|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | |  | **Wydział Informatyki i Zarządzania**  kierunek studiów: Informatyka  specjalność: wpisz właściwą  Praca dyplomowa - inżynierska  **Muzyczny serwis społecznościowy**  Mariusz Nabiałek  słowa kluczowe:  Ruby on Rails  AngularJS  Serwis Społecznościowy  krótkie streszczenie:  Praca polegała na wytworzeniu aplikacji webowej w formie serwisu społecznościowego dla ludzie interesujących się muzyką. Przedstawiono użyte technologie do wytworzenia aplikacji wraz z podziałem oraz strukturą. Rozpisano wymagania funkcjonalne, które w większości są typowe dla serwisów społecznościowych, czyli możliwość pisania treści w postaci postów, komentarzy, dodawanie zdjęć itp. Zaprezentowano projekt aplikacji, wraz z opisem wzorców. Pokazano sposób zaimplementowania części aplikacji. Na końcu zaproponowano możliwe kierunki rozwoju.   |  |  |  |  | | --- | --- | --- | --- | | opiekun pracy  dyplomowej | .................................................. | ....................... | ....................... | | *Tytuł/stopień naukowy/imię i nazwisko* | *ocena* | *podpis* |   *Do celów archiwalnych pracę dyplomową zakwalifikowano do:\**   1. *kategorii A (akta wieczyste)* 2. *kategorii BE 50 (po 50 latach podlegające ekspertyzie)*   *\* niepotrzebne skreślić*   |  | | --- | | pieczątka wydziałowa | |
|  |  |  | Wrocław 2015 |

Spis treści

[I. Streszczenie pracy 4](#_Toc436933613)

[Rozdział 1 Wstęp 5](#_Toc436933614)

[1.1. Motywacja i cel pracy 5](#_Toc436933615)

[Rozdział 2 Użyte technologie i wzorce projektowe 6](#_Toc436933616)

[2.1. Technologie 6](#_Toc436933617)

[2.1.1. AngularJS 6](#_Toc436933618)

[2.1.2. Ruby on Rails 7](#_Toc436933619)

[2.2. Wzorzec Projektowy MVC 7](#_Toc436933620)

[2.3. Wzorce projektowe W Ruby On Rails 12](#_Toc436933621)

[2.3.1 Active Record – serce modelu Ruby on Rails 12](#_Toc436933622)

[2.2.2. Asocjacje pomiędzy modelami 15](#_Toc436933623)

[2.4. Biblioteki 8](#_Toc436933624)

[2.4. RWD i Bootstrap 11](#_Toc436933625)

[2.5 AngularJS MVC 12](#_Toc436933626)

[Rozdział 3 Założenia projektowe 18](#_Toc436933627)

[3.1. Wymagania funkcjonalne 18](#_Toc436933628)

[3.2.Przypadki użycia 23](#_Toc436933629)

[Rozdział 4 Projekt aplikacji 59](#_Toc436933630)

[4.1 Aplikacja serwerowa 59](#_Toc436933631)

[4.1.1 Kontrolery w aplikacji 59](#_Toc436933632)

[4.1.2. API 62](#_Toc436933633)

[4.2 Frontend 63](#_Toc436933634)

[4.2.1 Organizacja aplikacji 63](#_Toc436933635)

[4.2.2. Resources - zasoby 64](#_Toc436933636)

[4.3. Projekt bazy danych i model klas 67](#_Toc436933637)

[Rozdział 5 Implementacja aplikacji 69](#_Toc436933638)

[5.1 Backend – aplikacja po stronie serwera 69](#_Toc436933639)

[5.1.1 Algorytmika 69](#_Toc436933640)

[5.1.2. API 69](#_Toc436933641)

[5.2. Frontend – aplikacja po stronie klienta 73](#_Toc436933642)

[5.2.1 Wyszukiwarka 73](#_Toc436933643)

[5.2.2. Tagowanie 74](#_Toc436933644)

[5.2.3 Przykładowe widoki, które zostały użyte w aplikacji 75](#_Toc436933645)

[Rozdział 6 Podsumowanie 78](#_Toc436933646)

[6.1. Podsumowanie 78](#_Toc436933647)

[6.2. Perspektywy rozwoju 78](#_Toc436933648)

[Bibliografia 80](#_Toc436933649)

[Spis rysunków 82](#_Toc436933650)

[Spis listingów 83](#_Toc436933651)

[A. Załącznik Dokumentacja API 88](#_Toc436933652)

[B. Załącznik Pozostałe Diagramy przypadków użycia 128](#_Toc436933653)

# Streszczenie pracy

Tematem pracy jest Muzyczny serwis społecznościowy. Celem pracy jest zaprojektowanie oraz zaimplementowanie serwisu społecznościowego przeznaczonego dla osób zainteresowanych muzyką i chcących poznać ludzi o podobnym zainteresowaniu. Przedstawiono platformy programistyczne (ang. Framework) AngularJS i Ruby on Rails, wraz z użytymi dodatkowymi bibliotekami. AngularJS jest to platforma działająca na bazie języka JavaScript, wykorzystująca wzorzec architektoniczny MVC, została ona użyta do stworzenia aplikacji po stronie klienta. Aplikacja po stronie serwera, działa na platformie Ruby on Rails, która napisana została w języku Ruby. Przedstawiono wymagania funkcjonalne takie jak pisanie postów, komentarzy, zakładanie grup, zespołów oraz wydarzeń, prowadzenie konwersacji, dodawanie zdjęć i wyszukiwanie. Następnie przedstawiono wzorzec architektoniczny MVC, wzorzec projektowy ActiveRecord, opisano technologię RWD oraz strukturę aplikacji wraz z opisem sposobu komunikacji z API serwera. Przedstawiono trasy API oraz sposób zaimplementowania funkcjonalności tagowania i wyszukiwania. Przedstawiono możliwe kierunki rozwoju aplikacji.

**Abstract**

The main topic of this work is Music social network service. The main goal is make project and implementation of application for people who are interested music and wants to meet new people who have the same interest. The application was developed in two frameworks AngularJS and Ruby on Rails, with additional libraries. AngularJS is Javascript framework. AngularJS use of MVC Pattern and it used in client side of application. The application on server side it's work on Ruby on Rails is Ruby framework. There are show functional requirements for example writing post, commenting, creating groups, bands and events, attending in conversations, adding photos and searching. There are described MVC design pattern and ActiveRecord pattern. There are explained RWD technology and structure of application, there are show how application on client side communicates with server API. There are described API routes and way to implemented tagging feature and search feature. On the end it described possible ways to improves of application.

# Wstęp

## 1.1. Motywacja i cel pracy

Ludzie zajmujący się hobbystycznie lub zawodowo muzyką, mogą spotykać się z wyzwaniem, który może wpłynąć na ich dalszą karierę muzyczną. Problem polega na tym, że w pewnym momencie granie samemu i dla samego siebie przestaje już wystarczać. Muzyk rozpoczyna, więc poszukiwania innych ludzi przy których mógłby rozwinąć swoje i ich umiejętności. Jednakże mimo dość dużego rozwoju portali społecznościowych ciężko jest znaleźć ludzi z poza kręgu znajomych, którzy grają np. na jakimś konkretnym instrumencie lub określony typ muzyki . Długie poszukiwania, które mogą trwać nawet tygodnie, mogą skutecznie zniechęcić do dalszych poszukiwań.

Kolejnym problemem to zorganizowanie ogólnodostępnego wydarzenia spotkania muzyków na wspólne tworzenie muzyki na żywo tzw. Jam Session. Większość serwisów społecznościowych umożliwiają tworzenie wydarzeń. Minusem, który można zauważyć, że w przypadku muzyki, ciężko jest znaleźć wydarzanie związane z określonym typem muzyki.

Z tych problemów zrodził się pomysł na rozwiązanie, jakim jest muzyczny serwis społecznościowy. Ideą, która by przyświecała tej aplikacji to łączenie ludzi o wspólnej pasji, jaką jest muzyka. Dzięki udostępnionej platformie ludzie zainteresowani muzyką mogą znaleźć i organizować wydarzenia, wyszukać grupę ludzi, którzy np. szukają zespołu i grają na specyficznym instrumencie i wymieniać się z nimi poglądami na tematy muzyczne lub inne.. Aplikacja również dostarczałaby możliwości promowania własnych zespołów, które dopiero rozpoczęły działalność.

Cel muzycznego serwisu społecznościowego byłby w pełni spełniony w sytuacji, gdyby zgromadziłby wokół siebie dużą ilość użytkowników, którzy by aktywnie z niej korzystali.

Rozdział 2 opisuje wykorzystane technologie oraz wzorce projektowe. Rozdział 3 opisuje wymaganie funkcjonalne, które zostały zdefiniowane do wytworzenia aplikacji. W rozdziale 4 omówiono architekturę aplikacji, przedstawiono również diagram klas oraz schemat bazy danych. W rozdziale 5 opisuje interfejs API biblioteki, oraz wybrane rozwiązania użyte w aplikacji po stronie klienta. Rozdział 6 przedstawia podsumowanie prac i możliwe kierunki rozwoju i perspektywy aplikacji.

# Użyte technologie i wzorce projektowe

Aplikację podzielono na dwie odrębne części. Działającą po stronie serwera i tą, która działa po stronie klienta. Do wytworzenia tej aplikacji wykorzystano platforma programistyczne (ang. framework) Ruby on Rails, który działa po stronie serwera i udostępnia interfejs API oraz AngularJS, który działa po stronie klienta.

## 2.1. Technologie

### 2.1.1. AngularJS

AngularJS [1] (aplikacja została wytworzona w wersji 1.4) jest platformą programistyczną napisaną w języku Javascript, który obecnie jest powszechnie wykorzystywany w aplikacjach działających po stronie klienta. W rzeczywistości AngularJS dostarcza programiście zestaw gotowych narzędzi oraz szkielet, za pomocą którego można w wydajny sposób wytworzyć aplikację. W przypadku Angulara, można zbudować aplikację internetową, bazując na jednym z najpopularniejszych wzorców architektonicznych, jakim jest MVC (model widok kontroler). Platforma ta cały czas jest wspierana i ulepszana przez firmę Google [1] [2].

Jak można się dowiedzieć, ze strony twórców biblioteki [3], AngularJS wspiera następujące przeglądarki internetowe:

1. Internet Explorer w wersji od 9 wzwyż
2. Firefox
3. Chrome
4. Safari
5. Opera
6. Natywną wersję przeglądarki Android Browser ( wzwyż wersji 2.3.x)

Na stronie nie ma podanej dokładnej listy przeglądarek pod którymi aplikacja napisana z wykorzystaniem AngularJS, byłaby niekompatybilna. Z informacji podanych na stronie [3] można dowiedzieć się, że przed wypuszczeniem kolejnych wersji, platforma jest testowana pod różnymi przeglądarkami, co powoduje, że większość błędów zostaje wychwycona i naprawiona. Wówczas programista korzystający z tej platformy może zająć się wytwarzaniem aplikacji. Konsekwencją dużej kompatybilności jest to, że aplikacja ta będzie dostępna nie tylko pod różnymi przeglądarkami, ale również urządzeniach mobilnych, wówczas więcej użytkowników będzie mogło z niej skorzystać.

Z książki [2] oraz strony poświęconej platformie [1] możemy przeczytać, że AngularJS zapewnia prosty sposób na dołączanie danych, manipulacje tzw. drzewem DOM, walidację i obsługę formularzy, reagowanie na zdarzenia oraz grupowanie kodu HTML w komponenty, które możemy użyć w różnych częściach naszej aplikacji. Funkcjonalności te zapewniają nam wbudowane, jak i stworzone przez programistę tzw. dyrektywy. Autor książki [2], opisuje tą funkcjonalność jako “rozszerzenie HTML w celu przygotowania podstaw dla bogatych i skomplikowanych aplikacji sieciowych”. Na stronie twórców platformy [1] można przeczytać o wbudowanych kilkunastu dyrektywach oraz innych funkcjonalnościach, które zapewniają nam podstawowe funkcjonalności np. walidację formularzy, ukrywanie elementów w specyficznych sytuacjach, zmianę widoków bez potrzeby przeładowania strony, obsługę zdarzeń, połączenie danego elementu z modelem itp.

### 2.1.2. Ruby on Rails

Jak podaje autor książki [4] „Rails jest środowiskiem pozwalającym na budowanie aplikacji WWW opartych na bazach danych.”. W przypadku aplikacji opisywanej w tej pracy RoR odgrywa roli aplikacji działającej po stronie serwera. Autor [4] dodatkowo dodaje informacje, że aplikacja ta nadaje się do „Rails najlepiej nadaje się do aplikacji, które muszą być tworzone szybko, mieć przejrzystą strukturę i być łatwe w utrzymaniu”.

Na stronie twórców [5] możemy przeczytać, że platforma ta cechuj się dwoma filozofiami:

**DRY (Don’t Repeat Yourself)** - oznacza to, że pisząc jakiś kod wykonujący to samo nie powinniśmy go powielać. Dzięki czemu taki kod jest lepiej zadbany, łatwiej go przetestować i zmniejszamy ryzyko popełnienia błędu

**COC (Convention Over Configuration) -** dzięki tej filozofii nie musimy mieć dużej ilości plików konfiguracyjnych by wdrożyć jakąś konwencję do naszej aplikacji webowej.

Platforma ta zbudowana jest z wykorzystaniem wzorca architektonicznego MVC. Dodatkowo zaletą jest potężna ilość rozszerzeń/bibliotek tzw. gemów, które rozszerzają możliwości Rails. Wspiera on różne systemy bazodanowe np. Postgres, SQLite, MySQL. Wymagane jest dogranie odpowiedniego rozszerzenia.

Ta platforma programistyczna działająca po stronie serwera będzie w aplikacji odpowiadać za :

1. Dostarczanie danych do aplikacji po stronie klienta, poprzez API
2. Zapis danych otrzymanych w żądaniu HTTP
3. Uwierzytelnianie użytkownika (za pomocą tzw. tokena)
4. Dodatkową walidację danych

Silnikiem bazy danych, wykorzystuje się SQLite, domyślnie wykorzystywany wraz z instalacją podstawowego projektu Ruby on Rails. W przyszłości należy rozważyć, czy zmiana silnika wpłynie na wydajność działanie portalu. Na stronie twórców tego silnika [6] podaje się, że silnik wydajny jest jako silnik dla aplikacji działającej po stronie serwera.

Dostęp do konkretnych zasobów zostało zrealizowane na podstawie interfejsu API wraz z odpowiednimi trasami typu REST.

## 2.2. Wzorzec Projektowy MVC

Jak możemy przeczytać w książce [4] „Wzorzec MVC szczególnie dobrze sprawdza się w aplikacjach sieciowych.”, na wzorcu tym jak już wcześniej wspomniano opiera się Ruby on Rails oraz AngularJS. Autor książki [4] uznaje, że „kluczem na zastosowanie wzorca MVC jest implementacja najważniejszego założenia, czyli podziału odpowiedzialności…”. Czytamy w [4], że „oznacza to, że dane modelu w aplikacji są oddzielone od logiki biznesowej i prezentacyjnej. Zgodnie z tym, możemy podzielić wzorzec MVC na trzy komponenty: model, kontroler oraz widok

Każdy, komponent odpowiada ma inną odpowiedzialność w działającej aplikacji. Zgodnie z tym, co pisze autor [4]. Model odpowiada, za przetwarzanie, pobieranie, odpowiedniego przekształcanie, tworzenie i innych operacji na danych. Dane te otrzymuje z magazynu danych, lub otrzymuje je z kontrolera, który otrzymał dostał dane od widoku na wskutek np. jakiejś interakcji użytkownika aplikacji. Istnieją dwa typy modeli: model domeny oraz model widoku

Zgodnie z tym z wzorcem MVC i podziału odpowiedzialności, wg. autora [4] model powinien posiadać dane z określonej domeny, oraz zapewniać interfejs umożliwiający operację na nich. Posiadanie innych odpowiedzialności takich jak wyświetleniem danych czy różne operacje na wskutek interakcji użytkownika z widokiem, jest złamaniem wzorca.

Zadaniem kontrolera jest połączenie widoku i modelu. Jak podaje autor [4], „Kontrolery dodają logikę biznesową (nazywaną funkcjami) do zakresu, które jest podzbiorem modelu”. Z tego płynie wniosek, że kontroler reaguje odmiennie na wskutek danych, które przekazał widok w odpowiedzi na jakąś interakcje z użytkownikiem. Do dobrych praktyk, które zaleca autor [4], jest to, że kontroler powinien zawierać logikę odpowiedzialną za przekazanie danych widokowi, logikę odpowiedzialną za inicjalizację zakresu i tą która ten zakres będzie aktualizowała na wskutek działań użytkownika.

Do złych praktyk autor [4] przetacza, że zły kontroler, posiada część odpowiedzialności modelu lub widoku, albo wykonuje operacje z poza swojego zakresu.

Widok natomiast odpowiada jedynie za wyświetlenie danych dla użytkownika i przesyła dane, które powstały na wskutek interakcji użytkownika z interfejsem. Dlatego też powinien zawierać odpowiedzialną za to logikę. Do złych praktyk zalicza się, nadawanie widokowi logiki modelu lub kontrolera.

## 2.3. Biblioteki

W aplikacjach wykorzystano dodatkowe biblioteki rozszerzające możliwości obu platform.

**RABL (Ruby API Builder Language), j**ak podaje autor tej biblioteki [7] dzięki tej bibliotece możemy zbudować odpowiedź API w wybranym formacie. Możemy wybrać atrybuty obiektu, które mają być zawarte w odpowiedzi np. identyfikator wpisu i jego treść. Dodatkowo możemy tworzyć własne atrybuty, które przechowują wynik jakiejś metody wykonanej na modelu. RABL oferuje dodatkowe zagnieżdżanie obiektów w obiekcie np. mając jakiś wpisu, możemy pobrać określone dane twórcy. Możemy też dodawać określone atrybuty, tylko po spełnieniu określonego warunku. Biblioteka umożliwia zwrot danych w formatach XML, MessagePack, PList, BSON i JSON. Bibliotekę skonfigurowano by zwracała dane w formacie JSON. Dodatkowo wymagana jest biblioteka [7] **OJ (Optimized JSON)**. Jest to parser dla obiektów JSON. Dokumentację biblioteki OJ możemy odnaleźć na stronie [8].

**Decent exposure**,jak możemy się dowiedzieć z dokumentacji [9], dzięki tej bibliotece możemy pobrać obiekt do przetwarzania bazując na tym jakie parametry zostały użyte w żądaniu. Wówczas nie musimy budować obiektu w sposób taki, jak zdefiniowali to twórcy platformy Ruby On Rails. Na listingu 1 i listingu 2 przedstawiono różnicę pomiędzy standardowym sposobie pobrania obiektu i po wykorzystaniu biblioteki.

Listing 1 Sposób pobierania obiektu w Rails

*def destroy*

*@post = Post.new(params[:post])*

*@post.save*

*end*

*def show*

*@post = Post.find(params[:post\_id])*

*end*

*def destroy*

*@post = Post.find(params[:post\_id])*

*@post.destroy*

*end*

Listing 2 Sposób pobierania obiektu z wykorzystaniem biblioteki Decent Exposure

*# plik posts\_concern.rb*

*expose (:post) { get\_post }*

*def get\_post*

*if params[:id]*

*Post.find(params[:id])*

*else*

*Post.new(params[:post])*

*end*

*end*

*# plik posts\_controller.rb*

*def create*

*post.save*

*end*

*def show*

*post*

*end*

*def destory*

*post.destroy*

*end*

Jak widać z listingu 12, dzięki tej bibliotece możemy napisać jedną metodę do pobierania obiektu i tworzenia obiektu. Dzięki czemu zachowujemy konwencję nie powtarzania kodu, czyli DRY.

**Devise token auth**, jak możemy się dowiedzieć z dokumentacji [10] jest nakładką na bibliotekę Devise [11], która zapewnia sposób kontroli sesji, uwierzytelniania użytkownika, tworzenie konta, logowanie za pomocą tzw. OAuth itp. Token generowany jest na bazie 16 znakowego base64. Natomiast hasło użytkownika generowane jest za pomocą standardowej biblioteki w Ruby on Rails o nazwie BCrypt, z dokumentacji [12] wynika, że algorytm bcrypt został stworzony przez Neilsa Probos’a oraz Davida Mazieres’a. Zgodnie z dokumentacją [12], hasło jest zabezpieczane za pomocą algorytmu o tej samej nazwie. Dodatkowo biblioteka ta zapewnia możliwość uwierzytelnienia użytkownika przy próbie dostępu do jakiegokolwiek zasobu, bez posiadania odpowiednich uprawnień czy bez podania danych autoryzacyjnych, użytkownik nie może się dostać do zasobu. W aplikacji użyto logowania za pomocą adresu email. Biblioteka ta wymaga dodatkowo zainstalowania dwóch dodatkowych bibliotek: Devise [11], Rack-cors [13], Omniauth [14].

