
AlbumRepository	Repozytorium udostępniające zestaw operacji na encjach albumów.
PhotoRepository	Repozytorium udostępniające zestaw operacji na encjach zdjęć.

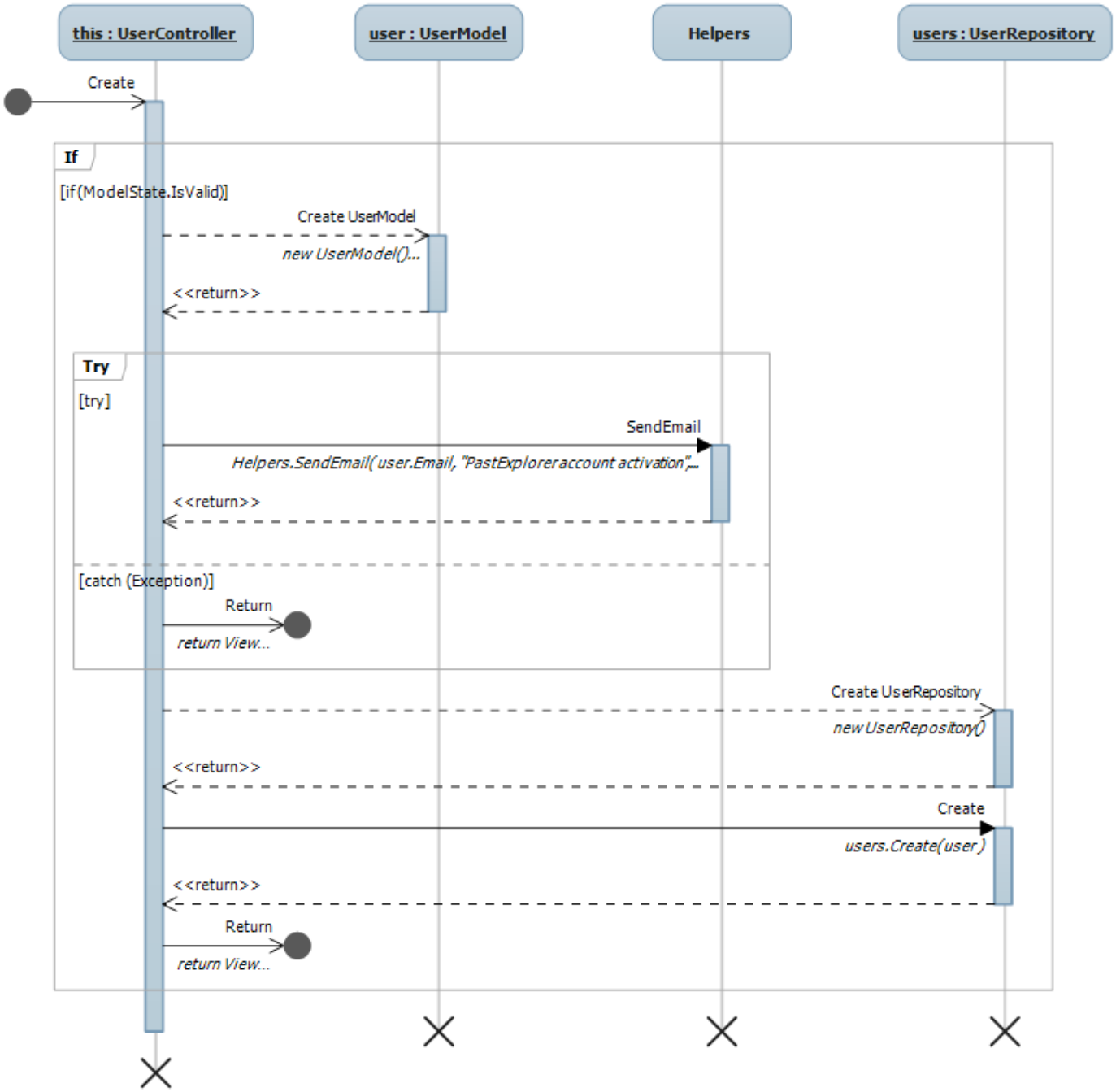
Warstwa logiki biznesowej

Nazwa klasy	Odpowiedzialność
Controller	Bazowa klasa dla wszystkich kontrolerów obsługujących akcje wykonywane przez użytkownika. Zapewniona przez framework ASP.NET MVC. Definiuje interfejs podstawowych operacji, które każdy kontroler musi implementować, oraz określa cykl życia kontrolera.
HomeController	Kontroler obsługujący stronę główną aplikacji internetowej, odpowiada za wyświetlanie miniatur albumów wybranych na podstawie określonych kryteriów, np. ostatnio modyfikowane, najpopularniejsze, itp.
UserController	Obsługuje akcje wykonywane na użytkownikach serwisu: rejestracje nowych, logowanie, pobieranie informacji na podstawie pseudonimu, modyfikacje danych profilowych, itp.
AlbumController	Obsługuje akcje wykonywane na albumach zdjęć: dodawanie nowych, edycję istniejących, wysyłanie zdjęć, przygotowywanie rankingów, itp.
NewUserModel	Encja będącą podzbiorem UserModel, przechowuje dane podane przez użytkownika w procesie rejestracji nowego konta.
SignInUserModel	Przechowuje informacje podane przez użytkownika podczas logowania (login, hasło).
NewPhotoModel	Przechowuje informacje podane przez użytkownika podczas tworzenia nowego albumu.


UserController
operations
+ Activate(..): ActionResult
+ ChangePassword: ActionResult
+ ChangePassword(..): ActionResult
+ Create: ActionResult
+ Create(..): ActionResult
+ Edit: ActionResult
+ Edit(..): ActionResult
+ Index: ActionResult
+ ResetPassword(..): ActionResult
+ ResetPassword(..): ActionResult
+ RestorePassword: ActionResult
+ RestorePassword(..): ActionResult
+ SignIn: ActionResult
+ SignIn(..): ActionResult
+ SignOut: ActionResult
+ ViewProfile(..): ActionResult
o View(..): return