**Paperclip** zgodnie z tym, co autor pisze w dokumentacji biblioteki [15]. rozszerzenie to zapewnia obsługę przetwarzania zdjęć przesłanych do serwera. Wymaga jednak zainstalowania otwartego oprogramowania ImageMagick [16], który służy do np. konwersji obrazu, skalowania. Dzięki temu, przy przesłaniu zdjęcia na serwer, jest ono przetwarzane, zapisane do wybranych formatów, skalowane do wybranych rozdzielczości. Wówczas przy próbie odwołania do zdjęcia, możemy odwołać się do zdjęcia o określonym formacie czy rozdzielczości. Przydatne jest to np. przy tworzeniu awatarów z wybranych zdjęć, tak żeby nie wykorzystywać zdjęcia o pełnej rozdzielczości.

Należy jednocześnie zaznaczyć, że biblioteka, gdy uruchamiamy aplikację po stronie serwera na systemie operacyjnym Windows, zawiera błąd, który uniemożliwia przetworzenie zdjęcia. Zaleca się wówczas rozwiązanie polegające na zastąpienie jednej linijki kodu, która odpowiada za błąd [17], w chwili gdy ufamy użytkownikom. Ponieważ aplikację tworzy się w środowisku deweloperskim możemy zastosować dane rozwiązanie.

**Pry-byebug**, zgodnie z tym, co możemy przeczytać w dokumentacji [18] bibliotekę tą wykorzystujemy do debugowania kodu, możemy to zrealizować za pomocą wpisania w interesujące nas miejsce komendy binding.pry wówczas gdy aplikacja natrafi na tą linijkę, wykonywanie dalszego skryptu jest blokowane. Wówczas możemy zdebugować kod.

**Factory girl rails** i **rspec rails**, te dwie biblioteki zapewniają budowanie obiektów testowych[19], jak i wygodny sposób napisania przypadków testowych[20] do aplikacji.

**Angular ui router**, zgodnie z dokumentacją [21] biblioteka zapewnia możliwość zagnieżdżania widoków w widokach. Opcja ta obecnie nie jest dostępna w Angularze, obecnie można w danym oknie zagnieździć tylko jeden widok za pomocą dyrektywy ng-view. Biblioteka ta operuje na tzw. stanach. Oznacza to, że skrypt rozpoznaje obecny stan w którym znajduje się dany widok i wówczas dołącza odpowiedni kod HTML. Dany stan możemy wywołać np. za pomocą odpowiedniej metody, wpisując odpowiedni adres URL w oknie przeglądarki.

**Ng token auth** biblioteka [22] ta jest stworzona przez autora, który stworzył inną bibliotekę, którą już tutaj wymieniono Devise Token Auth. Dodatek ten po stronie klienta zapewnia użyteczne metody, jak mechanizmy logowania, rejestracji, potwierdzanie tożsamości użytkownika po każdej wykonanej akcji do serwera, poprzez walidację posiadanego przez użytkownika tokena.

**angular file upload** biblioteka [23] ta dostarcza funkcjonalność wysłania wybranych przez użytkownika plików do serwera.

**Angular cookie**, jest to lekka biblioteka do tworzenia plików ciasteczek (cookies) [24]. Korzystam z tej biblioteki a nie ze standardowych metod angulara z uwagi na to, że biblioteka Ng token auth, wymaga używania tej biblioteki do przechowywania danych potwierdzających tożsamość użytkownika.

**Bootstrap** dostarcza komponenty, takie jak okno modalne, menu kontekstowe, użyteczne klasy np. do stylowania przycisków [25]. Opis tej biblioteki zostanie rozwinięty w rozdziale dotyczącym opisu dokumentacji aplikacji. Do poprawnego działania biblioteki potrzebna jest biblioteka **JQuery,** dokumentację tej biblioteki możemy znaleźć na stronie [26].

## 2.4. RWD i Bootstrap

W aplikacji wykorzystano darmową bibliotekę Bootstrap, jak możemy przeczytać na stronie internetowej poświęconej tej bibliotece [25], jak i książce Thoriqa Firdausa [27], “Bootstrap to system stworzony przez Marka Otto na wewnętrzny użytek portalu Twitter”.

Bootstrap dostarcza nam zestaw gotowych klas i komponentów napisanych w języku Javascript, co pozwala na szybkie zbudowanie elastycznego interfejsu użytkownika.

Dlaczego budowanie elastycznej strony www jest ważne?

Z uwagi na to, że coraz więcej ludzi korzysta ze swoich urządzeń mobilnych, które wyposażone są w przeglądarki internetowe. Jednakże gdyby strona wyświetlała się tak samo jak na ekranie komputera, wówczas prawdopodobnie użytkownik miałby duży problem z poruszaniem się po takiej stronie na dużo mniejszym ekranie. Dzięki responsywności, strona staje się bardziej dostępna pod względem możliwości odwiedzania jej z różnych urządzeń mobilnych.

W Bootstrapie obecne są tzw. zapytania medialne . W książce [28], możemy przeczytać, że “Zapytania medialne pozwalają na wywołanie określonych właściwości CSS w zależności od parametrów urządzenia”. Oznacza to, że możemy stworzyć odpowiednie style dla określonych rozdzielczości ekranu. W skutek czego możemy w zupełnie inny sposób wyświetlić naszą stronę na różnych rozdzielczościach.

Niestety nie zawsze wykorzystanie biblioteki pozwoli nam stworzenie wszystkiego co możemy potrzebować. Daje ona jedynie fundament, czasami istnieje potrzeba dostosowania biblioteki do swoich potrzeb. Przykładowymi zabiegami zmieniającymi wygląd domyślny to np. zmiana marginesów, koloru. Zmian możemy dokonywać w kodzie źródłowym, który udostępniła firma Twitter. Kod ten jest napisany w preprocesorze LESS i SASS . Dla potrzeb swojej aplikacji korzystam z preprocesora CSS Less.

Jak możemy przeczytać w książce [27], “...preprocesory CSS to rozszerzenie możliwości kaskadowych stylów. Przy jego użyciu można tworzyć arkusze w bardziej dynamiczny sposób. Dzięki preprocesorom w CSS można używać zmiennych i funkcji podobnie jak w językach PHP i Javascript.”

Gotowe komponenty w języku JavaScript, które wykorzystano w aplikacji to menu kontekstowe, w którym przechowuje opcje takie jak edycja postu czy usuwanie. Kolejnym komponentem, który wykorzystano w aplikacji to tzw. okno modalne. Pojawia się ono w chwili dodawania zespołu do wydarzenia czy przy opisywaniu zdjęcia przed jego wysłaniem do serwera. Niektóre jednak dodatkowe funkcjonalności takie jak np. możliwość tagowania czy wyszukiwarki zaimplementowano z wykorzystaniem biblioteki AngularJS.

## 2.5. Wzorce projektowe W Ruby On Rails

### 2.5.1 Active Record – serce modelu Ruby on Rails

Jak możemy przeczytać w książce [4] „Active Record obsługuje zagadnienia dostępu do bazy danych” oraz to, że „Martin Fowler skatalogował wzorzec projektowania mechanizmu Active Record w swojej książce pod tytułem Patterns of Enterprise Architecture (Przegląd wzorców występujących w architekturach oprogramowania dla firm).”

Wzorzec ten został zaimplementowany do modelu Ruby on Rails. W ramach tego wzorca zostały zaimplementowane cztery metody podstawowe. Odpowiedzialne kolejno za tworzenie, odczytywanie i usuwanie danych z bazy danych. Autor [4] wspomina o tym, że dzięki temu projekt staje się prosty i zapewnia mapowanie pomiędzy tabelami, a obiektami.

Jednakże, jak podaje dokumentacja [29] i autor książki [4], Active Record w Ruby on Rails ma zaimplementowane kilka funkcji rozszerzających. Dokumentacja [30] wymienia takie metody, jak find, find\_by, oraz inne, które zostaną opisane w tym rozdziale.

Listing 3 Tworzenie modelu postu

*class Post < ActiveRecord::Base*

*end*

Z dokumentacji [29] wynika, że konstrukcja przedstawiona na listingu 3, zapewnia już, że dany model będzie zmapowany wraz z tabelą “posts” w bazie danych, o ile ona istnieje. Dodatkowo jak podaje autor [4], „Rails dodaje atrybuty w sposób automatyczny, bazując na kolumnach dostępnych w bazie danych.”.

Listing 4 Atrybuty tabeli posts

Posts

*id, int*

*content, varchar*

Definiując następującą prostą przykładową tabelę znajdującą się na listingu 4.

Jedynym z ograniczeń podanych również przez autora [4], jest to, że nie możemy używać kluczy złożonych, bez dodatkowego kodu lub wtyczki. Autor zaleca , więc „stosować konwencje Rails i tworzyć identyfikatory jednokolumnowe”. Takie też zalecenie wykorzystuje w swojej aplikacji.

Dzięki temu, obiekt klasy Post, będzie miał atrybuty identyfikator oraz content. W ramach tego, że został użyty Active Record, będą dostępne podstawowe operacje CRUD (tworzenie, odczyt, aktualizacja, usuwanie) wyglądają następująco zgodnie z dokumentacją [29]:

**Create – tworzenie**

Listing 5 Przykład tworzenia obiektów

#tworzenie za pomocą metody create

*post.create(content: “Jakiś tekst”)*

*#tworzenie za pomocą metody new*

*post.new(content: “Jakiś tekst”)*

*post.save*

Różnicą pomiędzy tymi dwoma metodami przedstawionymi na listingu 5, polega na tym [29], że create tworzy obiekt i zapisuje go do bazy danych, a new tworzy jedynie lokalną kopię, którą trzeba zapisać wywołując metodę save.

**Read - odczyt**

Odczyt jest najłatwiejszą metodą, którą możemy wykonać. W ramach tego wzorca Ruby on Rails udostępnia nam szereg funkcji, za pomocą których możemy odczytać interesujący nas obiekt lub obiekty w bazie danych.

Są to m.in. następujące metody wyszukujące (lista pochodzi bezpośrednio ze strony [30]):

bind, create\_with, distinct, eager\_load, extending, from, group, having, includes, joins, limit, lock, none, offset, order, preload, readonly, references, reorder, reverse\_order, select, uniq, where

Istnieją również metody takie jak find, find\_by [30]. Metody **find, find\_by** wykorzystują w rzeczywistości klauzule where w zapytaniach SQL. W przypadku **find** domyślnie wykorzystuje wyszukiwanie po identyfikatorze, a find\_by wykorzystuje wybrany atrybut.

Listing 6 Log po wywołaniu metody post.all

*Post Load (69.0ms) SELECT "posts".\* FROM "posts"*

Metoda all, pobiera wszystkie rekordy z danego modelu wraz z wszystkimi atrybutami, jeśli nie ograniczyliśmy jej za pomocą innych metod. Wówczas tworzy się zapytanie do bazy danych, które możemy zaobserwować w logu przedstawionym na listingu 6.

Listing 7 Log po wywołaniu metody post.select(:content)

*Post Load (0.0ms) SELECT "posts"."content" FROM "posts"*

Metoda **select**, zapewnia możliwość zaznaczenia wybranych kolumn(atrybutów) z bazy danych. Wówczas tworzy nam się zapytanie do bazy danych, które możemy zaobserwować w listingu 7.

Listing 8 Log po wywołaniu metody post.offset(2).limit(10)

*Post Load (1.0ms) SELECT "posts".\* FROM "posts" LIMIT 10 OFFSET 2*

Metody **offset** i **limit**, zapewniają ograniczoną ilość pobranych rekordów (limit) i przesunięcie wyszukiwanych rekordów (offset) Wówczas powyższa metoda zwróci nam wszystkie posty poczynając od rekordu z identyfikatorem równym 3 do 13, jeśli istnieją. Wynik metody możemy zaobserwować na listingu 8.

Listing 9 Log po wywołaniu metody Post.where("id=?",2).order(:created\_at).first

Post Load (0.0ms) SELECT "posts".\* FROM "posts" WHERE (id=2) ORDER BY "posts"."created\_at" ASC LIMIT 1

Metody **last** i **first** działają analogicznie, z tym, że first zwraca pierwszy z pobranych rekordów, a last zwraca ostatni rekord z pobranych rekordów. Metoda **order** służy do posortowania rekordów wg. określonego kryterium. Metoda **where** ogranicza rekordy, które spełnia dany warunek. Zapytanie utworzone i wyświetlone w logu możemy zaobserwować na listingu 9.

Należy również wspomnieć również, że Ruby on Rails w ramach interfejsu wyszukiwania udostępnia podstawowe funkcje agregujące takie jak: count, max, min, avg, sum.

**Update – aktualizacja**

Podchodząc do aktualizacji możemy przyjąć dwie różne drogi [29]: zmianę wartości określonych parametrów, a następnie wywołanie metody update lub aktualizację za pomocą metody. Obie metody zostały zaprezentowane na listingu 10 i listingu 11.

Listing 10 Podejście z zmianą wartości określonych parametrów

*post = Post.find(2)*

*post.content = “Sak”*

*post.update*

Listing 11 Aktualizacja za pomocą parametru

*post = Post.find(2)*

*post.update(content: “Sak”)*

**Delete - usuwanie**

Usuwanie polega na pobraniu rekordów, które chcemy usunąć, a następnie wywołać metodę destroy [29]. Przykład wywołania tej metody zaprezentowano na listingu 12.

Listing 12 Usuwanie obiektu za pomocą metody destroy

|  |
| --- |
| *user = User.find\_by(name: „Maniek”)*  *user.destroy* |

W aplikacji korzysta się z tej metody jedynie przy usuwaniu zdjęć. W przypadku postów, komentarzy, wydarzeń, grup itp. nie są one usuwane z bazy danych, a jedynie oznaczone jako usunięte. W dzisiejszych czasach potrzebne są mechanizmy, które pozwoliłyby np. dostarczenie dowodu, że dany post powstał, wówczas w bardzo szybki sposób możemy dostarczyć informacji czy takie zdarzenie miało miejsce.

### 2.5.2. Asocjacje pomiędzy modelami

Ruby on Rails udostępnia wiele użytecznych mechanizmów. Jedną z ważnych funkcjonalności, to tworzenie asocjacji pomiędzy encjami na poziomie modelu. Wówczas nie jest wymagane tworzenie kluczy obcych w bazie danych, bo robi to za nas Ruby on Rails. W ramach tego komponentu platforma udostępnia kilka możliwości połączenia z obiektami.

Jak podaje autor [4],”Relację, nazwaną również asocjacją, definiujemy w trzech częściach: makra relacji, asocjacji lub celu oraz opcjonalnej tablicy dodatkowych parametrów”.

Aplikacja wykorzystuje jedynie dwa typy asocjacji opisanych w dokumentacji [31]:

1. belong\_to: zgodnie ze słowami autora [4] relacja ta obsługuje relację typu „wiele do jednego”

2. has\_many: Zgodnie z opisem w książce [4] „Metoda has\_many jest drugą stroną relacji belongs\_to”.

Dzięki temu, możemy sprawić, że możemy odnieść się z pozycji do obiektu powiązanego relacją z innym obiektem.

Relację możemy modyfikować, za pomocą następujących opcji:

1. through: jak podaje autor [4]. dziki temu możemy połączyć dwa modele, korzystając z dodatkowego modelu łączącego. Dzięki czemu możemy stworzyć relację wiele do wielu.

2. polymorphic: Modyfikowanie relacji za pomocą parametru polymorphic, na stronie [31] możemy przeczytać, że “z użyciem relacji polimorficznej dany model może być połączony z więcej niż jednym modelem za pomocą pojedynczej asocjacji.” Przykładowo post ma pole postable\_type i postable\_id. W przypadku gdy chcemy dodać post w ramach grupie, wówczas typem postu będzie “Group”, natomiast id będzie wskazywało na konkretną grupę w której został utworzony post.

Zaletą korzystania z tego typu rozwiązania jest to, że w chwili, gdy chcemy dodać możliwość np. pisania postów w wydarzeniu. Wówczas wystarczy jedynie dodać odpowiednią asocjację. Kolejną zaletą jest to, że nie tworzymy w bazie danych dodatkowych tabel.

## 2.6 AngularJS MVC

W AngularJS tworzę niestandardowy sposób routingu oparty na stanach, dzięki wykorzystaniu biblioteki Angular UI Router [21]. Każdy stan stworzony za pomocą biblioteki Angular UI Router, ma swój kontroler. Sposób tworzenia stanu zgodny z dokumentacją [21], możemy zaobserwować na listingu 13.

Listing 13 Definicja przykładowego stanu

*app.config(["$stateProvider",function($stateProvider) {*

*$stateProvider*

*.state("home.specificUser.photos",{*

*url:"/photos",*

*templateUrl:"partials/photos.html",*

*resolve:{*

*photos: function(PhotosService,$stateParams){*

*var data= {};*

*data.user\_id = $stateParams.user\_id;*

*data.page =0 ;*

*return PhotosService.fetchPhotos(data);*

*}*

*},*

*controller: "UserPhotosCtrl"*

*})*

W ramach kontrolera inicjalizujemy zmienne w tzw. $scope. Wówczas zmienna ta jest widoczna w widoku aplikacji. Jednakże jeśli chcemy w ramach widoku wprowadzić jakieś akcje np. obsługę kliknięcia przycisku, w zmiennej $scope musimy zdefiniować funkcje anonimową, która te akcje obsłuży. W ramach tworzenie kontrolera trzeba wykonać zgodnie z dokumentacją [32] konstrukcję, którą możemy zaobserwować na listingu 13. Należy dodać, że parametr resolve, służy nam do pobrania potrzebnych danych, przed przejściem do określonego stanu.

Listing 14 Konstrukcja kontrolera

*app.controller(“NazwaKontrolera”,[“$scope’,function($scope){}]);*

Stworzenie kilka mniejszych kontrolerów po stronie klienta pozwala stworzyć małe fragmenty kodu. Dzięki takiej konstrukcji tworzymy bardzo łatwo zarządzany kod.

Pojedynczymi widokami zajmuje się Angular UI Router dołączając odpowiednie pliki HTML znajdujące się w katalogu partials. Możemy to zaobserwować na listingu 13. Model jest widoczny w widoku, dzięki wykorzystaniu zmiennej $scope. Następnie jest dowiązany za pomocą dyrektyw wbudowanych w AngularJS lub za pomocą znaków „{{nazwaZmiennej}}”.

# Założenia projektowe

## 3.1. Wymagania funkcjonalne

W ramach aplikacji użytkownik będzie mógł uczestniczyć, zakładać wydarzenia. Komunikować się ze społecznością za pomocą komentarzy, wiadomości prywatnych, postów czy to na swoim profilu czy na tzw. grupie. Grupa składa się z paru użytkowników, którzy np. lubią tą samą muzykę, są z tego samego miasta. Dodatkowo użytkownik może znaleźć swój ulubiony zespół lub założyć własny. W aplikacji znajduje się również mechanizm wydarzeń, w ramach których użytkownik może uczestniczyć. Wydarzenie może być założone przez indywidualnego użytkownika, w ramach grupy czy zespołu. Wydarzenie ma swój opis, zespoły, które uczestniczą w wydarzeniu, gatunek muzyki z którym jest związane, ma określoną datę początku i zakończenia oraz miejsce.

Dodatkowo wyszukiwarka pomaga wyszukiwać użytkowników, wydarzenia, grupy oraz zespoły. Tagowanie natomiast zapewnia możliwość oznaczenia użytkowników, wydarzeń, grup i zespołów. Dodatkowo każdy użytkownik, zespół, grupy mogą dodawać zdjęcia w ramach profilu. Każda grupa , użytkownik, zespół ma swój awatar.

W ramach aplikacji zostały zdefiniowany następujące historyjki użytkownika:

1. Moduł rejestracji:

1. Jako niezarejestrowany użytkownik chcę się zarejestrować.
2. Jako niezalogowany użytkownik chcę się zalogować.

2. Moduł profil użytkownika:

1. Jako użytkownik chcę opisać swoją historie na profilu.
2. Jako użytkownik chcę opisać o sobie na profilu.
3. Jako użytkownik chcę opisać swoją karierę muzyczną.
4. Jako użytkownik chcę zaznaczyć na jakich instrumentach gram.
5. Jako użytkownik chcę wybrać czy szukam zespołu czy nie.
6. Jako użytkownik chcę ustawić kiedy mam urodziny.
7. Jako użytkownik chcę ustawić z jakiego miasta jestem.
8. Jako użytkownik chcę zmienić swój nick.
9. Jako użytkownik chcę zmienić swoje hasło.

3. Moduł Strony Głównej

1. Jako użytkownik chcę zobaczyć posty grup, użytkowników, zespołów na głównej stronie.

4. Moduł postów na profilu użytkowników.

1. Jako użytkownik chcę napisać post na swoim profilu.
2. Jako użytkownik chcę usunąć swój post.
3. Jako użytkownik chcę edytować swój post.
4. Jako użytkownik chcę napisać komentarz do swojego postu lub postów innych użytkowników.
5. Jako użytkownik chce widzieć posty.
6. Jako użytkownik chce widzieć komentarze.
7. Jako użytkownik chcę usunąć swój komentarz do postu.
8. Jako użytkownik chcę edytować swój komentarz do postu.

5. Moduł profilu innego użytkownika:

1. Jako użytkownik chcę widzieć na czym gra inny użytkownik.
2. Jako użytkownik chcę widzieć czy dany użytkownik szuka zespołu.
3. Jako użytkownik chcę widzieć opisaną historie użytkownika.
4. Jako użytkownik chcę widzieć opis użytkownika.
5. Jako użytkownik chcę widzieć z jakiego miasta jest użytkownik.
6. Jako użytkownik chcę widzieć jaką karierę miał użytkownik.
7. Jako użytkownik chcę widzieć kiedy jakiś użytkownik ma urodziny.

6. Moduł zdjęć:

1. Jako użytkownik chcę widzieć zdjęcia jakie dodał użytkownik.
2. Jako użytkownik chcę dodawać zdjęć.
3. Jako użytkownik chcę usuwać zdjęć.
4. Jako użytkownik chcę ustawić wybrane swoje zdjęcie jako awatar.
5. Jako użytkownik chcę komentować zdjęcia.
6. Jako użytkownik chcę zmieniać opis swojego zdjęcia.
7. Jako użytkownik chcę wyświetlić opis zdjęcia.
8. Jako użytkownik chcę czytać komentarze innych użytkowników dotyczących zdjęcia.
9. Jako użytkownik chcę edytować komentarz do zdjęcia zespołu.
10. Jako użytkownik chcę usunąć komentarz do zdjęcia zespołu.

7. Moduł Tagowania

1. Jako użytkownik chcę tagować ludzi.
2. Jako użytkownik chcę tagować zespołów.
3. Jako użytkownik chcę tagować wydarzeń.
4. Jako użytkownik chcę tagować grup.

8. Moduł przynależności

1. Jako użytkownik chcę przeglądać swoje wydarzenia w których uczestniczę.
2. Jako użytkownik chcę przeglądać grupy do których należę.
3. Jako użytkownik chcę przeglądać zespoły do których należę lub które obserwuje.
4. Jako użytkownik chcę przeglądać wydarzenia w których dany użytkownik uczestniczy.
5. Jako użytkownik chcę przeglądać grupy do których dany użytkownik należy.
6. Jako użytkownik chcę przeglądać zespoły do których użytkownik należy lub które .obserwuje.

9. Moduł znajomości

1. Jako użytkownik chcę zawierać znajomość z użytkownikiem.
2. Jako użytkownik chcę usunąć znajomość z użytkownikiem.
3. Jako użytkownik chcę akceptować znajomość z danym użytkownikiem.
4. Jako użytkownik chcę odrzucić zaproszenie danego użytkownika.

10. Moduł profilu zespołu (skład zespołu):

1. Jako użytkownik chcę założyć zespołu.
2. Jako użytkownik chcę zmieniać nazwę zespołów.
3. Jako użytkownik chcę ustawić miasto z którego pochodzi zespół.
4. Jako użytkownik chcę edytować opis zespołu.
5. Jako użytkownik chcę ustawić start działalności zespołu.
6. Jako użytkownik chcę edytować historie zespołu.
7. Jako użytkownik chcę edytować opis członków zespołu.
8. Jako użytkownik chcę edytować gatunki muzyczne, które gra zespół.
9. Jako użytkownik chcę usunąć zespół.

11. Moduł przynależności do zespołu:

1. Jako użytkownik chcę dodać członków do zespołu.
2. Jako użytkownik chcę usunąć członka z zespołu.
3. Jako użytkownik chcę wyświetlać członków zespołu.
4. Jako użytkownik chcę filtrować ludzi którzy szukają zespołu po określonych kryteriach.

12. Moduł zdjęć zespołów:

1. Jako użytkownik chcę widzieć zdjęcia dodane przez zespół.
2. Jako użytkownik chcę dodawać zdjęcia do zespołu.
3. Jako użytkownik chcę usunąć zdjęcia do zespołu.
4. Jako użytkownik chcę ustawić awatar zespołu.
5. Jako użytkownik chcę komentować zdjęcia zespołu .
6. Jako użytkownik chcę edytować opis zdjęcia zespołu .
7. Jako użytkownik chcę wyświetlić opis zdjęcia zespołu.
8. Jako użytkownik chcę edytować komentarz do zdjęcia zespołu.
9. Jako użytkownik chcę usunąć komentarz do zdjęcia zespołu.
10. Jako użytkownik chcę wyświetlać komentarze do zdjęcia zespołu.

13. Moduł wydarzeń zespołów:

1. Jako użytkownik chcę akceptować zaproszenia na wydarzenia jako zespół.
2. Jako użytkownik chcę usuwać zaproszenia na wydarzenia jako zespół.
3. Jako użytkownik chcę wyświetlać zaproszenia na wydarzenia jako zespół.
4. Jako użytkownik chcę organizować wydarzenia jako zespół.

14. Moduł obserwatora zespołu:

1. Jako użytkownik chcę wyświetlić miasto z którego pochodzi zespół.
2. Jako użytkownik chcę wyświetlić opis zespołu.
3. Jako użytkownik chcę wyświetlić start działalności zespołu.
4. Jako użytkownik chcę wyświetlić historie zespołu.
5. Jako użytkownik chcę wyświetlić opis członków zespołu.
6. Jako użytkownik chcę wyświetlić gatunki muzyczne, które gra zespół.
7. Jako użytkownik chcę wyświetlić ludzi którzy obserwują zespół.
8. Jako użytkownik chcę zacząć obserwować zespół.
9. Jako użytkownik chcę skończyć obserwować zespół.

15. Moduł postów zespołu

1. Jako użytkownik chcę komentować posty zespołu.
2. Jako użytkownik chcę edytować swoje komentarze.
3. Jako użytkownik chcę usuwać komentarze.
4. Jako użytkownik chcę wyświetlać posty zespołów.
5. Jako użytkownik chcę pisać posty jako zespół.
6. Jako użytkownik chcę edytować posty jako zespół.
7. Jako użytkownik chcę usuwać posty jako zespół.

16. Moduł profilu grupy

1. Jako użytkownik chcę mieć możliwość założenia grupy.
2. Jako użytkownik (administrator grupy) chcę zmieniać nazwę grupy.
3. Jako użytkownik (administrator grupy) chcę usunąć grupę.
4. Jako użytkownik chcę edytować opis grupy.
5. Jako użytkownik chcę wyświetlać opis grupy.
6. Jako użytkownik chcę edytować nazwę grupy.
7. Jako użytkownik chcę wyświetlać nazwę grupy.

17. Moduł zdjęć grupy:

1. Jako użytkownik chcę widzieć zdjęcia dodane do grupy.
2. Jako użytkownik chcę dodawać zdjęcia do grupy.
3. Jako użytkownik chcę usunąć zdjęcia do grupy.
4. Jako użytkownik chcę ustawić awatar grupy.
5. Jako użytkownik chcę komentować zdjęcia grupy.
6. Jako użytkownik chcę edytować opis zdjęcia grupy.
7. Jako użytkownik chcę wyświetlić opis zdjęcia grupy.
8. Jako użytkownik chcę edytować komentarz do zdjęcia grupy.
9. Jako użytkownik chcę usunąć komentarz do zdjęcia grupy.
10. Jako użytkownik chcę wyświetlać komentarze do zdjęcia grupy.

18. Moduł uczestnika grupy

1. Jako użytkownik chcę dołączyć do grupy.
2. Jako użytkownik chcę odejść z grupy .
3. Jako użytkownik chcę organizować wydarzenia w ramach grupy.
4. Jako użytkownik chcę wyświetlić ludzi którzy są w grupie .
5. Jako użytkownik chcę mianować użytkownika jako administratora.

19. Moduł postów grupy

1. Jako użytkownik chcę komentować posty grupy.
2. Jako użytkownik chcę edytować swoje komentarze.
3. Jako użytkownik chcę usuwać grupy.
4. Jako użytkownik chcę wyświetlać posty grupy.
5. Jako użytkownik chcę pisać posty.
6. Jako użytkownik chcę edytować posty.
7. Jako użytkownik chcę usuwać posty.

20. Moduł Wydarzeń

1. Jako użytkownik chcę stworzyć wydarzenie.
2. Jako użytkownik (organizator) chcę mianować użytkownika, który dołączył do wydarzenia organizatorem.
3. Jako użytkownik (organizator) chcę ustawić z jaką muzyką związane jest wydarzenie.
4. Jako użytkownik (organizator) chcę edytować opis wydarzenia.
5. Jako użytkownik (organizator) chcę zapraszać zespoły na wydarzenie.
6. Jako użytkownik (organizator) chcę ustawić miejsce wydarzenia.
7. Jako użytkownik (organizator) chcę ustawić datę końca i początku wydarzenia.

21. Moduł uczestnictwa w wydarzeniach

1. Jako użytkownik chcę dołączać do wydarzenia.
2. Jako użytkownik chcę odchodzić z wydarzenia.
3. Jako użytkownik chcę widzieć kto będzie uczestniczył w wydarzeniu.
4. Jako użytkownik chcę wyświetlić jaką muzyką związane jest wydarzenie.
5. Jako użytkownik chcę wyświetlić opis wydarzenia.
6. Jako użytkownik chcę wyświetlić zaproszone zespoły na wydarzenie.
7. Jako użytkownik chcę wyświetlić miejsce wydarzenia.
8. Jako użytkownik chcę wyświetlić datę końca i początku wydarzenia.

22. Moduł wyszukiwarki

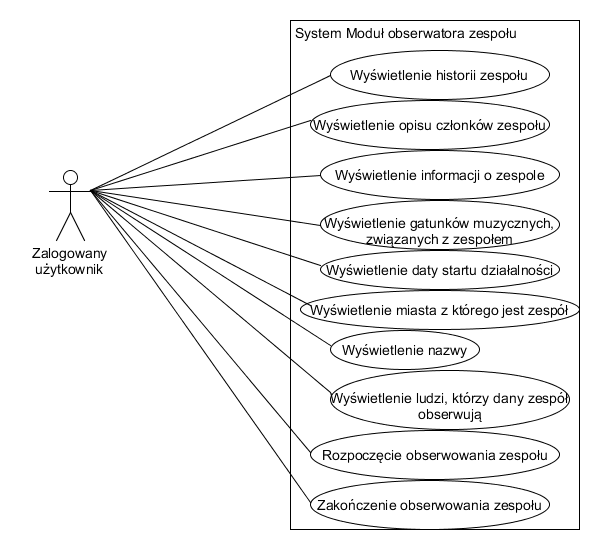
1. Jako użytkownik chcę wyszukiwać wydarzenia po nazwie.
2. Jako użytkownik chcę wyszukiwać grupy po nazwie.
3. Jako użytkownik chcę wyszukiwać użytkowników po nazwie.
4. Jako użytkownik chcę wyszukiwać zespoły po nazwie.
5. Jako użytkownik chcę wyszukiwać wydarzenia po nazwie i filtrami.
6. Jako użytkownik chcę wyszukiwać grupy po nazwie.
7. Jako użytkownik chcę wyszukiwać użytkowników po nazwie i filtrami.
8. Jako użytkownik chcę wyszukiwać zespoły po nazwie i filtrami.

23. Moduł konwersacji

1. Jako użytkownik chcę tworzyć konwersacje.
2. Jako użytkownik chcę zapraszać do konwersacji użytkowników.
3. Jako użytkownik chcę wypraszać z konwersacji użytkowników.
4. Jako użytkownik chcę wychodzić z konwersacji.
5. Jako użytkownik chcę czytać wiadomości w konwersacji.
6. Jako użytkownik chcę widzieć kto uczestniczy w konwersacji.
7. Jako użytkownik chcę zmieniać nazwę konwersacji.
8. Jako użytkownik chcę wysyłać wiadomości.
9. Jako użytkownik chcę wyświetlić nazwę konwersacji.

## 3.2.Przypadki użycia

Kierując się wskazówkami dotyczącymi tworzenia diagramów UML, znajdującymi się w książce [33]. Stworzyłem następujące diagramy przypadków użycia i opisałem wybrane z nich.



Rysunek 3.1 Moduł obserwatora zespołu

**Przypadek użycia**: Rozpoczęcie obserwowania zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.1, ten przypadek użycia opisuje możliwość rozpoczęcie obserwowania zespołu.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi się znajdywać na profilu zespołu.

Użytkownik musi nie obserwować zespołu.

Zespół musi istnieć w bazie danych.

**Przebieg**:

1. Użytkownik klika obserwuj zespół.
2. System przetwarza dane.
3. System dodaje użytkownika jako obserwatora zespołu.

**Przypadek użycia**: Zakończenie obserwowania zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.1, ten przypadek użycia opisuje możliwość zakończenia obserwowania zespołu.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi obserwować zespołu.

Użytkownik musi się znajdywać na profilu zespołu.

Zespół musi istnieć w bazie danych.

**Przebieg**:

1. Użytkownik klika skończ obserwować zespół.
2. System przetwarza dane.
3. System usuwa użytkownika jako obserwatora zespołu.

**Przypadek użycia**: Wyświetlenie jakie gatunki muzyczne gra zespół

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.1, ten przypadek użycia opisuje możliwość wyświetlania, jakie gatunki muzyczne gra zespół.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Gatunki muzyczne muszą być wybrane.

Zespół musi istnieć w bazie danych.

Użytkownik musi znajdywać się na profilu zespołu

**Przebieg**:

1. Użytkownik klika przycisk informacje o zespole.
2. System wybiera gatunki muzyczne które gra zespół.
3. System wyświetla gatunki muzyczne które gra zespół.

**Alternatywny przebieg**:

Jeśli zespół nie wybrał żadnych gatunków muzycznych, które gra zespół, system wyświetla informacje "Zespół nie wybrał żadnych gatunków muzycznych.".

**Przypadek użycia**: Wyświetlenie historii zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.1, ten przypadek użycia opisuje możliwość wyświetlania historii zespołu.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Pole musi być wypełnione

Zespół musi istnieć w bazie danych.

**Przebieg**:

1. Użytkownik klika przycisk informacje o zespole.
2. System wyświetla historię zespołu

**Alternatywny przebieg**:

1. Jeśli zespół nie wypełnił pola system wyświetla informacje "Zespół nie napisał swojej historii”

**Przypadek użycia**: Wyświetlenie opisu zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.1, ten przypadek użycia opisuje możliwość wyświetlania opisu zespołu.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Pole musi być wypełnione.

Zespół musi istnieć w bazie danych.

Użytkownik musi znajdywać się na profilu zespołu.

**Przebieg**:

1. Użytkownik klika przycisk informacje o zespole.
2. System wyświetla opis zespołu.

**Alternatywny przebieg**:

1. Jeśli zespół nie wypełnił pola system wyświetla informacje "Zespół nie napisał swojego opisu”

**Przypadek użycia:** Wyświetlenie miasta zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.1, ten przypadek użycia opisuje możliwość wyświetlania, skąd jest dany zespół.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Miasto musi być wybrane.

Zespół musi istnieć w bazie danych..

Użytkownik musi znajdywać się na profilu zespołu.

**Przebieg**:

1. Użytkownik klika przycisk informacje o zespole.
2. System wyświetla miasto, z którego pochodzi zespołu.

**Alternatywny przebieg:**

1. Jeśli zespół nie wypełnił pola miasta system wyświetla informacje "Zespół nie wybrał miasta”

**Przypadek użycia:** Wyświetlenie daty startu rozpoczęcia działalności

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.1, ten przypadek użycia opisuje możliwość wyświetlanie daty startu rozpoczęcia działalności

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Pole musi być wypełnione

Zespół musi istnieć w bazie danych.

Użytkownik musi znajdywać się na profilu zespołu.

**Przebieg**:

1. Użytkownik klika przycisk informacje o zespole.
2. System wyświetla datę startu rozpoczęcia działalności zespołu.

**Alternatywny przebieg**:

1. Jeśli zespół nie wypełnił pola system wyświetla informacje "Zespół nie ustawił daty startu działalności"

**Przypadek użycia**: Wyświetlenie opisu członków zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.1, ten przypadek użycia opisuje możliwość wyświetlanie opisu skłądu zespołu

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Pole musi być wypełnione

Zespół musi istnieć w bazie danych.

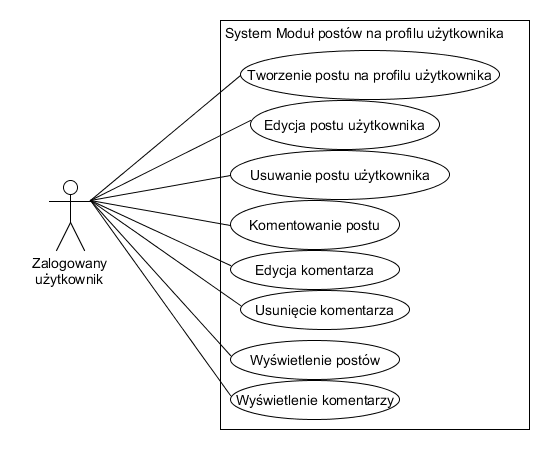
Użytkownik musi znajdywać się na profilu zespołu.

**Przebieg**:

1. Użytkownik klika przycisk informacje o zespole.
2. System wyświetla opis składu zespołu.

**Alternatywny przebieg**:

Jeśli zespół nie wypełnił pola system wyświetla informacje "Zespół nie ustawił opisu składu zespołu"



Rysunek 3.2 Moduł postów na profilu użytkownika

**Przypadek użycia**: Dodanie postu na swoim profilu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.2, ten przypadek użycia opisuje możliwość dodania wpisu na profilu użytkownika, którego jest właścicielem.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być właścicielem profilu.

Użytkownik musi wypełnić pole, co najmniej jednym znakiem drukowanym.

**Przebieg**:

1. Użytkownik wypełnia treść postu.
2. System przetwarza dane.
3. System dodaje post.
4. System wyświetla wśród postów dodany post.

**Alternatywny przebieg**:

a) W przypadku błędu podczas przetwarzania postu (brak treści postu):

System wyświetla informacje o niewypełnionym poście.

b) Użytkownik rezygnuje z dodania postu.

**Przypadek użycia**: Usunięcia postu na swoim profilu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.2, ten przypadek użycia opisuje możliwość usunięcia postu na profilu użytkownika

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być właścicielem profilu.

Użytkownik musi być właścicielem postu.

**Przebieg**:

1. Użytkownik wybiera opcje usunięcia postu.
2. System wyświetla informacje “Czy chcesz usunąć post?”.
3. Użytkownik klika na przycisk “tak”.
4. System przetwarza dane.
5. System usuwa post.
6. System aktualizuje wyświetlaną listę postów.

**Alternatywny przebieg**:

1. Użytkownik klika na przycisk “nie” - ponownie wyświetla się lista postów.

**Przypadek użycia**: Edycja postu na swoim profilu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.2, ten przypadek użycia opisuje możliwość edycji postu na profilu użytkownika.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być właścicielem profilu.

Użytkownik musi być właścicielem postu.

Użytkownik musi wypełnić pole co najmniej jednym znakiem drukowanym.

Post musi istnieć w bazie danych.

**Przebieg**:

1. Użytkownik klika edytuj post.
2. Użytkownik edytuje treść postu.
3. System przetwarza dane.
4. System aktualizuje post.
5. System wyświetla aktualny post.

**Alternatywny przebieg**:

1. W przypadku błędu podczas przetwarzania postu (brak treści postu):

System wyświetla dane o niewypełnionym polu.

**Przypadek użycia**: Dodanie komentarza do postu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.2, ten przypadek użycia opisuje możliwość dodania komentarza do postu użytkownika na jego profilu.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi wypełnić pole co najmniej jednym znakiem drukowanym.

Post musi istnieć w bazie danych.

**Przebieg**:

1. Użytkownik wypełnia pole komentarza.
2. Użytkownik klika wyślij.
3. System przetwarza dane.
4. System dodaje komentarz do postu.
5. System wyświetla komentarz.

**Alternatywny przebieg**:

1. W przypadku błędu podczas przetwarzania komentarza (brak treści komentarza):

System wyświetla dane o niewypełnionym komentarzu.

**Przypadek użycia**: Usunięcia komentarza z postu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.2, ten przypadek użycia opisuje możliwość usunięcia komentarza z postu.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być właścicielem komentarza.

Post musi istnieć w bazie danych.

**Przebieg**:

1. Użytkownik wybiera opcje usunięcia komentarza.
2. System wyświetla okno, “czy chcesz usunąć komentarz?”.
3. Użytkownik klika na przycisk “tak”.
4. System przetwarza dane.
5. System usuwa komentarz.
6. System aktualizuje wyświetlaną listę komentarzy.

**Alternatywny przebieg**:

1. Użytkownik klika na przycisk “nie” - ponownie wyświetla się lista postów.

**Przypadek użycia**: Edycja komentarza postu

**Opis** Zgodnie z tym, co widzimy na rysunku 3.2, ten przypadek użycia opisuje możliwość edycji komentarza na stronie użytkownika

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być właścicielem komentarza.