o aa.CompareTo(..): return
UserSettingsModel
attributes
+ About: string
+ DateOfBirth: DateTime?
+ NotifyComment: bool
+ NotifyPhoto: bool
+ NotifySubscription: bool
UserProfileModel
attributes
+ About: string
+ Age: string
+ Albums: List<AlbumProfileModel>
+ Name: string
Albums
AlbumProfileModel
attributes
+ Comments: int
+ EndDate: string
+ Id: int?
+ Name: string
+ Rating: int
+ StartDate: string
+ Views: int
operations
+ FromAlbum(..): AlbumProfileModel
RestoreUserPasswordModel
attributes
+ Account: string
NewUserModel
attributes
+ ConfirmedPassword: string
+ Email: string
+ Login: string
+ Password: string
SignInUserModel
attributes
+ Login: string
+ Password: string
+ Remember: bool
Albums
HomepageAlbumModel
attributes
+ Albums: List<AlbumProfileModel>
+ Name: string
AlbumModelSortedPhotos
attributes
+ SortedPhotos: List<PhotoModel>
operations
+ AlbumModelSortedPhotos(..)
ChangeUserPasswordModel
attributes
+ ConfirmedNewPassword: string
+ NewPassword: string
SortedPhotos
ChangeExistingUserPasswordModel
attributes
+ CurrentPassword: string
ResetUserPasswordModel
attributes
+ Token: string
+ Username: string
PhotoModel
attributes
+ Album: AlbumModel
+ Date: DateTime
+ Description: string
+ LocationLatitude: double?
+ LocationLongitude: double?
+ Path: string
HomeController
operations
+ Index: ActionResult
AlbumController
operations
+ AddPhoto: ActionResult
+ AddPhoto(..): ActionResult
+ Browse: ActionResult
+ Charts(..): ActionResult
+ Create: ActionResult
+ Create(..): ActionResult
+ Delete(..): ActionResult
+ DeletePhotos(..): ActionResult
+ Edit(..): ActionResult
+ Edit(..): ActionResult
+ Index: ActionResult
+ Manage: ActionResult
+ ManageAlbum(..): ActionResult
+ Show(..): ActionResult
+ Vote(..): ActionResult
Controller
NewPhotoModel
attributes
+ AlbumId: int?
+ Date: String
+ Description: string
+ FileInput: HttpPostedFileBase
+ PhotoURL: string
+ Source: string