Użytkownik musi wypełnić pole co najmniej jednym znakiem drukowanym.

**Przebieg**:

1. Użytkownik klika edytuj komentarz.
2. Użytkownik edytuje treść komentarza.
3. System przetwarza dane.
4. System aktualizuje komentarz.
5. System wyświetla aktualny komentarz.

**Alternatywny przebieg**:

1. W przypadku błędu podczas przetwarzania komentarza (brak treści komentarza):

System wyświetla dane o niewypełnionym polu.

**Przypadek użycia**: Wyświetlenie komentarzy pod postem

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.2, ten przypadek użycia opisuje wyświetlenie komentarzy.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

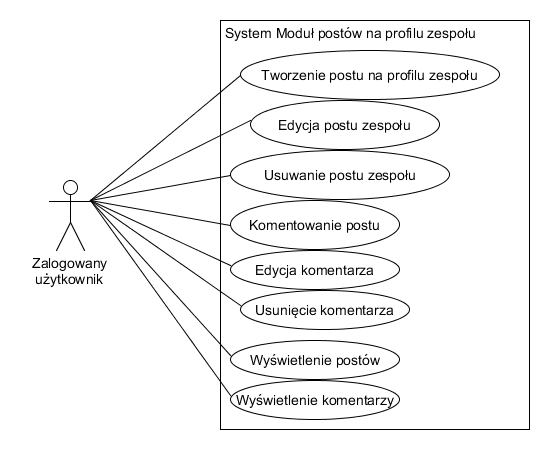
Post musi istnieć i nie być usuniętym.

**Przebieg**:

1. System pobiera post.
2. System pobiera komentarze do postu.
3. System wyświetla post z komentarzami.

**Alternatywny przebieg:**

a) W przypadku braku komentarzy system wyświetla informacje o braku komentarzy.



Rysunek 3.3 Moduł postów na profilu zespołu

**Przypadek użycia**: Dodanie postu na profilu zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.3, ten przypadek użycia opisuje możliwość dodania postu na profilu zespołu. Wówczas twórcą postu jest zespół.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być w składzie zespołu.

Użytkownik musi się znajdywać na profilu zespołu.

Użytkownik musi wypełnić treścią post co najmniej jednym znakiem drukowanym.

**Przebieg**:

1. Użytkownik wypełnia pole postu.
2. System przetwarza dane.
3. System dodaje post.
4. System wyświetla wśród postów dodany post, jako post dodany przez zespół.

**Alternatywny przebieg**:

a) W przypadku błędu podczas przetwarzania postu (brak treści postu):

System wyświetla dane o niewypełnionym poście

**Przypadek użycia**: Usunięcia postu na profilu zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.3 ten przypadek użycia opisuje możliwość usunięcia postu z profilu zespołu

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być w składzie zespołu.

Użytkownik musi się znajdywać na profilu zespołu.

**Przebieg**:

1. Użytkownik wybiera opcje usunięcia postu.
2. System wyświetla komunikat “czy na pewno chcesz usunąć post?”.
3. Użytkownik klika “tak”.
4. System przetwarza dane.
5. System usuwa post.
6. System aktualizuje wyświetlaną listę postów.

**Przypadek użycia**: Edycja postu na profilu zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.3, ten przypadek użycia opisuje możliwość edycji postu na profilu zespołu.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być w składzie zespołu.

Użytkownik musi wypełnić treścią post, co najmniej jednym znakiem drukowanym.

Użytkownik musi się znajdywać na profilu zespołu.

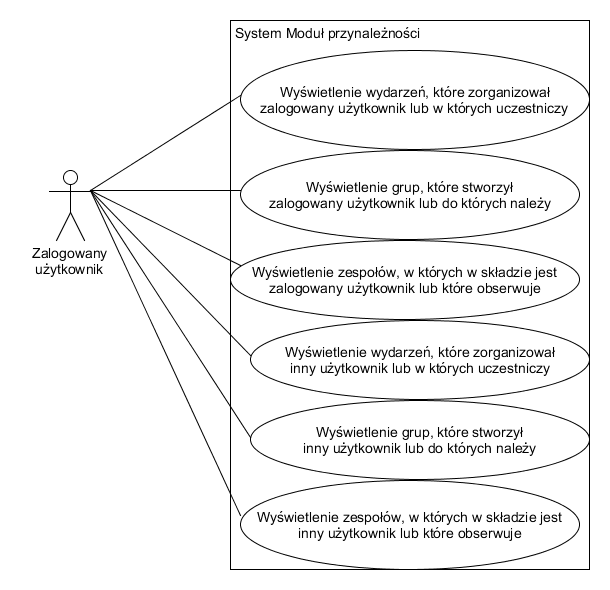
**Przebieg**:

1. Użytkownik klika edytuj post.
2. Użytkownik edytuje treść postu.
3. System przetwarza dane.
4. System aktualizuje post.
5. System wyświetla aktualny post.

**Alternatywny przebieg**:

a) W przypadku błędu podczas przetwarzania postu (brak treści postu):

System wyświetla dane o niewypełnionym poście.



Rysunek 3.4 Moduł przynależności

**Przypadek użycia**: Wyświetlenie wydarzeń, w których użytkownik (właściciel profilu) uczestniczy

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.4, ten przypadek użycia opisuje możliwość wyświetlenie wydarzeń w których uczestniczy użytkownik właściciel profilu.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik znajduje się na swoim profilu.

**Przebieg**:

1. Użytkownik wybiera zakładkę wydarzenia na swoim profilu.
2. System pobiera wydarzenia.
3. System wyświetla wydarzenia w których uczestniczy w kolejności wg. daty z informacją czy użytkownik jest organizatorem wydarzenia czy bierze w nim tylko udział.

**Alternatywny przebieg:**

a) W przypadku nieuczestniczenia w wydarzeniu, system wyświetla informacje “nie bierzesz udziału w żadnym wydarzeniu”

**Przypadek użycia**: Wyświetlenie grup do których należy użytkownik (właściciel profilu).

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.4, ten przypadek użycia opisuje możliwość wyświetlenia grup do których należy użytkownik.

**Warunki wstępne:**

Użytkownik musi być zalogowany.

Użytkownik znajduje się na swoim profilu.

**Przebieg**:

1. Użytkownik wybiera zakładkę grupy na swoim profilu
2. System wyświetla grupy w których uczestniczy

**Alternatywny przebieg**:

a) W przypadku braku uczestnictwa w grupach, system wyświetla informacje “nie jesteś członkiem żadnej grupy”

**Przypadek użycia**: Wyświetlenie zespołów do których należy użytkownik (właściciel profilu).

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.4, ten przypadek użycia opisuje możliwość wyświetlenia zespołów, które użytkownik obserwuje lub znajduje się w składzie zespołu

**Warunki wstępne:**

Użytkownik musi być zalogowany.

Użytkownik znajduje się na swoim profilu.

**Przebieg**:

1. Użytkownik wybiera zakładkę zespoły na swoim profilu.
2. System wyświetla zespoły w których uczestniczy lub obserwuje.

**Alternatywny przebieg**:

a) W przypadku obserwacji jakiegoś zespołu lub w jego składzie, system wyświetla informacje “nie obserwujesz żadnego zespołu i w żadnym nie uczestniczysz”.

**Przypadek użycia**: Wyświetlenie wydarzeń użytkownika

Opis: Zgodnie z tym, co widzimy na rysunku 3.4, ten przypadek użycia opisuje możliwość wyświetlenia wydarzeń w których uczestniczy użytkownik

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi istnieć.

Użytkownik znajduje się nie na swoim profilu.

**Przebieg**:

1. Użytkownik wybiera zakładkę wydarzenia na profilu użytkownika.
2. System wyświetla wydarzenia, w których uczestniczy użytkownik wg. daty wydarzenia.

**Alternatywny przebieg**:

a) W przypadku nieuczestniczenia w wydarzeniu, system wyświetla informacje “użytkownik nie bierze udziału w żadnym wydarzeniu”.

**Przypadek użycia**: Wyświetlenie grup użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.4, ten przypadek użycia opisuje możliwość wyświetlenia grup, które dany użytkownik uczestniczy.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi istnieć.

Użytkownik znajduje się nie na swoim profilu.

**Przebieg**:

1. Użytkownik wybiera zakładkę grupy.
2. System wyświetla grupy w których uczestniczy dany użytkownik.

**Alternatywny przebieg**:

a) W przypadku braku uczestnictwa w grupie, system wyświetla informacje “użytkownik nie uczestniczy w żadnej grupie”.

**Przypadek użycia**: Wyświetlenie zespołów użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.4, ten przypadek użycia opisuje możliwość wyświetlenia zespołów które obserwuje lub w których jest w składzie dany użytkownik.

**Warunki wstępne:**

Użytkownik musi być zalogowany.

Użytkownik musi istnieć.

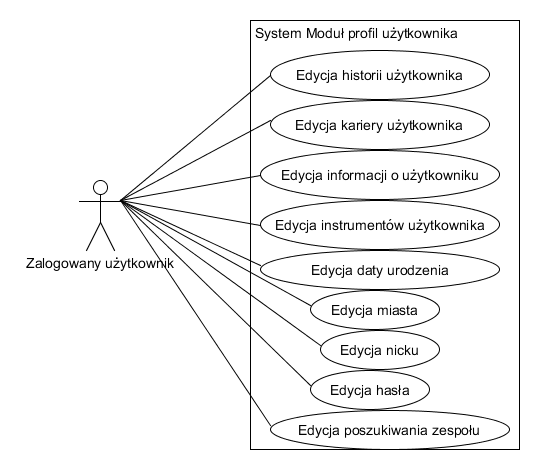
Użytkownik znajduje się nie na swoim profilu.

**Przebieg**:

1. Użytkownik wybiera zakładkę zespoły.
2. System wyświetla zespoły które obserwuje lub uczestniczy dany użytkownik.

**Alternatywny przebieg**:

a) W przypadku braku uczestnictwa w zespole i braku obserwacji, system wyświetla informacje “użytkownik nie obserwuje ani nie uczestniczy w żadnym zespole”



Rysunek 3.5 Moduł profil użytkownika

**Przypadek użycia:** Edycja własnej historia użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.5, ten przypadek użycia opisuje możliwość dodania własnej historii użytkownika.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi edytować historię na swoim profilu.

**Przebieg**:

1. Użytkownik klika przycisk edycji historii.
2. System wyświetla mu pole z obecną wypełnioną historią.
3. Użytkownik edytuje pole.
4. Użytkownik klika zapisz.
5. System przetwarza dane.
6. System zapisuje dane.
7. System wyświetla aktualną historię.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.

c) Jeśli pole jest puste, po zapisaniu danych system wyświetla informacje “Użytkownik nie wypełnił swojej historii”.

**Przypadek użycia:** Edycja pola o sobie

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.5, ten przypadek użycia opisuje możliwość opisania samego siebie przez użytkownika.

**Warunki wstępne:**

Użytkownik musi być zalogowany.

Użytkownik musi edytować swój opis własnej osoby.

**Przebieg**:

1. Użytkownik klika przycisk edycji opisu o sobie.
2. System wyświetla mu pole z obecnym opisem o sobie.
3. Użytkownik edytuje pole.
4. Użytkownik klika zapisz.
5. System przetwarza dane.
6. System zapisuje dane.
7. System wyświetla aktualny opis o sobie.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.

1. Jeśli pole jest puste, po zapisaniu danych system wyświetla informacje “Użytkownik nie wypełnił informacji o sobie”.

**Przypadek użycia:** Edycjawłasnej kariera muzyczna

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.5, ten przypadek użycia opisuje możliwość dodania własnej kariery muzycznej użytkownika.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi edytować własną kariery muzycznej użytkownika.

**Przebieg**:

1. Użytkownik klika przycisk edycji opis kariery muzycznej.
2. System wyświetla mu pole z obecnym opisem kariery muzycznej.
3. Użytkownik edytuje pole.
4. Użytkownik klika zapisz.
5. System przetwarza dane.
6. System zapisuje dane.
7. System wyświetla aktualny opis kariery muzycznej.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.

c) Jeśli pole jest puste, po zapisaniu danych system wyświetla informacje “Użytkownik nie wypełnił swojej historii kariery”.

**Przypadek użycia:** Edycja instrumentów użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.5, ten przypadek użycia opisuje możliwość wyboru i edycji instrumentów na których gra użytkownik.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi edytować dane na swoim profilu.

**Przebieg**:

1. Użytkownik klika przycisk edycji listy instrumentów muzycznych.
2. Użytkownik zaznacza/odznacza pozycję lub filtruje z listy instrumenty muzyczne.
3. Użytkownik klika zapisz.
4. System przetwarza dane.
5. System zapisuje dane.
6. System wyświetla aktualną listę instrumentów muzycznych.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.

c) Jeśli użytkownik nic ni wypełnił, po zapisaniu danych system wyświetla informacje “Użytkownik nie wypełnił informacji na czym gra”.

**Przypadek użycia:** Edycja pola szukam zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.5, ten przypadek użycia opisuje możliwość wyboru przez użytkownika czy szuka zespołu. Wówczas użytkownik staje się widoczny dla zespołów jako kandydat poszukujący zespołu.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi edytować dane na swoim profilu.

**Przebieg**:

1. Użytkownik klika przycisk edycji listy instrumentów muzycznych.
2. Użytkownik zaznacza/odznacza pozycję czy szuka zespołu.
3. Użytkownik klika zapisz.
4. System przetwarza dane.
5. System zapisuje dane.
6. System wyświetla aktualny status czy użytkownik szuka zespołu.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.

**Przypadek użycia:** Edycja daty urodzin

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.7, ten przypadek użycia opisuje możliwość wyboru przez użytkownika swojej daty urodzin.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi edytować dane na swoim profilu.

**Przebieg**:

1. Użytkownik klika przycisk edycji daty urodzin.
2. Użytkownik wybiera datę.
3. Użytkownik klika zapisz.
4. System przetwarza dane.
5. System zapisuje dane.
6. System wyświetla aktualną datę urodzin.

**Alternatywny przebieg:**

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.

c) Jeśli pole jest puste, po zapisaniu danych system wyświetla informacje “Użytkownik nie wypełnił daty swojej urodzin”.

**Przypadek użycia:** Edycja miasta użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.5, ten przypadek użycia opisuje możliwość wyboru przez użytkownika miasta, w którym aktualnie mieszka.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi edytować dane na swoim profilu.

**Przebieg**:

1. Użytkownik klika przycisk edycji.
2. Użytkownik wybiera miasto z listy (opcja filtra dla listy miast).
3. Użytkownik klika zapisz.
4. System przetwarza dane.
5. System zapisuje dane.
6. System wyświetla aktualne miasto w którym mieszka użytkownik.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.

c) Jeśli pole jest puste, po zapisaniu danych system wyświetla informacje “Użytkownik nie wybrał miasta z którego jest”.

**Przypadek użycia:** Edycja nicku użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.5, ten przypadek użycia opisuje możliwość edycji nicka użytkownika.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi edytować dane na swoim profilu.

Użytkownik musi wybrać unikalny nick.

**Przebieg**:

1. Wypełnia pole nick.
2. Użytkownik klika zapisz.
3. System przetwarza dane.
4. System aktualizuje nick.
5. System wyświetla informacje o poprawnej zmianie nicku.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.

c) Jeżeli nick nie był unikalny system wyświetla informacje: ‘Nick nie jest unikalny”.

**Przypadek użycia:** Edycja hasła użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.5, ten przypadek użycia opisuje możliwość edycji hasła użytkownika.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi edytować dane na swoim profilu.

Hasło musi mieć co najmniej 8 znaków alfanumerycznych.

**Przebieg**:

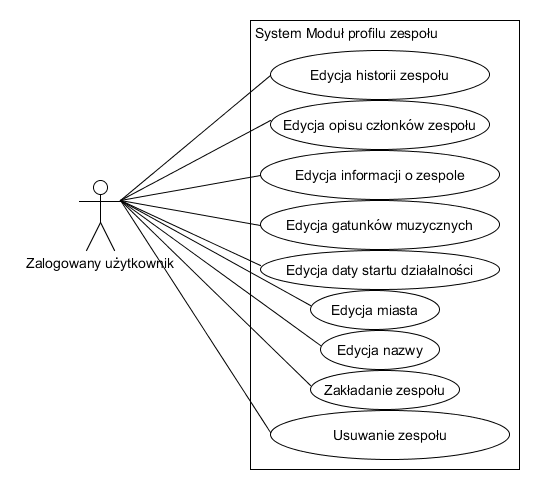
1. Użytkownik wypełnia formularz.
2. Użytkownik klika zapisz.
3. System przetwarza dane.
4. System zapisuje nowe hasło.
5. System wyświetla informacje o poprawnej zmianie hasła.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.



Rysunek 3.6 Moduł profilu zespołu

**Przypadek użycia:** Edycja nazwę zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.6, ten przypadek użycia opisuje możliwość edycji nazwy zespołu. Nazwa zespołu musi mieć co najmniej jeden znak.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być w składzie zespołu.

Użytkownik musi się znajdywać na profilu zespołu.

Zespół musi istnieć w bazie danych.

**Przebieg**:

1. Wypełnia pole nazwa zespołu.
2. Użytkownik klika zapisz.
3. System przetwarza dane.
4. System aktualizuje nazwę zespołu.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.

**Przypadek użycia:** Edycja historia zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.6, ten przypadek użycia opisuje możliwość edycji historii zespołu.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być w składzie zespołu.

Użytkownik musi się znajdywać na profilu zespołu.

**Przebieg**:

1. Użytkownik klika przycisk edycji historii.
2. System wyświetla mu pole z obecną wypełnioną historią.
3. Użytkownik edytuje pole.
4. Użytkownik klika zapisz.
5. System przetwarza dane.
6. System zapisuje dane.
7. System wyświetla aktualną historię.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.

**Przypadek użycia:** Edycja pola opis zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.6 ten przypadek użycia opisuje możliwość opisania zespołu przez użytkownika.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi się znajdywać na profilu zespołu.

Użytkownik musi być w składzie zespołu.

**Przebieg**:

1. Użytkownik klika przycisk edycji opisu zespołu.
2. System wyświetla mu pole z obecnym opisem zespołu.
3. Użytkownik edytuje pole.
4. Użytkownik klika zapisz.
5. System przetwarza dane.
6. System zapisuje dane.
7. System wyświetla aktualny opis zespołu.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.

**Przypadek użycia:** Edycja pola członkowie zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.6, ten przypadek użycia opisuje możliwość opisania ludzi wchodzących w skład zespołu.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi się znajdywać na profilu zespołu.

Użytkownik musi być w składzie zespołu.

**Przebieg**:

1. Użytkownik klika przycisk edycji opisu składu zespołu.
2. System wyświetla mu pole z obecnym opisem składu zespołu.
3. Użytkownik edytuje pole.
4. Użytkownik klika zapisz.
5. System przetwarza dane.
6. System zapisuje dane.
7. System wyświetla aktualny opis członków zespołu.

**Alternatywny przebieg:**

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.

**Przypadek użycia:** Edycja gatunków muzycznych, które gra zespół

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.6, ten przypadek użycia opisuje możliwość wyboru przez użytkownika i edycji gatunków muzycznych, które gra zespół.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być w składzie zespołu.

Użytkownik musi się znajdywać na profilu zespołu.

**Przebieg**:

1. Użytkownik klika przycisk edycji gatunków muzycznych, które gra zespół.
2. Użytkownik zaznacza/odznacza pozycję lub filtruje z listy gatunki muzyczne, które gra zespół.
3. Użytkownik klika zapisz.
4. System przetwarza dane.
5. System zapisuje dane.
6. System wyświetla aktualną listę gatunków muzycznych, które gra zespół.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.

**Przypadek użycia:** Edycja daty startu działalności zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.6, ten przypadek użycia opisuje możliwość edycji daty startu działalności zespołu.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być w składzie zespołu.

Użytkownik musi się znajdywać na profilu zespołu.

**Przebieg**:

1. Użytkownik klika przycisk edycji daty startu działalności zespołu.
2. Użytkownik wybiera datę.
3. Użytkownik klika zapisz.
4. System przetwarza dane.
5. System zapisuje dane.
6. System wyświetla aktualną datę startu działalności zespołu.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji

**Przypadek użycia:** Edycja miasta, z którego jest zespół

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.6, ten przypadek użycia opisuje możliwość wyboru przez użytkownika miasta, z którego jest zespół.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi się znajdywać na profilu zespołu.

Użytkownik musi być w składzie zespołu.

**Przebieg**:

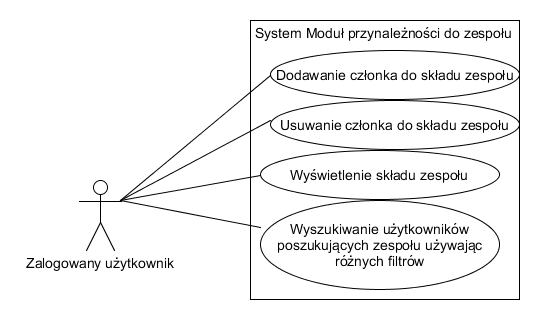
1. Użytkownik klika przycisk edycji miasta.
2. Użytkownik wybiera miasto z listy (opcja filtra dla listy miast).
3. Użytkownik klika zapisz.
4. System przetwarza dane.
5. System zapisuje dane.
6. System wyświetla aktualne miasto z którego jest zespół.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj.

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji.



Rysunek 3.7 Moduł przynależności do zespołu

**Przypadek użycia**: Dodanie użytkownika do zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.7, ten przypadek użycia opisuje możliwość dodania użytkownika do składu zespołu.

**Warunki wstępne:**

Użytkownik musi być zalogowany.

Użytkownik musi być w składzie zespołu.

Użytkownik, który ma być dodany musi istnieć.

Użytkownik musi się znajdywać na profilu zespołu.

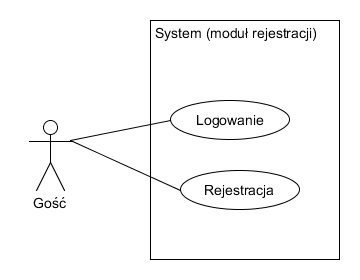
Zespół musi istnieć w bazie danych.

**Przebieg**:

1. Użytkownik wypełnia pole nicku.
2. System wyświetla listę użytkowników o podobnym nicku.
3. Użytkownik klika przycisk dodaj przy interesującym go użytkowniku.
4. System przetwarza dane.
5. System dodaje użytkownika do zespołu.

**Alternatywny przebieg**;

1. Użytkownik może zrezygnować z dodania użytkownika.



Rysunek 3.8 Moduł rejestracji

**Przypadek użycia**: Rejestracja

**Opis**: Zgodnie z tym, co widzimy na rysunek 3.8, ten przypadek użycia opisuje możliwość rejestracji niezarejestrowanego użytkownika.

**Warunki wstępne**:

Email i nick muszą być unikalne.

Hasło musi być złożone z co najmniej 8 znaków alfanumerycznych.

**Przebieg**:

1. Użytkownik wypełnia wszystkie pola.
2. System weryfikuje dane wraz wypełnianiem pól.
3. Użytkownik klika rejestruj.
4. System przetwarza dane.
5. System tworzy użytkownika.
6. System wyświetla informacje o pomyślnej operacji.

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z rejestracji.

b) użytkownik źle wypełnia dane:

1. System zwraca informacje o niepoprawnych danych, blokuje przycisk wysłania.

c) System natrafił na problem podczas przetwarzania np. dane mimo wszystko zostały źle wypełnione:

1. System zwraca odpowiednią informacje.

**Przypadek użycia**: Logowanie

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.8, ten przypadek użycia opisuje możliwość zalogowania przez użytkownika.

**Warunki wstępne**:

Email musi być w bazie danych by było możliwe logowanie.

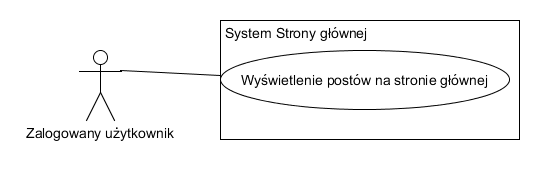
**Przebieg**:

1. Użytkownik wypełnia wszystkie pola formularza logowania.
2. System przetwarza dane.
3. System wysyła informacje o poprawnym logowaniu.
4. System wyświetla stronę główną.

**Alternatywny przebieg**:

a) Użytkownik wypełnia formularz niepoprawnymi lub nieistniejącymi danymi:

System wyświetla informacje o nieistniejącym koncie lub niepoprawnym haśle i loginie



Rysunek 3.9 Moduł strony głównej

**Przypadek użycia**: Wyświetlenie wpisów na stronie głównej

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.9. Ten przypadek użycia opisuje wyświetlenie postów na stronie głównej dla użytkownika.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi mieć co najmniej jednego znajomego, który dodał post lub zespół, który dodał post lub grupę w której napisano co najmniej jeden post.

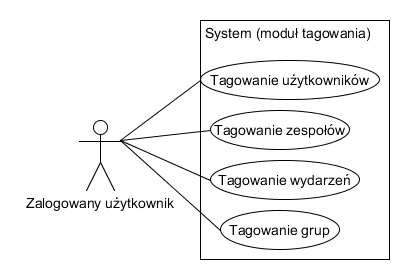
**Przebieg**:

1. System na podstawie tożsamości użytkownika wybiera najnowsze posty, do których użytkownik ma uprawnienia do obejrzenia.
2. System wyświetla posty.

**Alternatywny przebieg**:

a) W przypadku braku postów do wyświetlenia :

System wyświetla dane o braku postów do wyświetlenia, zaproszenie do poszukania jakiegoś zespołu , grupy itp.



Rysunek 3.10 Moduł tagowania

**Przypadek użycia**: Tagowanie ludzi

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.10, ten przypadek użycia opisuje możliwość tagowania użytkowników.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Pole musi mieć możliwość tagowania.

Użytkownik do otagowania musi istnieć.

**Przebieg**:

1. Użytkownik wpisuje znak “@”.
2. System wyświetla pole z możliwymi użytkownikami do otagowania.
3. Użytkownik specyfikuje nick użytkownika.
4. System filtruje listę.
5. Użytkownik wybiera interesującego go użytkownika do tagowania.
6. System przekształca tag w link do profilu użytkownika.

**Alternatywny przebieg**:

a) W przypadku chęci usunięcia tagu, użytkownik musi zaznaczyć cały nick.

b) użytkownik może po wybraniu tagowania zmodyfikować wyświetlaną nazwę ale musi się składać z co najmniej jednego znaku.

c) jeśli wprowadzi on kropkę, dwie spacje, przecinek lub inne znaki niebędące znakami alfanumerycznymi wówczas tagowanie jest możliwe jedynie przed tym znakiem.

**Przypadek użycia**: Tagowanie zespołów

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.10, ten przypadek użycia opisuje możliwość tagowania zespołów.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Pole musi mieć możliwość tagowania.

Zespół do otagowania musi istnieć.

**Przebieg**:

1. Użytkownik wpisuje znak “@”.
2. System wyświetla pole umożliwiające tagowanie.
3. Użytkownik wybiera zakładkę zespoły.
4. Użytkownik specyfikuje nazwę zespołu.
5. System filtruje listę.
6. Użytkownik wybiera interesującego go zespołu do otagowania.
7. System przekształca tag w link do profilu zespołu.

**Alternatywny przebieg**:

a) W przypadku chęci usunięcia tagu, użytkownik musi zaznaczyć całą nazwę.

b) użytkownik może po wybraniu tagowania zmodyfikować wyświetlaną nazwę ale musi się składać z co najmniej jednego znaku.

c) jeśli wprowadzi on kropkę, dwie spacje, przecinek lub inne znaki niebędące znakami alfanumerycznymi wówczas tagowanie jest możliwe jedynie przed tym znakiem.

**Przypadek użycia**: Tagowanie wydarzeń

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.10, ten przypadek użycia opisuje możliwość tagowania wydarzeń

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Pole musi mieć możliwość tagowania

Wydarzenie do otagowania musi istnieć.

**Przebieg**:

1. Użytkownik wpisuje znak “@”.
2. System wyświetla pole umożliwiające tagowanie.
3. Użytkownik wybiera zakładkę wydarzeń.
4. Użytkownik specyfikuje nazwę wydarzeń.
5. System filtruje listę.
6. Użytkownik wybiera interesującego go wydarzenia do otagowania.
7. System przekształca tag w link do profilu wydarzenia.

**Alternatywny przebieg:**

a) W przypadku chęci usunięcia tagu, użytkownik musi zaznaczyć całą nazwę.

b) użytkownik może po wybraniu tagowania zmodyfikować wyświetlaną nazwę, ale musi się składać z co najmniej jednego znaku.

c) jeśli wprowadzi on kropkę, dwie spacje, przecinek lub inne znaki niebędące znakami alfanumerycznymi wówczas tagowanie jest możliwe jedynie przed tym znakiem.

**Przypadek użycia**: Tagowanie grup

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.10, ten przypadek użycia opisuje możliwość tagowania grup.

**Warunki wstępne**:

Użytkownik musi być zalogowany. .

Pole musi mieć możliwość tagowania.

Grupa do otagowania musi istnieć.

**Przebieg**:

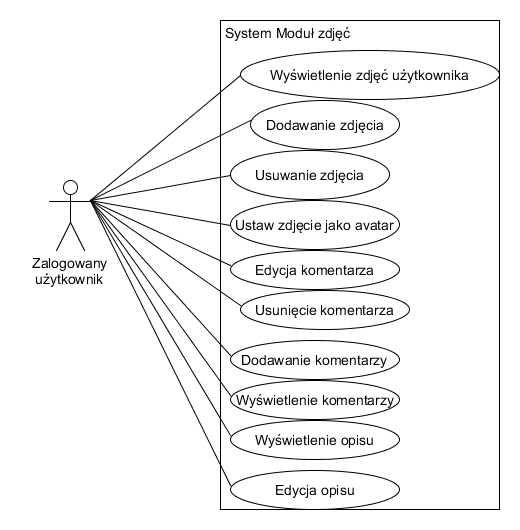
1. Użytkownik wpisuje znak “@”.
2. System wyświetla pole umożliwiające tagowanie.
3. Użytkownik wybiera zakładkę grupy.
4. Użytkownik specyfikuje nazwę grupy.
5. System filtruje listę.
6. Użytkownik wybiera interesującego go grupy do otagowania.
7. System przekształca tag w link do profilu grupy.

**Alternatywny przebieg**:

a) W przypadku chęci usunięcia tagu, użytkownik musi zaznaczyć całą nazwę.

b) użytkownik może po wybraniu tagowania zmodyfikować wyświetlaną nazwę ale musi się składać z co najmniej jednego znaku

c) jeśli wprowadzi on kropkę, dwie spacje, przecinek lub inne znaki niebędące znakami alfanumerycznymi wówczas tagowanie jest możliwe jedynie przed tym znakiem.



Rysunek 3.11 Moduł zdjęć

**Przypadek użycia:** Wyświetlenie dodanych zdjęć do profilu przez użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.11, ten przypadek użycia opisuje możliwość wyświetlania zdjęć dodanych do profilu użytkownika.

**Warunki wstępne:**

Użytkownik musi być zalogowany.

Użytkownik musi istnieć,

**Przebieg**:

1. Użytkownik wchodzi na profil innego użytkownika.
2. System pobiera dane użytkownika.
3. Użytkownik klika na zakładkę zdjęcia.
4. System pobiera zdjęcia użytkownika
5. System wyświetla dodane zdjęcia

**Alternatywny przebieg**:

Jeśli właściciel profilu nie ma zdjęć wówczas system wyświetla informacje w widoku zakładki zdjęcia “Użytkownik nie ma zdjęć”.

**Przypadek użycia:** Dodawanie zdjęć do profilu przez użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.11, przypadek użycia opisuje możliwość dodawania zdjęć do profilu użytkownika

**Warunki wstępne:**

Użytkownik musi być zalogowany.

Użytkownik musi być właścicielem użytkownika.

Zdjęcia muszą mieć odpowiedni format.

**Przebieg**:

1. Użytkownik wybiera zakładkę zdjęcia na swoim profilu.
2. Użytkownik kilka dodaj zdjęcia.
3. Użytkownik wybiera zdjęcia do dodania.
4. System wstępnie tworzy kolejkę do dodania.
5. Użytkownik ma możliwość dodania opisu do zdjęcia.
6. Użytkownik klika wyślij zdjęcia.
7. System przetwarza zdjęcia.
8. System zapisuje zdjęcia.
9. System wyświetla dodane zdjęcia.

**Alternatywny przebieg**:

1. Użytkownik może zrezygnować z dodawania zdjęcia.
2. Użytkownik nie musi dodawać opisu do zdjęcia.
3. Użytkownik może zrezygnować z dodawania zdjęć.
4. Jeśli użytkownik wybrał niepoprawny format zdjęcia, wówczas wyświetla mu się informacja o błędzie.

**Przypadek użycia:** Usunięcie zdjęcia przez użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.11, przypadek użycia opisuje możliwość usunięcia zdjęcia z profilu użytkownika.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być właścicielem zdjęcia.

**Przebieg**:

1. Użytkownik wybiera zdjęcia do usunięcia.
2. System przekierowuje użytkownika do wybranego zdjęcia.
3. Użytkownik klika usuń zdjęcie.
4. System wyświetla informacje “czy chcesz usunąć zdjęcie”.
5. Użytkownik klika przycisk “tak”.
6. System przetwarza żądanie.
7. System usuwa zdjęcie, jeśli zdjęcie to jest profilowe, wówczas jest używane zdjęcie domyślne.
8. System wyświetla informacje o pomyślnym usunięciu.

**Alternatywny przebieg:**

1. użytkownik może zrezygnować z usunięcia zdjęcia aż do kroku 5.

**Przypadek użycia:** Ustawienie zdjęcia jako awatar przez użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.11, przypadek użycia opisuje możliwość ustawienia zdjęcia jako awatar użytkownika.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi być właścicielem zdjęcia.

Użytkownik musi być właścicielem profilu.

**Przebieg**:

1. Użytkownik wchodzi na swój profil.
2. System pobiera dane użytkownika.
3. Użytkownik klika na zakładkę zdjęcia.
4. System pobiera zdjęcia użytkownika i je wyświetla.
5. Użytkownik klika na wybrane zdjęcie.
6. System przekierowuje go na widok zdjęcia.
7. Użytkownik klika ustaw przycisk jako awatar.
8. System przetwarza żądanie.
9. System ustawia zdjęcie jako awatar.
10. System wyświetla informacje o pomyślnym zmienieniu zdjęcia.

**Alternatywny przebieg**:

1. użytkownik może zrezygnować z ustawienia zdjęcia jako awatar.

**Przypadek użycia:** Dodanie komentarza do zdjęcia przez użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.11, przypadek użycia opisuje możliwość dodawania komentarza do zdjęcia.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik jest na widoku zdjęcia.

Użytkownik musi wypełnić pole co najmniej jednym znakiem drukowanym.

Zdjęcie musi istnieć w bazie danych.

**Przebieg**:

1. Użytkownik wypełnia pole komentarza pod zdjęciem.
2. Użytkownik klika wyślij.
3. System przetwarza dane.
4. System dodaje komentarz do zdjęcia.
5. System wyświetla komentarz.

**Alternatywny przebieg:**

a) W przypadku błędu podczas przetwarzania komentarza (brak treści komentarza):

System wyświetla dane o niewypełnionym komentarzu.

**Przypadek użycia:** Edycja opisu zdjęcia użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.17, przypadek użycia opisuje możliwość edycji opisu zdjęcia.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi mieć uprawnienia do edycji opisu zdjęcia jest właścicielem zdjęcia.

Użytkownik jest na widoku zdjęcia.

Zdjęcie musi istnieć w bazie danych.

**Przebieg**:

1. Użytkownik klika przycisk edycji opisu
2. System wyświetla mu pole z obecnym opisem
3. Użytkownik edytuje pole
4. Użytkownik klika zapisz
5. System przetwarza dane
6. System zapisuje dane
7. System wyświetla aktualny opis

**Alternatywny przebieg**:

a) Użytkownik rezygnuje z edycji i klika anuluj

b) System natrafia na błąd podczas przetwarzania:

System wyświetla informacje o błędzie i prosi o powtórzenie operacji

**Przypadek użycia**: Wyświetlenie komentarzy pod zdjęciem

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.11, przypadek użycia opisuje wyświetlenie komentarzy.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Zdjęcie musi istnieć w bazie danych.

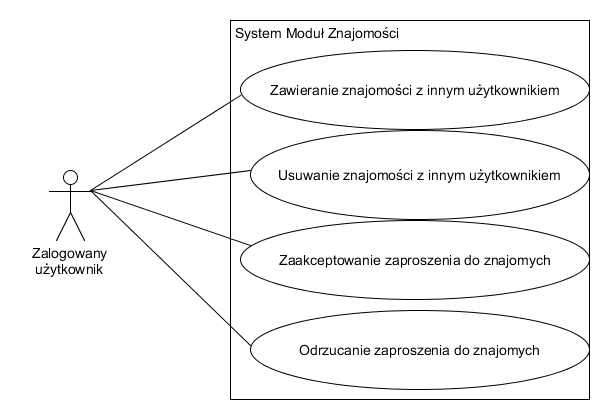
Użytkownik musi znajdywać się w widoku zdjęcia.

**Przebieg**:

1. System pobiera zdjęcie i komentarze.
2. System wyświetla użytkownikowi zdjęcie i komentarze.

**Alternatywny przebieg:**

a) W przypadku braku komentarzy system wyświetla informacje o braku komentarzy.



Rysunek 3.12 Moduł znajomości

**Przypadek użycia**: Dodanie do znajomych

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.12, ten przypadek użycia opisuje możliwość dodania użytkownika do znajomych

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik nie może mieć danego użytkownika do znajomych.

Użytkownik jest na widoku profilu danego użytkownika.

**Przebieg**:

1. Użytkownik klika dodaj do znajomych.
2. System przetwarza żądanie i dodaje zaproszenie do użytkownika.

**Alternatywny przebieg**:

a) Jeśli użytkownik miał wcześniej zaproszenie od danego użytkownika, którego dodał do znajomych. Wówczas stają się oni znajomymi.

**Przypadek użycia**: Usunięcie ze znajomych

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.12, ten przypadek użycia opisuje możliwość usunięcia użytkownika ze znajomych.

**Warunki wstępne:**

Użytkownik musi być zalogowany.

Użytkownik musi mieć danego użytkownika w znajomych.

Użytkownik musi się znajdować na liście swoich znajomych lub profilu danego użytkownika.

**Przebieg**:

1. Użytkownik klika przycisk usunięcia ze znajomych.
2. System przetwarza żądanie i ustawia znajomość jako usuniętą.

**Przypadek użycia**: Akceptacja zaproszenia do znajomych

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.12, ten przypadek użycia opisuje możliwość akceptacji zaproszenia do znajomych.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Musi istnieć zaproszenie.

**Przebieg**:

1. Użytkownik klika zaakceptuj zaproszenie.
2. System przetwarza żądanie i akceptuje znajomość z danym użytkownikiem.

**Alternatywny przebieg**;

1. użytkownik może zrezygnować z akceptacji.

**Przypadek użycia**: Odrzucenie zaproszenia do znajomych

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.12, ten przypadek użycia opisuje możliwość odrzucenia zaproszenia do znajomych.

**Warunki wstępne:**

Użytkownik musi być zalogowany.

Musi istnieć zaproszenie.

**Przebieg**:

1. Użytkownik klika zaakceptuj zaproszenie.
2. System przetwarza żądanie i odrzuca znajomość.

**Alternatywny przebieg**;

1. użytkownik może zrezygnować z odrzucenia zaproszenia.

**Przypadek użycia**: Założenie zespołu

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.12, ten przypadek użycia opisuje możliwość założenia zespołu.

**Warunki wstępne**:

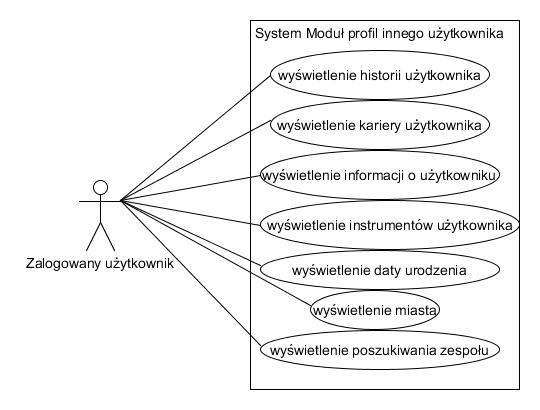
Użytkownik musi być zalogowany.

**Przebieg**:

1. Użytkownik wypełnia pole nazwy.
2. Użytkownik klika utwórz.
3. System przetwarza dane.
4. System tworzy zespół.
5. System dodaje użytkownika do składu zespołu.
6. System wyświetla profil nowo utworzonego zespołu.

**Alternatywny przebieg**;

1. Użytkownik może zrezygnować z tworzenia zespołu klikając anuluj.



Rysunek 3.13 Moduł profil innego użytkownika

**Przypadek użycia:** Wyświetlenie na czym gra inny użytkownik

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.13, ten przypadek użycia opisuje możliwość wyświetlania na czym gra użytkownik.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Instrumenty muzyczne na których gra użytkownik muszą być wybrane.

Wybrany użytkownik musi istnieć.

**Przebieg**:

1. Użytkownik wchodzi na profil innego użytkownika.
2. System pobiera dane użytkownika
3. Użytkownik klika przycisk informacje o użytkowniku.
4. System wybiera instrumenty na których gra użytkownik.
5. System wyświetla instrumenty na których gra użytkownik.