Obsługa akcji

Rejestracja użytkownika



Tworzenie nowego albumu

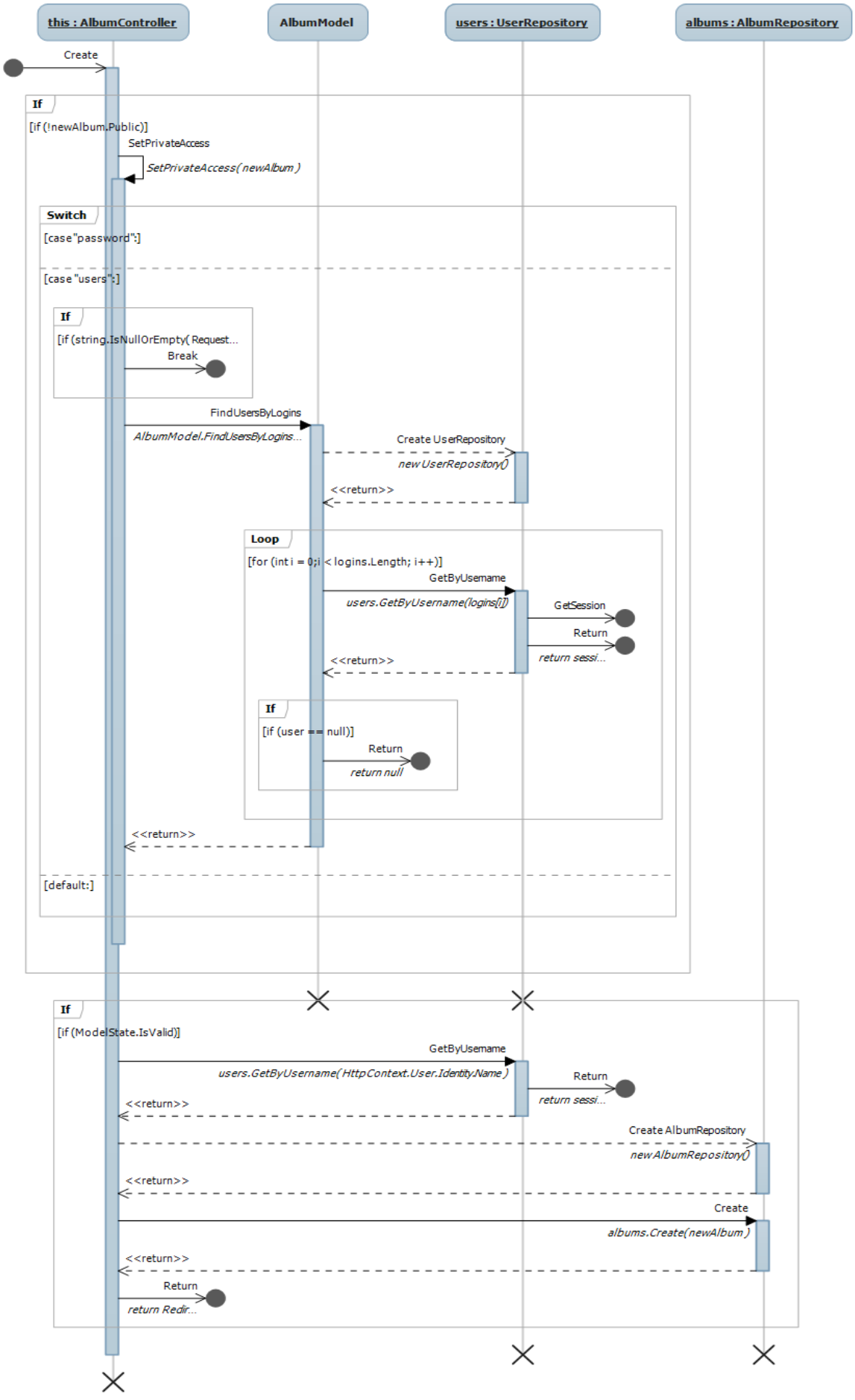


Diagram stanów aplikacji internetowej

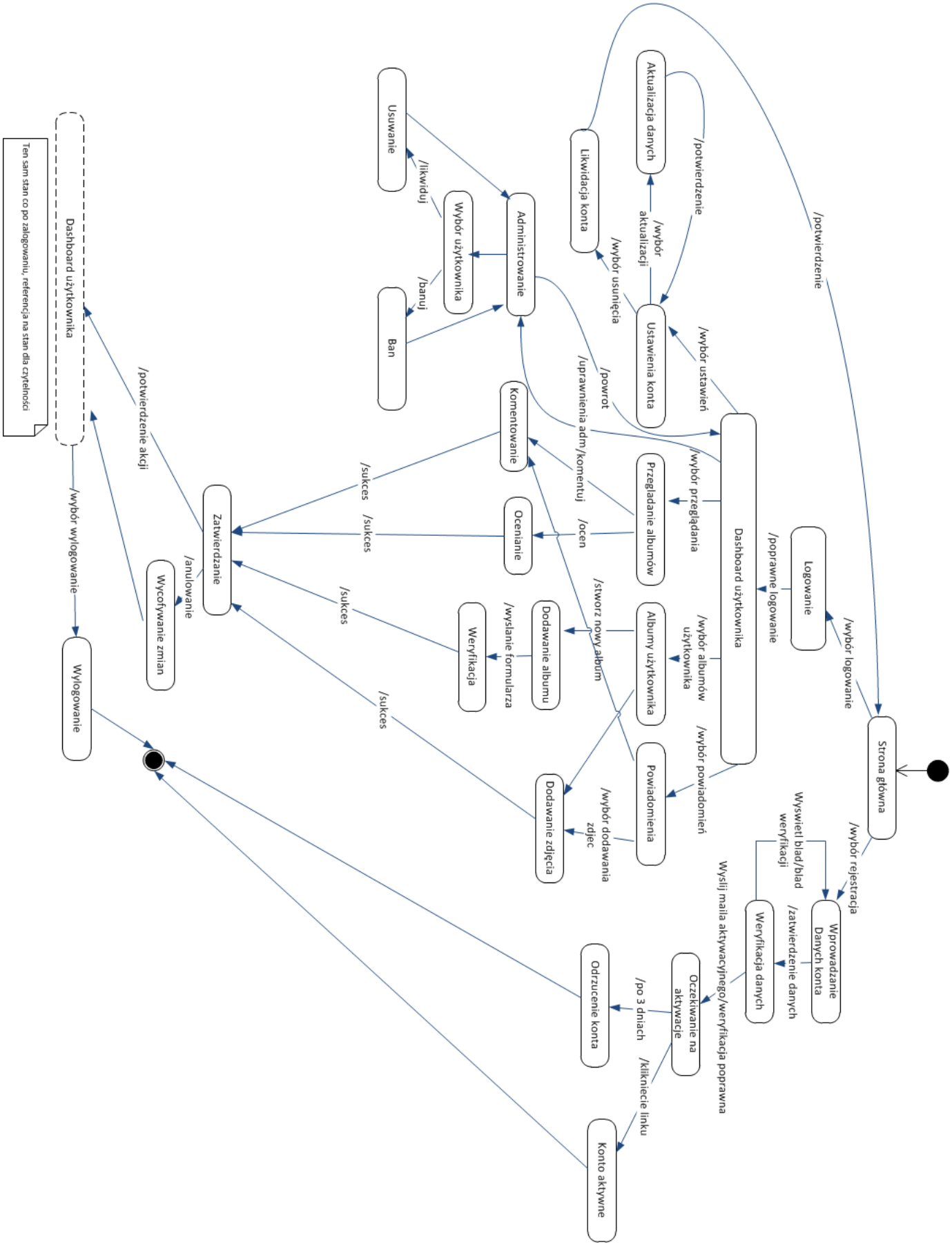


Diagram STD ukazuje możliwe przejścia w aplikacji.

Aby w pełni wykorzystywać możliwości aplikacji, użytkownik musi się zarejestrować. Po wprowadzeniu danych rejestracyjnych są one walidowane. Po poprawnej walidacji do aktywowania konta wymagane jest kliknięcie linku, który jest wysłany e-mailem. Jeśli użytkownik nie aktywuje konta w ciągu 3 dni, konto jest odrzucane.

Po zalogowaniu się użytkownik zostaje przekierowany na stronę z dashboardem użytkownika. Stamtąd istnieje możliwość przejścia do ustawień konta, zarządzania oraz przeglądania albumów.

Na stronie ustawień konta użytkownik może zmienić swoje dane oraz ustawienia powiadomień. Istnieje także opcja usunięcia konta, która realizowana jest po potwierdzeniu.

Użytkownik ma możliwość przeglądania albumów wg rankingów (najczęściej komentowane, najpopularniejsze, najwięcej wyświetleń, największe) lub wg kategorii. Po wybraniu konkretnego albumu ukazują się opcje komentowania oraz oceniania albumu.