**Alternatywny przebieg**:

Jeśli właściciel profilu nie wybrał żadnych instrumentów na których gra system wyświetla informacje “Nie wybrałeś żadnych instrumentów”.

**Przypadek użycia:** Wyświetlenie historii użytkownika.

**Opis:** Zgodnie z tym, co widzimy na rysunku 3.13, ten przypadek użycia opisuje możliwość wyświetlania historii użytkownika.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Pole musi być wypełnione.

Wybrany użytkownik musi istnieć.

**Przebieg**:

1. Użytkownik wchodzi na profil innego użytkownika.
2. System pobiera dane użytkownika.
3. Użytkownik klika przycisk informacje o użytkowniku.
4. System wyświetla historię użytkownika.

**Alternatywny przebieg**:

Jeśli właściciel profilu nie wypełnił pola system wyświetla informacje ‘Użytkownik nie napisał swojej historii”.

**Przypadek użycia:** Wyświetlenie opisu użytkownika

**Opis:** Zgodnie z tym, co widzimy na rysunku 3.13 ten przypadek użycia opisuje możliwość wyświetlania opisu użytkownika

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Pole musi być wypełnione.

Użytkownik musi istnieć.

**Przebieg**:

1. Użytkownik wchodzi na profil innego użytkownika.
2. System pobiera dane użytkownika.
3. Użytkownik klika przycisk informacje o użytkowniku.
4. System wyświetla opis użytkownika.

**Alternatywny przebieg**:

Jeśli właściciel profilu nie wypełnił pola system wyświetla informacje ‘Użytkownik nie napisał swojego opisu”.

**Przypadek użycia:** Wyświetlenie miasta z którego jest użytkownik

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.13, ten przypadek użycia opisuje możliwość wyświetlania, skąd jest dany użytkownik.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Miasto musi być wybrane przez użytkownika.

Użytkownik musi istnieć.

**Przebieg**:

1. Użytkownik wchodzi na profil innego użytkownika.
2. System pobiera dane użytkownika.
3. Użytkownik klika przycisk informacje o użytkowniku.
4. System wyświetla miasto użytkownika.

**Alternatywny przebieg**:

Jeśli właściciel profilu nie wypełnił miasta system wyświetla informacje ‘Użytkownik nie wybrał miasta”.

**Przypadek użycia:** Wyświetlenie opisu kariery użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.13, ten przypadek użycia opisuje możliwość wyświetlania opisu kariery użytkownika.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Pole musi być wypełnione.

Użytkownik musi istnieć.

**Przebieg**:

1. Użytkownik wchodzi na profil innego użytkownika.
2. System pobiera dane użytkownika.
3. Użytkownik klika przycisk informacje o użytkowniku.
4. System wyświetla opis kariery użytkownika.

**Alternatywny przebieg**:

Jeśli właściciel profilu nie wypełnił pola system wyświetla informacje ‘Użytkownik nie opisał swojej kariery”.

**Przypadek użycia:** Wyświetlenie daty urodzin użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.13, ten przypadek użycia opisuje możliwość wyświetlania daty urodzin użytkownika

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Pole musi być wypełnione.

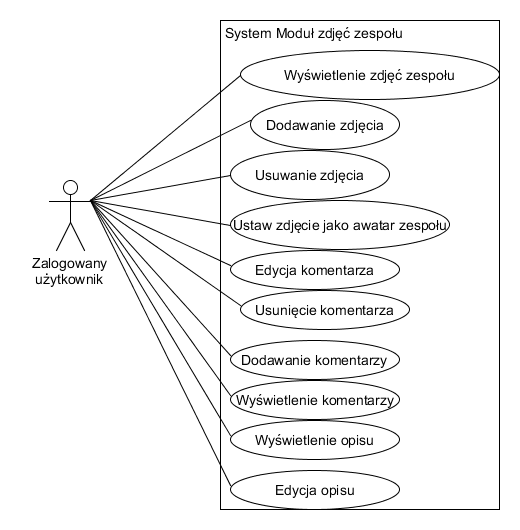
Użytkownik musi istnieć,

**Przebieg**:

1. Użytkownik wchodzi na profil innego użytkownika.
2. System pobiera dane użytkownika.
3. Użytkownik klika przycisk informacje o użytkowniku.
4. System wyświetla datę urodzin.

**Alternatywny przebieg**:

Jeśli właściciel profilu nie wypełnił pola system wyświetla informacje ‘Użytkownik nie ustawił daty urodzin”.



Rysunek 3.14 Moduł zdjęć zespołów

**Przypadek użycia:** Wyświetlenie dodanych zdjęć do profilu zespołu przez użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.14, ten przypadek użycia opisuje możliwość wyświetlania zdjęć dodanych do profilu zespołu.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Zespół musi istnieć w bazie danych.

Użytkownik musi się znajdywać na profilu zespołu.

**Przebieg**:

1. Użytkownik wybiera zakładkę zdjęcia.
2. System pobiera zdjęcia.
3. System wyświetla dodane zdjęcia.

**Alternatywny przebieg**:

Jeśli zespół nie ma zdjęć wówczas system wyświetla informacje zespół nie dodał żadnych zdjęć.

**Przypadek użycia:** Dodawanie zdjęć do profilu zespołu przez użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.14,, ten przypadek użycia opisuje możliwość dodawania zdjęć do profilu zespołu przez użytkownika.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi się znajdywać w składzie zespołu.

Użytkownik musi się znajdywać na profilu zespołu.

Zdjęcia muszą mieć odpowiedni format.

Zespół musi istnieć w bazie danych.

**Przebieg**:

1. Użytkownik wybiera zdjęcia do dodania.
2. System wstępnie tworzy kolejkę do dodania.
3. Użytkownik ma możliwość dodania opisu do zdjęcia.
4. Użytkownik klika wyślij zdjęcia.
5. System przetwarza zdjęcia.
6. System zapisuje zdjęcia.
7. System wyświetla dodane zdjęcia.

**Alternatywny przebieg:**

1. Użytkownik może zrezygnować z dodawania zdjęcia.
2. Użytkownik nie musi dodawać opisu do zdjęcia.
3. Użytkownik może zrezygnować z dodawania zdjęć.

**Przypadek użycia:** Usunięcie zdjęcia zespołu przez użytkownika

**Opis**: Zgodnie z tym, co widzimy na rysunku 3.14,, ten przypadek użycia opisuje możliwość usunięcia zdjęcia z profilu zespołu przez użytkownika.

**Warunki wstępne**:

Użytkownik musi być zalogowany.

Użytkownik musi się znajdywać w składzie zespołu.

Użytkownik musi się znajdywać na profilu zespołu.

Zespół musi istnieć w bazie danych.

Zdjęcie musi istnieć

**Przebieg**:

1. Użytkownik wybiera zdjęcia do usunięcia.
2. Użytkownik klika usuń zdjęcie.
3. Użytkownik potwierdza chęć usunięcia żądania.
4. System przetwarza żądanie.
5. System usuwa zdjęcie.
6. System wyświetla informacje o pomyślnym usunięciu.

**Alternatywny przebieg**:

1. Użytkownik może zrezygnować z usuwania zdjęcia.

Pozostałe przypadku użycia w formie diagramów możemy zobaczyć w załączniku B. Rysunki od B.1 do B.7 przedstawiają pozostaje przypadki użycia w projekcie aplikacji.

# Projekt aplikacji

## 4.1 Aplikacja serwerowa

### 4.1.1 Kontrolery w aplikacji

Do każdego kontrolera dołączona jest klasa pomocnicza zapewniająca dostarczenie odpowiednich danych. W której wykorzystuje się bibliotekę decent exposure. Jak już wcześniej wspomniano udostępnia ona możliwość dostarczenia konkretnych obiektów.

**BandsContoller**

Opis: Ten kontroler jest związany z zespołami.

Akcje:

1. query - ta akcja jest związana z zwracaniem wyniku wyszukiwania.
2. index - ta akcja związana jest z pobraniem wszystkich zespołów dla konkretnego użytkownka..
3. fallows - akcja związana z pobraniem wszystkich, którzy obserwują dany zespół.
4. create - akcja związana z tworzeniem zespołu.
5. setAvatar - akcja związana z ustawieniem awatara dla konkretnego zespołu.
6. show - akcja związana z wyświetleniem konkretnego zespołu.
7. set\_music\_generes - aktualizacja gatunków muzycznych, jakie gra dany zespół.
8. destroy - usuwanie zespołu.
9. update - aktualizacja konkretnych informacji zespole.
10. add\_member - dodanie użytkownika jako członka zespołu.
11. delete\_member - usunięcie użytkownika jako członka zespołu.
12. add\_fallower - rozpoczęcie obserwacji zespołu przez użytkownika.
13. delete\_fallower - usunięcie obserwacji zespołu przez użytkownika.

**EventsController**

Opis: Ten kontroler jest związany z wydarzeniami.

Akcje:

1. query - ta akcja jest związana z zwracaniem wyniku wyszukiwania wydarzenia.
2. index - ta akcja związana jest z pobraniem wszystkich wydarzeń dla konkretnego użytkownka/zespołu/grupy.
3. create - akcja związana z tworzeniem wydarzenia.
4. show - akcja związana z wyświetleniem konkretnego wydarzenia.
5. set\_music\_generes - aktualizacja gatunków muzycznych związane z danym wyderzaniem.
6. destroy - usuwanie zespołu.
7. update - aktualizacja konkretnych informacji wydarzeniu.
8. set\_member - ustawienie użytkownika jako organizatora wydarzenia, który może edytować wydarzenie.
9. add\_me - rozpoczęcie brania udziału w wydarzeniu przez użytkownika
10. delete\_me- usunięcie brania udziału w wydarzeniu przez użytkownika.
11. add\_band - dodanie zespołu do wydarzenia.
12. delete\_band- usunięcie zespołu z wydarzenia.
13. accept\_band - akceptacja zespołu wzięcia udziału w wydarzeniu.

**ConversationsController**

Opis: Ten kontroler jest związany z konwersacjami.

Akcje:

1. index - ta akcja związana jest z pobraniem wszystkich konwersacji dla konkretnego użytkownika.
2. create - akcja związana z konwersacji.
3. show - akcja związana z wyświetleniem konkretnego konwersacji.
4. destroy - usuwanie konwersacji.
5. update - aktualizacja konkretnych informacji konwersacji.
6. addUserToConversation- dodanie użytkownika jako członka konwersacji.
7. deleteUserFromConversation- usunięcie użytkownika jako członka konwersacji.

**MessagesController**

Opis: Ten kontroler jest związany z wiadomościami.

Akcje:

1. index - ta akcja związana jest z wszystkich wiadomości dla konkretnego użytkownika.
2. create - akcja związana z tworzeniem wiadomości.
3. show - akcja związana z wyświetleniem konkretnej wiadomości.
4. unreaded\_messages - pobranie nieprzeczytanych wiadomości z konkretnej konwersacji.

**UsersController**

Opis: Ten kontroler jest związany z użytkownikami.

Akcje:

1. query - ta akcja jest związana z zwracaniem wyniku wyszukiwania.
2. show - akcja związana z wyświetleniem konkretnego użytkownika.
3. setAvatar - akcja związana z ustawieniem awatara dla konkretnego użytkownika.
4. setAddonationalInfo - update bardziej specyficznych informacji.
5. plays - ustawienie na czym gra dany użytkownik.

Pozostałe akcje takie jak tworzenie użytkownika, zmiana jego hasła, nicku itp. Wykorzystuje się bibliotekę Devise Token Auth, które te akcje obsługuje.

**GroupsController**

Opis: Ten kontroler jest związany z grupami.

Akcje:

1. query - ta akcja jest związana z zwracaniem wyniku wyszukiwania grup.
2. index - ta akcja związana jest z pobraniem wszystkich grup dla konkretnego użytkownika.
3. group\_members- akcja związana z pobraniem wszystkich, którzy uczestniczą w danej grupie.
4. create - akcja związana z tworzeniem zespołu.
5. setAvatar - akcja związana z ustawieniem awatara dla konkretnej grupy.
6. show - akcja związana z wyświetleniem konkretnej grupy.
7. destroy - usuwanie grupy.
8. update - aktualizacja konkretnych informacji o grupie.
9. add\_me - dołączenie do grupy przez użytkownika.
10. delete\_me- odejście z grupy przez użytkownika.
11. set\_member - ustawianie użytkownika jako administratora grupy, który może edytować grupę.

**PostsController**

Opis: Ten kontroler jest związany z postami.

Akcje:

1. index -ta akcja związana jest z pobraniem wszystkich postów dla grupy/użytkownika/zespołu.
2. create - akcja związana z tworzeniem postu.
3. show - akcja związana z wyświetleniem konkretnego postu (opcja rozwojowa)
4. destroy - usuwanie postu.
5. update - aktualizacja postu.

**UtilsController**

Opis: Ten kontroler jest związany z danymi, które są wykorzystywane w różnych częściach aplikacji.

Akcje:

1. get\_music\_generes - pobranie wszystkich gatunków muzycznych.
2. get\_user\_instrument - pobranie wszystkich instrumentów.
3. get\_cities - pobranie wszystkich miast.

**BuddyshipsController**

Opis: Ten kontroler jest związany z znajomościami.

Akcje:

1. notaccepted - wyświetlenie niezaakceptowanych zaproszeń konkretnego użytkownika
2. accepted wyświetlenie niezaakceptowanych zaproszeń konkretnego użytkownik
3. create - etap tworzenia znajomości
4. accept - akceptowanie znajomości
5. destroy - usuwanie znajomości

**PhotosContoller**

Opis: Ten kontroler jest związany ze zdjęciami.

Akcje:

1. index ta akcja związana jest z pobraniem wszystkich zdjęć grupy/użytkownika/zespołu.
2. create - akcja związana z tworzeniem zdjęć.
3. show - akcja związana z wyświetleniem konkretnego zdjęcia.
4. destroy - usuwanie zdjęcia.
5. update - aktualizacja konkretnych informacji o zdjęciu.
6. download - konkretny link do miniaturki zdjęcia.
7. download\_org - link do oryginalnego zdjęcia.

**CommentsController**

Opis: Ten kontroler jest związany z komentarzami.

Akcje:

1. index - ta akcja związana jest z pobraniem wszystkich komentarzy dla postu/zdjęcia.
2. create - akcja związana z tworzeniem komentarza.
3. show - akcja związana z wyświetleniem konkretnego komentarzu (opcja rozwojowa).
4. destroy - usuwanie komentarza.
5. update - aktualizacja komentarza.

### 4.1.2. API

Trasy API zbudowane są na bazie pliku routes.rb Do komunikacji jest zbudowane na bazie modułów. Przyjmując, że mamy następującą trasę API przedstawioną na listingu 14.

Listing 15 Przykładowa trasa API

*/api/v1/posts/1*

Api - jest to moduł API, taka budowa jest często spotykaną praktyką.

v1 - oznacza pierwszą wersję api w przyszłości może być więcej wersji api.

posts - odwołujemy się do konkretnego kontrolera

1 - jest to id zasobu

W Ruby on Rails mamy możliwość zagnieżdżania zasobu w zasobie, co możemy przeczytać w dokumentacji [34]. Wówczas odwołujemy się do konkretnego kontrolera odpowiedzialnego za dany zasób. Zagnieżdżenie jednak dostarcza nam dodatkowe parametry. Takie jak np. jeśli tworzymy post korzystając z trasy np. /api/v1/bands/1/posts i wykorzystując metodę POST. Wówczas odwołujemy się do kontrolera posts. Z tą różnicą , że do tablicy parametrów dołączony zostaje nowy parametr band\_id. Przykładowe zagnieżdżenie zostało zaprezentowane na listingu 15.

Listing 16 Zagnieżdżanie zasobu w zasobie routes.rb

*# w zagnieżdżanie zasobu postów w zespole, wówczas możemy pobrać wszystkie posty*

*# konkretnego zespoły poprzez odwołanie do tray /api/v1/bands/1/posts*

*resources :bands do*

*resources :posts*

*end*

Ruby on Rails dostarcza możliwość stworzenia dodatkowych akcji [34], za pomocą których możemy odwołać się w API, jeśli użyjemy następującej konstrukcji, jak na listingu 16, uzyskamy następującą trasę przedstawioną na listingu 17.

Listing 17 Plik routes.rb i dodanie metody "members" konkretnego zespołu

*resources :bands do*

*get ‘members’*

*end*

Listing 18 Wynik stworzenia trasy

*/api/v1/bands/1/members*

Wówczas musimy odwołać się za pomocą metody get i do konkretnego id zasobu. Dzięki temu tworzymy nową akcje np. do pobrania wszystkich członków zasobu. W przypadku, gdy chcemy coś zrobić na bazie kolekcji? Trzeba użyć konstrukcji przedstawionej na listingu 18 i wówczas uzyskamy następującą trasę listingu 19.

Listing 19 Plik routes.rb i dodanie metody do kolekcji "members"

*resources :bands do*

*collection do*

*get ‘members’*

*end*

*end*

Listing 20 Wynik stworzenia trasy

*/api/v1/bands/members*

Należy dodać, że każdy wynik akcji jest przekazywany do widoku o nazwie akcji. Następnie wynik ten przekształcany jest w obiekt JSON i odsyłany do klienta, który wykonał żądanie.

## 4.2 Frontend

### 4.2.1 Organizacja aplikacji

Wstępnie pliki są podzielone, ze względu na typ:

1. Widoki częściowe HTML - katalog partials
2. Skrypty Javascript - katalog js
3. Arkusze styli - katalog css
4. Obrazki statyczne - katalog images
5. Fonty - katalog fonts

Poza bibliotekami JavaScript z których korzysta się w bibliotece, w katalogu “js” znajdują się istotne dla aplikacji moduły.

W pliku directives.js znajdują się dyrektywy JavaScript, które rozszerzają możliwości HTML, wspomagają np. możliwość tagowania, dodawanie miniaturek zdjęć przed wysłaniem na serwer itp. Plik main.js definiuje stany (widoki) wykorzystując bibliotekę “Angular UI Router”, które aplikacja zapewnia, zabezpieczania w postaci autoryzacji, konfiguracje usług oraz prostymi skryptami jQuery, które mają na celu np. zapewnić podążające menu. Cotrollers.js zawiera wszystkie kontrolery znajdujące się w aplikacji. Filter.js definiuje filtry używane w aplikacji Resources.js definiuje usługi, które służą do komunikacji z API serwera.

### 4.2.2. Resources - zasoby

Resources są to usługi, które mają za zadanie komunikowanie się z API. Opierają się one na innych usługach wbudowanych w platformę programistyczną AngularJS, a mianowicie $http. Umożliwiają one wykonywanie zapytań asynchronicznych. Wzorując się po części na dokumentacji [35] podstawowa konstrukcja z wstrzykniętą usługą $http wygląda tak, jak na listingu 21.

Listing 21 Konstrukcja serwisu z wstrzykniętą usługą $http

*app.service(“PostService”,[“$http”,function($http){*

*this.getPosts = function(data){*

*return $http({*

*method:'get',*

*url:"http://localhost:3001/api/v1/users/"+data.user\_id+"/posts",*

*params:{*

*page: data.page*

*}*

*})*

*}*

*}]);*

Usługa $http służy nam do komunikacji z API serwera za pomocą odpowiedniej trasy, metody itp. Tworzenie powyższego prostego serwisu składa się ze wstrzyknięcia zależności, wówczas możemy skorzystać z usługi $http. Następnie deklarujemy interesującą nas metodę “getPosts”. Po wywołaniu funkcji następuje zwrócenie efektu wywołania usługi $http.

Należy dodać, że wstrzyknięcie zależności zgodnie z tym ,co możemy przeczytać w [2], w uproszczeniu oznacza, że AngularJS po przeanalizowaniu argumentów podanych w konstruktorze komponentu, wyszukuje wymagane komponenty i dołącza je do elementu.

Usługę $http przyjmuje m.in innymi następujące parametry zgodnie z [36]:

* method - ten parametr określa jaką metodę żądania wykonamy do aplikacji serwera, są zdefiniowane następujące metody: get, post, put, delete, head, patch.
* url - w ten sposób definiujemy url pod który będzie kierowane żądanie
* params - w przypadku metody wykorzystującą query string, np. przy metodzie get, wówczas by przekazać parametry do serwera. Musimy skorzystać z tej metody
* data - w tej metodzie możemy przekazać dane do serwera, jednakże ten parametr wykorzystujemy w przypadku metody post, patch, put itp.
* headers - definiujemy tutaj dodatkowe nagłówki

Po wykonaniu metody możemy podłączyć kilka metod, co możemy przeczytać na [36], co można zaobserwować na listingu 22.

Listing 22 Wykorzystanie odpowiedzi na wywołanie metody getPosts

*PostService.getPosts(someData)*

*success(function(data, status, headers, config) {  
 }).  
 error(function(data, status, headers, config) {  
 });*

Metoda success wywołuje się wtedy, gdy żądanie do serwera zostanie pomyślnie wykonane. Natomiast metoda error, gdy żądanie do serwera się nie powiedzie. Parametry data, status, headers, config.

Do funkcji success i error, zgodnie z [36] zostają przekazane następujące obiekty:

data - jest to obiekt odpowiedzi JSON (lub inny zdefiniowany).

status - jest to status odpowiedzi

headers - nagłówek odpowiedzi

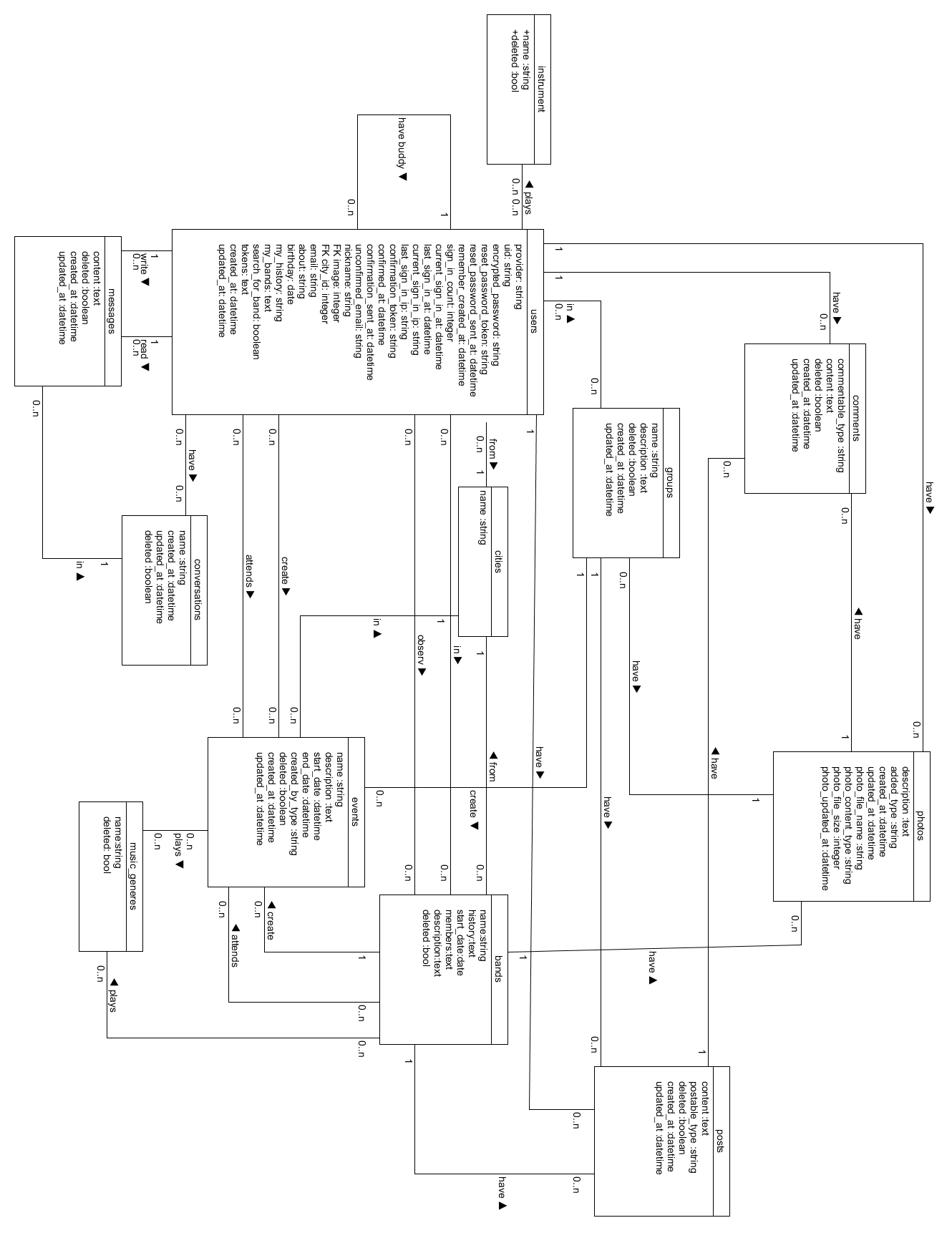
config - jest to konfiguracja pliku, która została użyta do wygenerowania żądania.

Dzięki takim usługom nie powtarzamy kodu, dodatkowo zadbany kod i zgodny z API serwera.

W aplikacji zaimplementowano kilka serwisów, które komunikują się z API:

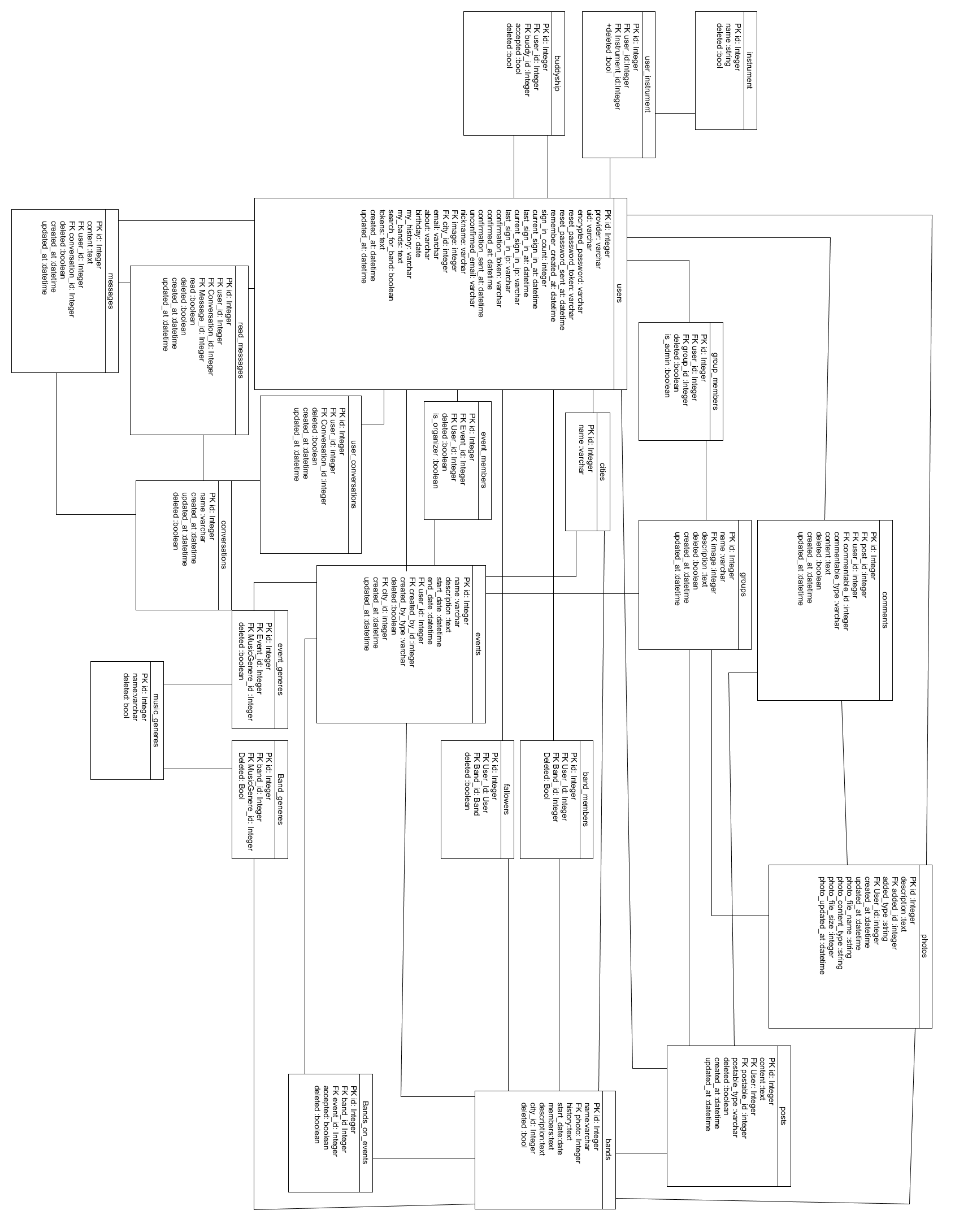
* BandService- ten serwis jest związany z zespołami, pobierania ich postów, tworzenia, aktualizacji, dodawania postów, edytowania postów. Jak i również pobierania wszystkich zespołów lub konkretnego zespołu. Dodawania członków, dodanie zespołu do obserwowanych, jak i opuszczania zespołu, usuwania członków. Ustawienia awataru głównego zespołu. Za pomocą tego serwisu można również stworzyć zespół..
* PostsService - ten serwis związany jest z postami, do tworzenia postów, pobierania postów konkretnego użytkownika, usuwania postów, dodawania, edytowania, usuwania komentarzy.
* GroupService - ten serwis jest związany z grupami, pobierania postów konkretnej grupy. Jak i również pobierania wszystkich grup lub konkretnej grupy. Dodania siebie i usunięcia z grupy. Ustawienia awataru głównego grupy. Możemy za pomocą tego serwisu mianować administratorem grupy. Ustawić podstawowe informacje o grupie.
* EventsService - ten serwis jest związany z wydarzeniami. Jak i również pobierania wszystkich wydarzeń lub konkretnej wydarzania. Wzięcia udziału w wydarzeniu lub odejście z wydarzenia. Możemy za pomocą tego serwisu mianować organizatorem wydarzenia. Ustawić podstawowe informacje o wydarzeniu. Dodatkowo możemy pobrać wszystkie wydarzenia utworzone przez konkretnego użytkownika, zespół lub grupę.
* UserService - jest to serwis związany z użytkownikami, pobierania profilów, edycji podstawowych danych profilu użytkownika.
* QueryService - jest to serwis związany z wyszukiwarkami
* BuddyService - serwis związany z nawiązywaniem znajomości między użytkownikami
* UtilsService - jest to uniwersalny serwis, do pobierania miast, instrumentów oraz gatunków muzycznych.
* ConversationService - jest to serwis związany z modułem konwersacji między użytkownikami, służy do wysyłania wiadomości, tworzenia konwersacji, dodawania i usuwania użytkowników do konwersacji.
* PhotosService - służy do pobierania zdjęć konkretnych użytkowników, zespołów lub grup. Umożliwia również ich usuwanie lub też edycję. Serwisem do tworzenia zdjęć zajmuje się osobna biblioteka angular file upload. Pobrania konkretnego zdjęcia oraz funkcję dodawania i usuwania komentarzy.

## 4.3. Projekt bazy danych i model klas

Na podstawie wskazówek udzielonych w [33]. Stworzyłem diagram klas, który możemy zobaczyć na rysunku 4.1.

Rysunek 4.1 Diagram klas

Następnie zgodnie ze wskazówkami zawartymi w książce [37] znormalizowałem bazę danych, który widzimy na rysunku 4.2.



Rysunek 4.2 Schemat bazy danych

# Implementacja aplikacji

## 5.1 Backend – aplikacja po stronie serwera

### 5.1.1 Algorytmika

W przypadku usuwania zasobu, oprócz zdjęć, wywoływana jest metoda trydelete. Wówczas pole deleted znajdujący się w modelu przyjmuje wartość true, dzięki czemu nie będzie on widoczny przy wyszukiwaniu, wyświetlaniu profilu użytkownika itp.

Jeśli stały zasób, taki jak np. znajomość, ma zostać ponownie stworzony. Wówczas dany rekord jest odnajdywany w bazie danych i pole deleted z powrotem przyjmuje wartość domyślną false. Dzięki temu będzie znowu widoczny przy odszukiwaniu zasobu.

Algorytm wyszukiwarki w przypadku obecności jakiegoś parametru jest zawężany do jego wartości. Z każdym kolejnym parametrem dodawane są kolejne filtry. W aplikacji istnieją 4 typy wyszukiwarek przeznaczonych dla użytkowników, zespołów, wydarzeń i grup.

Przykładowy algorytm pobierający określone zespoły można zobaczyć na listingu 23.

Listing 23 Fragment kodu obsługi wyszukiwarki po stronie serwera

*band\_q= Band.where("bands.name LIKE ?", "%#{params[:name]}%").distinct*

*if params[:plays]*

*band\_q=band\_q.joins(:music\_generes).where("music\_generes.name IN (?)", params[:plays]).limit(10)*

*end*

*if params[:cities]*

*band\_q= band\_q.joins(:city).where("cities.name IN (?)",params[:cities]).limit(10);*

*end*

*band\_q*

### 5.1.2. API

Z aplikacją po stronie serwera możemy komunikować się z za pomocą API. Zapytania przyjmują postać w następującym formacie, jak na listingu 24.

Listing 24 Szablon trasy API

*/api/v1/(zasób)/(id zasobu lub metoda)/(podzasób)?page=0*

Z wysłaniem zapytania na konkretny adres, należy posiadać token powiązany z konkretnym użytkownikiem, by nastąpiło uwierzytelnienie. Zgodnie z dokumentacją na stronie biblioteki [10] należy wysłać dane takie jak email i hasło, na adres /api/v1/auth/sign\_in jako parametry metodą POST. W odpowiedzi otrzymamy 5 danych, które musimy wysyłać w nagłówku kolejnego zapytania np. dołączając plik cookie (co wykorzystuje w swojej aplikacji). W ramach aplikacji zostały stworzone przykładowe trasy, wymagane dane wraz z odpowiedziami możemy zaobserwować na listingach 25-181.

**Trasa: /api/v1/users/:user\_id/posts**

**Opis:** Trasę wykorzystujemy do tworzenia postu dla zalogowanego użytkownika. Zalogowany użytkownik musi dodawać post do swojego profilu.

**Metoda:** POST

**Wymagane dane:**

Listing 25 Wymagane dane do tworzenia postu

*{*

*post:{*

*content: treść postu,*

*}*

*}*

**Wynik:**

Listing 26 Wynik stworzenia postu na profilu użytkownika

*{*

*{*

*id: identyfikator postu*

*content: treść postu.*

*created\_at: data stworzenia postu*

*is\_author: czy obecny użytkownik jest autorem postu*

*user: {*

*id: identyfikator użytkownika*

*nickname; nazwa użytkownika*

*count: ilość wszystkich postów*

*avatar\_original: adres awatara grupy*

*}*

*}*

*}*

**Trasa: /api/v1/comments/:id**

**Opis:** Trasę wykorzystujemy do usunięcia komentarza.

**Metoda:** Delete

**Wymagane dane:**

Listing 27 Wymagane dane do usunięcia komentarza

*{*

*id: identyfikator komentarza.*

*}*

**Wynik:**

Listing 28 Wynik operacji po usunięciu komentarza

*{*

*id: identyfikator komentarza.*

*deleted: wynik operacji.*

*}*

**Trasa: /api/v1/utils/get\_cities**

**Opis**: Trasę wykorzystujemy do pobrania wszystkich miast.

**Metoda:** GET

**Wymagane dane:** brak

**Wynik:**

Listing 29 Wynik operacji pobrani wszystkich miast

*{*

*{*

*id: identyfikator miasta.*

*name: nazwa miasta np. Wrocław.*

*}*

*...*

*}*

**Trasa: /api/v1/bands/query?name=someValue&plays[]=someGenere&cities[]=someCity**

**Opis**: Trasę wykorzystujemy do wyszukiwania zespołów.

**Metoda**: GET

**Wymagane dane** (opcjonalne):

Listing 30 Dane opcjonalne do wyszukiwania zespołów

*{*

*name: nazwa zespołu,*

*plays[]: tablica z nazwami gatunków muzycznych np. “Rock”.*

*cities[]: tablica z nazwami miast.*

*page: paginacja po wynikach.*

*}*

**Wynik**:

Listing 31 Wynik wyszukiwania zespołów

*{*

*id: identyfikator zespołu.*

*name: nazwa zespołu.*

*avatar\_original: nazwa zespołu.*

*band\_generes[]: gatunki muzyczne, które gra zespół.*

*city: nazwa miasta z którego jest zespół.*

*}*

**Trasa: /api/v1/events/:event\_id/set\_music\_generes**

**Opis:** Trasę wykorzystujemy do zmiany gatunków muzycznych związanych z wydarzeniem.

**Metoda:** POST

**Wymagane dane:**

Listing 32 Dane wymagane do zmiany gatunków muzycznych związanych z wydarzeniem.

*{*

*event\_id: identyfikator wydarzenia,.*

*toAppend[]: {*

*id: identyfikator gatunku muzycznego do dodania*

*}*

*toDelete[]:{*

*id: identyfikator gatunku muzycznego do usunięcia*

*}*

*}*

**Wynik:**

Listing 33 Wynik zmiany gatunków muzycznych związanych z wydarzeniem

*{*

*id: identyfikator wydarzenia.*

*music\_generes[]:{*

*id: identyfikator gatunku muzycznego.*

*name : nazwa gatunku muzycznego.*

*}*

*}*

**Trasa: /api/v1/events/:event\_id/set\_member**

**Opis:** Trasę wykorzystujemy do nadania/usunięcia praw organizatora dla konkretnego użytkownika. Możemy to zrealizować tylko, jeśli użytkownik nadający/usuwający uprawnienia jest organizatorem wydarzenia.

**Metoda:** PUT

**Wymagane dane:**

Listing 34 Wymagane dane do nadania/usunięcia praw organizatora

*{*

*event\_id: identyfikator wydarzenia.*

*organizer: czy użytkownik ma być organizatorem.*

*user\_id: identyfikator użytkownika.*

*}*

**Wynik:**

Listing 35 Wynik nadania/usunięcia praw organizatora

*{*

*id: identyfikator wydarzenia.*

*users[]:{*

*id: identyfikator użytkownika.*

*name: nazwa użytkownika.*

*avatar\_original: awatar użytkownika.*

*is\_organizer: czy użytkownik ma być organizatorem.*

*}*

*}*

**Trasa: /api/v1/bands/:id**

**Opis:** Trasę wykorzystujemy do pobrania zespołu .

**Metoda:** GET

**Wymagane parametry**:

Listing 36 Dane wymagane do pobrania danych danego zespołu

*{*

*band\_id: identyfikator zespołu.*

*}*

**Wynik**:

Listing 37 Wynik pobrania zespołu

*{*

*id - identyfikator zespołu.*

*name -nazwa zespołu.*

*history - historia zespołu.*

*members - opis członków zespołu.*

*start\_date - data rozpoczęcia działalności.*

*description - opis zespołu.*

*is\_member - czy aktywny użytkownik jest członkiem zespołu.*

*is\_fallower - czy aktywny użytkownik obserwuje zespół.*

*music\_generes[]:{*

*id: identyfikator gatunku muzycznego.*

*name : nazwa gatunku muzycznego.*

*}*

*city {*

*id: identyfikator miasta*

*name : nazwa miasta*

*}*

*avatar\_original awatar zespołu*

*user\_members[]: {*

*id: identyfikator użytkownika.*

*nickname : nazwa użytkownika.*

*avatar\_original: awatar użytkownika.*

*}*

*}*

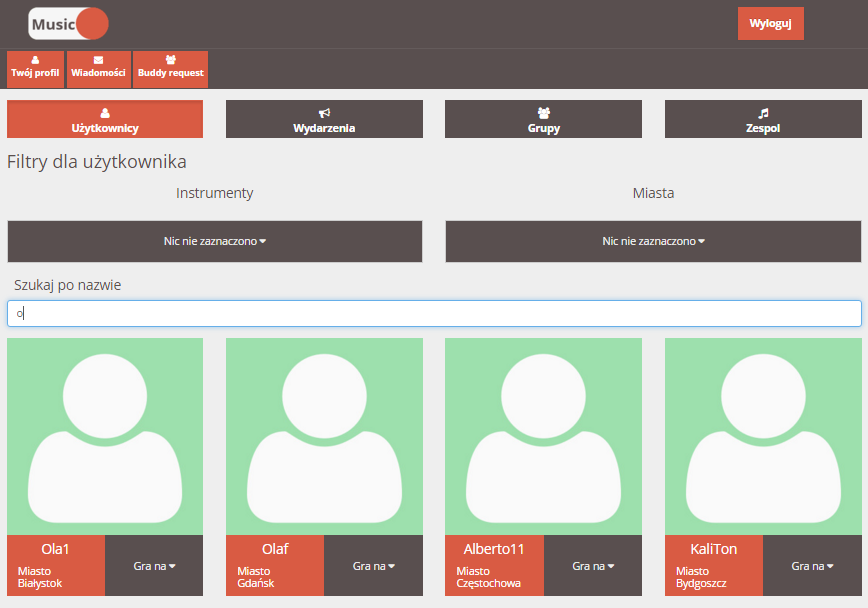
## 5.2. Frontend – aplikacja po stronie klienta

### 5.2.1 Wyszukiwarka

Wyszukiwanie działa w następujący sposób. W chwili wpisywania tekstu zostanie wykonywane zapytanie za pomocą AJAX-a z odpowiednią trasą API. Dane te są przetwarzane przez serwer, a następnie wynik jest zwracany do klienta. Podobnie sytuacja ma miejsce w chwili, gdy modyfikujemy filtr np. gatunek muzyczny. Wówczas wyniki są zwracane z zastosowaniem filtra.