Po przejściu do albumów użytkownika, możliwe jest dodanie albumu oraz dodanie zdjęcia. Stworzenie nowego albumu polega na wypełnieniu odpowiedniego formularza, w którym podaje się m.in. nazwę, opis, kategorię, opcje udostępniania, opcje powiadomień, możliwości komentowania. Następnie formularz jest weryfikowany, w przypadku braku błędów następuje przekierowanie do nowo stworzonego albumu. Proces dodawania nowego zdjęcia przebiega analogicznie, wymagane jest podanie m.in. albumu, daty wykonania zdjęcia oraz pliku ze zdjęciem.

Bezpieczeństwo aplikacji

Zabezpieczenia przed potencjalnymi atakami oraz złośliwymi danymi stanowią istotną kwestię dla ogólnodostępnych aplikacji internetowych. Ze względu na społecznościową charakterystykę naszej aplikacji, narażona jest ona na dodatkowe ryzyko związane z dużą ilością potencjalnych atakujących oraz dostępem do dużej ilości zgromadzonych danych. Dlatego zdecydowaliśmy się podjąć kroki, które uniemożliwiają ataki na aplikację.

Zabezpieczenia przed nieautoryzowanym edytowaniem danych

Edytowanie danych w aplikacji możliwe jest poprzez odpowiednie przyciski nawigacyjne. Po kliknięciu w edycję albumu, jego właściciel zostaje przekierowany na stronę edycji albumu (np. Album/Edit/1).

Należy zauważyć, że powyższy adres URL może być również wprowadzony do przeglądarki przez użytkownika, który nie jest właścicielem albumu. W takim przypadku nie może on uzyskać dostępu do edycji albumu. Za wyżej wymienione zachowanie odpowiada następujący kod umieszczony w kontrolerze w akcji Edit:

```
        if (!albums.isUserAuthorizedToEditAlbum(album, user))
            return View("NotAuthorizedEdit");
```

Metoda isUserAuthorizedToEditAlbum zaimplementowana została następująco:

```
public bool isUserAuthorizedToEditAlbum(AlbumModel album, UserModel user)
{
    // given user is the owner of the album
    if (album.User == user)
        return true;
    else
        return false;
}
```

Podobnie zostały zabezpieczone inne akcje umożliwiające autoryzowaną zmianę danych.

Zabezpieczenia przed wielokrotnym głosowaniem

Każdy zalogowany użytkownik ma możliwość oddania pozytywnego lub negatywnego głosu na album. Głosowanie należy zabezpieczyć w taki sposób, aby niemożliwe było wielokrotne głosowanie na ten sam album. W tym celu została utworzona tabela Votes, której struktura wygląda następująco:

album_id, INT PRIMARY KEY - *klucz główny, klucz obcy do tabeli Album, określa na który album został oddany głos*
user_id, INT PRIMARY KEY - *klucz główny, klucz obcy do tabeli User, określa użytkownika, który głosował na album*
up, boolean NOT NULL - *określa czy głos był pozytywny (true) czy negatywny (false)*

Istotnym faktem jest tutaj deklaracja user_id oraz album_id jako primary key. Dzięki temu niemożliwe jest oddanie więcej niż jednego głosu na jeden album przez użytkownika.

Autoryzowany dostęp do albumu

Widoczność albumu dla innych użytkowników może zostać ograniczona przez jego właściciela. Istnieją trzy metody prywatnego dostępu:

- dostęp poprzez hasło
- dostęp dla wyspecyfikowanych użytkowników
- dostęp tylko dla autora

W celu weryfikacji uprawnień do wyświetlenia albumu w akcji Show wywoływana jest metoda sprawdzająca uprawnienia. W przypadku braku uprawnień, użytkownik przekierowywany jest na stronę z odpowiednim komunikatem. Metoda sprawdzająca czy użytkownik może wyświetlić album została zaimplementowana następująco:

```
        public bool IsUserAuthorizedToViewAlbum(AlbumModel album, UserModel user, bool
passwordChecked)
        {
            if (album == null)
                throw new ArgumentException("album");

            // public album - eveyrone can view it
            if (album.Public)
                return true;

            // given user is the owner of the album
            if (album.User == user)
                return true;

            // given user is among trusted users for this album
            if (album.TrustedUsers != null && album.TrustedUsers.Contains(user))
                return true;

            // album has password access - allow if password was checked before
            if (album.Password != null)
                return passwordChecked;

            return false;
        }
```

SQL Injection

SQL injection jest popularnym atakiem na systemy www, który polega na wstrzykiwaniu zapytań do bazy danych poprzez pola formularzy. Używany przez nas framework NHibernate zabezpiecza przed takimi atakami poprzez filtrowanie otrzymanych danych. Należy jednak pamiętać, aby nie konstruować zapytań do bazy 'ręcznie', np:

```
// niebezpieczny kod !
string userName = ctx.GetAuthenticatedUserName();
        string query = "SELECT * FROM items WHERE owner = '"
                                + userName + "' AND itemname = '"
                                + ItemName.Text + "'";
        List items = sess.CreateSQLQuery(query).List();
```

W projekcie używamy konstrukcji 'SetParameter', dzięki której dane są filtrowane:

```
session.CreateQuery( "from UserModel where Login = :login" ).SetParameter( "login",
username )
```

Webservice – API w stylu REST

W ramach projektu postanowiliśmy 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)

API – listing z przykładowymi wywołaniami

1. 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"
        ]
    }
}
```


2. 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": []
    }
}
```


3. 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
    }
}
```


4. Wysyłanie zdjęcia

POST /api/photos

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

Autentykacja/autoryzacja użytkownika

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:

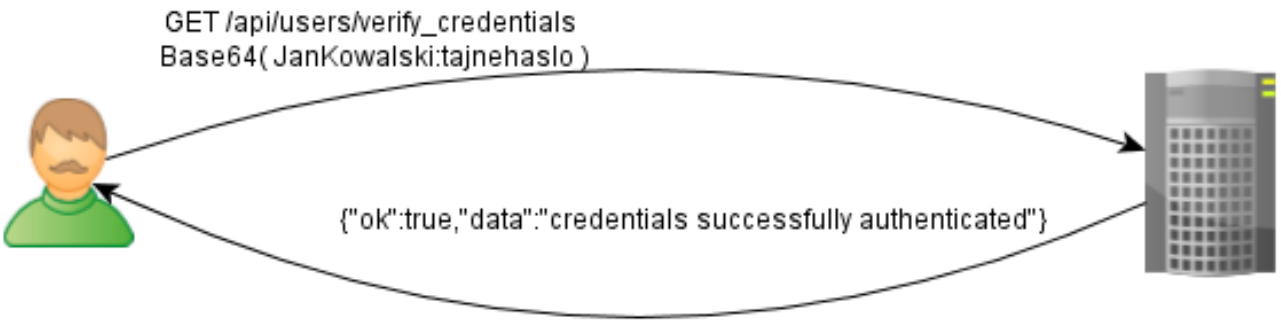
- prostota, autentykacja polega na dodaniu jednego nagłówka do zapytania HTTP z zakodowanymi w Base64 danymi użytkownika (login + hasło)
- brak konieczności przetrzymywania po stronie serwera żadnego stanu zalogowania użytkownika, jak w przypadku sesji, co naruszałoby postulaty modelu REST
- jest to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

Metoda ta jednak posiada pewne wady:

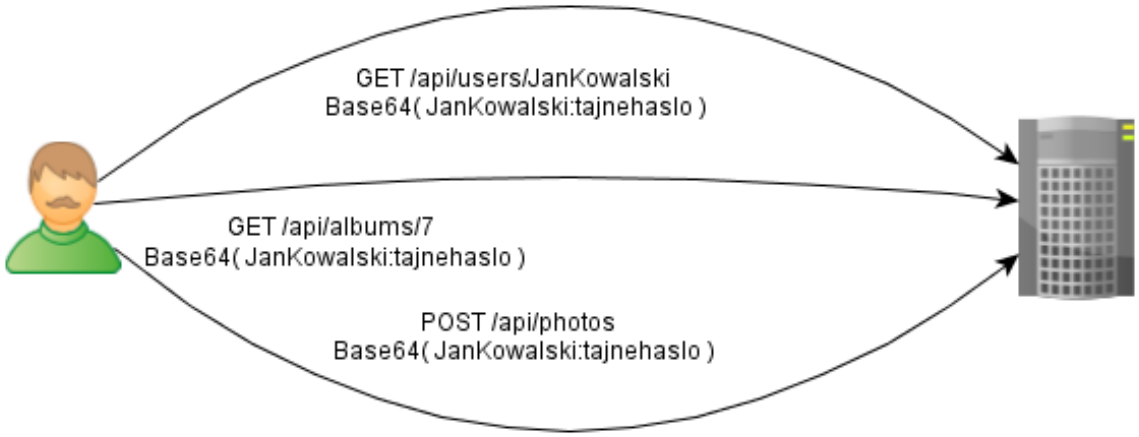
- dane użytkownika (w tym hasło) przesyłane są w zapytaniu HTTP tekstem jawnym, stąd konieczność komunikacji przez HTTPS (szyfrowanie SSL/TLS)
- dane użytkownika przesyłane są wraz z każdym zapytaniem, co może stwarzać niepotrzebne zagrożenie; bezpieczniejszym rozwiązaniem może być modyfikacja mechanizmu, 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 zaimplementowaliśmy)

Autentykacja użytkownika (dwustopniowa):

1. Weryfikacja danych logowania wprowadzonych przez użytkownika



2. 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.