Dodatkowa opcja ma miejsce w zależności od tego jaki typ danych wyszukujemy. Obecnie możliwe wyszukiwanie jest możliwe do zrealizowania:

1. Użytkowników
2. Grupy
3. Wydarzeń
4. Zespołów



Rysunek 5.1 Widok wyszukiwarki

Jak możemy zauważyć z rysunku 5.1możemy zauważyć, że zostali wyszukiwani użytkownicy, którzy mają w swoimi nicku literę o.

### 5.2.2. Tagowanie

Tagowanie jest zrealizowane w sposób następujący. Na element blokowy div ustawiamy atrybut contenteditable. Wówczas możemy wprowadzać w nim wartości wprowadzone z klawiatury oraz zagnieżdżać kod HTML.

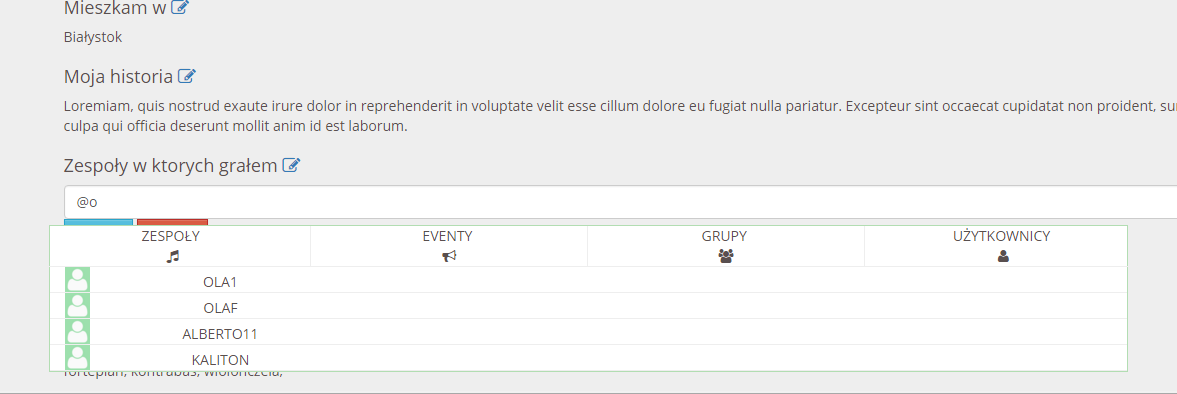
Po każdym wprowadzonym znaku tekst jest przeszukiwany do kursora w poszukiwaniu znaku “@”. W chwili wykrycia tego znaku i spełnieniu warunków (po znaku nie znajduje się kropka, średnik, przecinek, inne znaki specjalne, podwójna spacje). Zostaje wyciągany tekst począwszy od początku znaku „@” do pozycji kursowa. Następnie wyświetlane jest okno modalne, które ma 4 możliwe opcje otagowania:

1. Użytkowników
2. Grupy
3. Wydarzeń
4. Zespołów

Tekst występujący po znaku “@” będzie służył jako tekst wyszukujący np. po nicku użytkownika. Kolejnym znakiem kończącym wyszukiwanie jest:

* podwójna spacja
* kropka, średnik, przecinek oraz inne znaki specjalne

Gdy klikniemy link w okienku modalnym tag z tekstem zostanie zamieniony na link, tekst w linku możemy od razu zmodyfikować lub po prostu kontynuować pisanie tekstu.



Rysunek 5.2 Widok tagowania

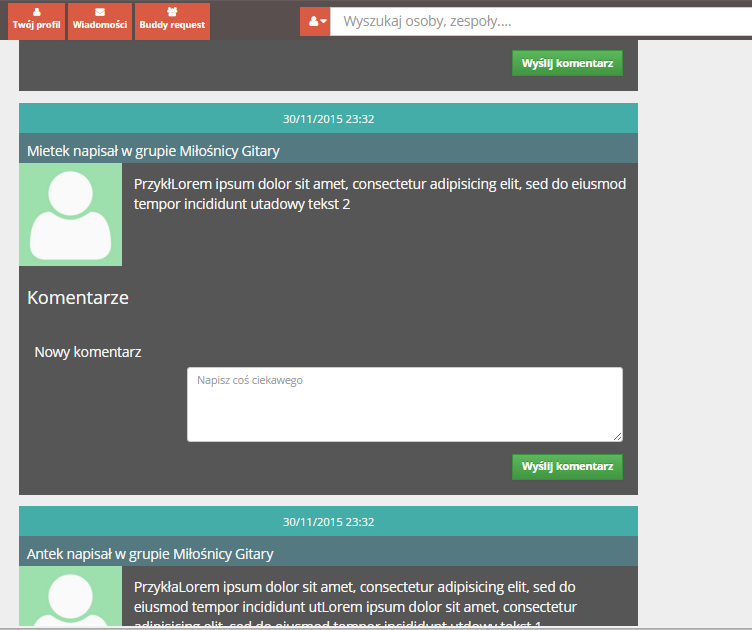
Jak można zauważyć na rysunku 5.2, zostali wybrani wszyscy użytkownicy, których nick zawierał literę o. Awatar jak i nazwa użytkownika są linkiem, który dodaje tag do pola.

### 5.2.3 Przykładowe widoki, które zostały użyte w aplikacji

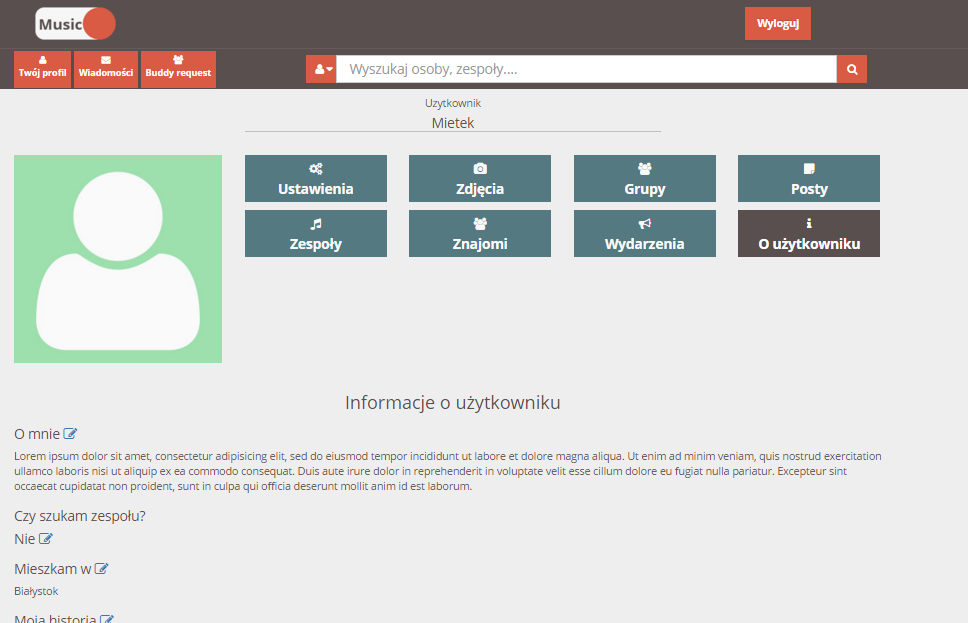
Jak możemy zauważyć na rysunkach 5.3, 5.4, 5.5 , zostały stworzone następujące widoki z wykorzystaniem klas Bootstrap oraz własnych stylów CSS. Widok na rysunku 5.3 przedstawia widok odpowiadający pojedynczemu wydarzniu, natomiast widok 5.4, przedstawia fragment strony głównej. Na rysunku 5.5 możemy zaobserwować wygląd podstawowych informacji o użytkowniku.



Rysunek 5.3 Widok wydarzenia



Rysunek 5.4 Widok postów na stronie głównej



Rysunek 5.5 Widok informacji o użytkowniku

# Podsumowanie

## 6.1. Podsumowanie

W ramach implementacji zostały wdrożone funkcjonalności takie jak: zakładanie grup, wydarzeń, zespołów. Możliwe jest dodanie postów i zdjęć na profilu użytkownika, zespołu oraz grup. Dodatkowo wprowadzono możliwość komentowania postów oraz zdjęć. Wprowadzono wyszukiwanie użytkowników, grup, wydarzeń i zespołów. W ramach zespołu możemy wyświetlić listę ludzi, którzy obserwują zespół, wyszukiwać ludzi chcących znaleźć zespół, zobaczyć na jakich wydarzeniach zagra dany zespół oraz pokazać podstawowe informacje o zespole wraz ze zdjęciami i postami. W ramach użytkownika możemy wyświetlić zespoły, w których użytkownik uczestniczy lub obserwuje. Dodatkowo wyświetlić grupy, w których jest użytkownik. Możliwe jest tworzenie konwersacji z użytkownikami i wysyłanie wiadomości. Możliwe jest tagowanie użytkowników, zespołów, wydarzeń oraz grup. Możliwe jest utworzenie grup wraz z podstawowymi funkcjonalnościami, takimi jak: tworzenie postów, dodawanie zdjęć itp.

## 6.2. Perspektywy rozwoju

W ramach rozwoju projektu można dołączyć kilka funkcjonalności, które uczyniłoby korzystanie z portalu atrakcyjniejszym.

Integracja z Google API Maps

W ramach tej funkcjonalności, użytkownik mógłby wyszukiwać coś w ramach jakiegoś obszaru na mapie. Z uwagi na ograniczenia narzucone przez firmę Google w ramach wykorzystania ich interfejsu. Wymagany będzie zawarcie umowy o wersję Premium. W przypadku tego serwisu ma sens wyszukiwanie użytkowników, wydarzeń oraz zespołów. Ograniczenia, które występują przy tym interfejsie to m.in. ilość możliwych zapytań. Za więcej zapytań trzeba już niestety zapłacić określoną sumę. Z uwagi na to, że serwis może być odwiedzany przez bardzo dużą grupę użytkowników, trzeba będzie rozważyć nawiązanie stałej współpracy z Google lub implementację własnego rozwiązania podobnego do Google Maps.

Lokalizacja serwisu

Obecnie serwis jest w języku polskim. Jednakże jeśli będzie potrzeba rozwinięcia działalności serwisu. Będzie trzeba umożliwić użytkownikowi zmienienie języku przy użytkownikowi serwisu. Planowane języki to np. angielski, niemiecki, rosyjski, francuski. Dodatkowo przy otwarciu serwisu na większy obszar terytorialny, będzie trzeba wprowadzić możliwość wybrania kraju oraz miasta, z którego pochodzi użytkownik.

Możliwość dodawania utworów

Jako, że jest to portal muzyczny. Przydałaby się opcja dodawania własnych utworów muzycznych np. w formie filmiku, utworu audio itp. Wówczas każde nagranie może być dodane przez użytkownika lub zespół. Należałoby również umożliwić komentowanie utworu. Najbardziej popularne demo lub wybrane przez użytkownika byłoby wyświetlane w ramach promocji użytkownika.

W ramach jakiejś konwersacji, użytkownik mógłby prowadzić rozmowę wideo z wybranymi przez siebie użytkownikami. W ramach innych serwisów takich jak YouTube, Vimeo, LastFM dać możliwość udostępniania np. jakiegoś filmiku, który byłby wartościowy by dodać go jako np. post. Poprawa jakości stylowania dla mniejszych rozdzielczości np. wyglądu strony na urządzeniach mobilnych.

Jednym z ostatnich potencjalnych perspektyw rozwoju aplikacji byłaby możliwość dodania logowania za pomocą Google+ czy Facebooka. Jest rozwiązanie, które staje się coraz popularniejsze wśród rozwiązań serwisów społecznościowych.

# Bibliografia

[1] **Społeczność AngularJS:** *Dokumentacja platformy AngularJS.* 20.11.2015 https://docs.angularjs.org/.

[2] **Freeman Adam:** *AngularJS Profesjonalne Techniki.* Gliwice  : Helion, 2015

[3] **Społeczność AngularJS:** *Informacje dotyczące kompatybilności biblioteki z przeglądarkami internetowymi.* 19.11.2015 https://docs.angularjs.org/misc/faq.

[4] **Bruce Tate, Lance Carlson, Curt Hibbs.** *Ruby on Rails. Wprowadzenie. Wydanie II .* Gliwice : Helion, 2009.

[5] **Społeczność Ruby On Rails:** *Informacje o platformie programistycznej.* 20.11.2015 http://guides.rubyonrails.org/getting\_started.html.

[6] **Społeczność SQLite:** *Informacje o silniku SQLite.* 20.11.2015 www.sqlite.org/whentouse.html.

[7] **Użytkownik** **Nesquena:** *Dokumentacja biblioteki RABL.* 26.11.2015 https://github.com/nesquena/rabl.

[8] **Użytkownik** **Olher55:** *Dokumentacja biblioteki OJ.* 20.11.2015 https://github.com/ohler55/oj.

[9] **Użytkownik** **Hashrocket:** *Dokumentacja biblioteki Decent exposure*. 20.11.2015 https://github.com/hashrocket/decent\_exposure.

[10] **Użytkownik** **Lynndylanhurley:** *Dokumentacja biblioteki devise token auth.* 20.11.2015 https://github.com/lynndylanhurley/devise\_token\_auth.

[11] **Użytkownik** **Plataformatec:** *Dokumentacja biblioteki Devise.* 20.11.2015 https://github.com/plataformatec/devise.

[12] **Użytkownik** **Codehale:** *Biblioteka wprowadzająca BCrypt.* 30.11.2015 https://github.com/codahale/bcrypt-ruby.

[13] **Użytkownik** **Cyu:** *Dokumentacja biblioteki rack-cors.* 20.11.2015 https://github.com/cyu/rack-cors.

[14] **Użytkownik** **Intridea:** *Dokumentacja biblioteki omniauth.* 20.11.2015 https://github.com/intridea/omniauth .

[15] **Użytkownik** **Thoughtbot:** *Dokumentacja biblioteki paperclip.* 20.11.2015 https://github.com/thoughtbot/paperclip.

[16] **ImageMagick Studio LLC:** *Strona internetowa oprogramowania ImageMagick.* 20.11.2015 http://www.imagemagick.org/script/index.php.

[17] **Społeczność Github:** *Strona związana z opisem błędu Paperclip i jego obejściem.* 20.11.2015 https://github.com/thoughtbot/paperclip/issues/1429 .

[18] **Deivid-rodriguez:** *Dokumentacja biblioteki pry-byebug.* 20.11.2015 https://github.com/deivid-rodriguez/pry-byebug/.

[19] **Społeczność Thoughtbot:** *Dokumentacja biblioteki factory\_girl.* 20.10.2015 https://github.com/thoughtbot/factory\_girl\_rails.

[20] **Steven Baker:** *Dokumentacja i biblioteka rspec.* 20.11.2015 https://github.com/rspec/rspec-rails.

[21] **Społeczność Angular UI:** *Dokumentacja biblioteki Angular ui router.* 10.11.2015 https://github.com/angular-ui/ui-router.

[22] **Użytkownik** **Lynndylanhurley:** *Dokumentacja biblioteki ng token auth.* 20.11.2015 https://github.com/lynndylanhurley/ng-token-auth.

[23] **Użytkownik** **Nervgh:** *Dokumentacja Angular File upload*. 20.11.2015 https://github.com/nervgh/angular-file-upload.

[24] **Użytkownik** **Ivpusic:** *Dokumentacja Angular Cookie*. 20.11.2015 https://github.com/ivpusic/angular-cookie.

[25] **Społeczność Bootstrap:** *O Bootstrapie.* 20.11.2015 http://getbootstrap.com.

[26] **Społeczność JQuery:** *Dokumentacja jQuery.* 30.11.2015 https://api.jquery.com/.

[27] **Firdaus Thoriq:** *Responsive Web Design. Nowoczesne strony WWW na przykładach.* Gliwice  : Helion, 2014.

[28] **Frain Ben:** *Responsive Web Design. Projektowanie elastycznych witryn w HTML5 i CSS3.* Gliwice  : Helion, 2013.

[29] **Społeczność Ruby On Rails:** *Dokumentacja frameworku Ruby on Rails - o Active Record.* 20.11.2015 http://guides.rubyonrails.org/active\_record\_basics.html.

[30] **Społeczność Ruby On Rails:** *Dokumentacja frameworku Ruby on Rails o interfejsie wyszukiwania.* 20.11.2015 http://guides.rubyonrails.org/active\_record\_querying.html.

[31] **Społeczność Ruby On Rails:** *Dokumentacja Ruby on Rails - o asocjacji.* 20.11.2015 http://guides.rubyonrails.org/association\_basics.html.

[32] **Społeczność Angular UI:** *Dokumentacja biblioteki Angular UI Router - przewodnik.* 20.11.2015 https://github.com/angular-ui/ui-router/wiki.

[33] **Śmiałek Michał:** *Zrozumieć UML 2.0. Metody modelowania obiektowego.* Gliwice : Helion, 2005.

[34] **Społeczność Ruby On Rails:** *Dokumentacja frameworku Ruby on Rails - o trasach.*  20.11.2015 http://guides.rubyonrails.org/routing.html.

[35] **Społeczność AngularJS:** *Dokumentacja Angular JS o serwisach. 20.11.2015* https://code.angularjs.org/1.4.0/docs/guide/providers.

[36] **Społeczność AngularJS:** *Dokumentacja Angular JS o $http. 20.11.2015* https://code.angularjs.org/1.4.0/docs/api/ng/service/$http.

[37] **Hanna Mazur, Zygmunt Mazur.** *Projektowanie relacyjnych baz danych.* Wrocław : Oficyna Wydawnicza Politechniki Wrocławskiej, 2004.

[38] **Społeczność AngularJS:** *Dokumentacja Angular JS - o kontrolerach.* 20.11.2015 https://code.angularjs.org/1.4.0/docs/guide/controller.

# Spis rysunków

[Rysunek 3.1 Moduł obserwatora zespołu 17](#_Toc436934539)

[Rysunek 3.2 Moduł postów na profilu użytkownika 20](#_Toc436934540)

[Rysunek 3.3 Moduł postów na profilu zespołu 23](#_Toc436934541)

[Rysunek 3.4 Moduł przynależności 25](#_Toc436934542)

[Rysunek 3.5 Moduł profil użytkownika 28](#_Toc436934543)

[Rysunek 3.6 Moduł profilu zespołu 33](#_Toc436934544)

[Rysunek 3.7 Moduł przynależności do zespołu 36](#_Toc436934545)

[Rysunek 3.8 Moduł rejestracji 37](#_Toc436934546)

[Rysunek 3.9 Moduł strony głównej 38](#_Toc436934547)

[Rysunek 3.10 Moduł tagowania 39](#_Toc436934548)

[Rysunek 3.11 Moduł zdjęć 41](#_Toc436934549)

[Rysunek 3.12 Moduł znajomości 45](#_Toc436934550)

[Rysunek 3.13 Moduł profil innego użytkownika 47](#_Toc436934551)

[Rysunek 3.14 Moduł zdjęć zespołów 50](#_Toc436934552)

[Rysunek 4.1 Diagram klas 60](#_Toc436934553)

[Rysunek 4.2 Schemat bazy danych 61](#_Toc436934554)

[Rysunek 5.1 Widok wyszukiwarki 67](#_Toc436934555)

[Rysunek 5.2 Widok tagowania 68](#_Toc436934556)

[Rysunek 5.3 Widok wydarzenia 69](#_Toc436934557)

[Rysunek 5.4 Widok postów na stronie głównej 69](#_Toc436934558)

[Rysunek 5.5 Widok informacji o użytkowniku 70](#_Toc436934559)

[Rysunek B.1 Moduł postów na profilu grupy 119](#_Toc436934560)

[Rysunek B.2 Moduł profilu grupy 120](#_Toc436934561)

[Rysunek B.3 Moduł zdjęć grupy 121](#_Toc436934562)

[Rysunek B.4 Moduł wydarzeń zespołu 122](#_Toc436934563)

[Rysunek B.5 Moduł użytkownika grupy 122](#_Toc436934564)

[Rysunek B.6 Moduł profilu wydarzenia 123](#_Toc436934565)

[Rysunek B.7 Moduł uczestnika wydarzenia 124](#_Toc436934566)

[Rysunek B.8 Moduł konwersacji 125](#_Toc436934567)

Rysunki diagramów UML zostały wykonane w programie Umlet dostępnego pod adresem [www.umlet.com](http://www.umlet.com)

# Spis listingów

[Listing 1 Sposób pobierania obiektu w Rails 6](#_Toc436935113)

[Listing 2 Sposób pobierania obiektu z wykorzystaniem biblioteki Decent Exposure 6](#_Toc436935114)

[Listing 3 Tworzenie modelu postu 7](#_Toc436935115)

[Listing 4 Atrybuty tabeli posts 7](#_Toc436935116)

[Listing 5 Przykład tworzenia obiektów 8](#_Toc436935117)

[Listing 6 Log po wywołaniu metody post.all 8](#_Toc436935118)

[Listing 7 Log po wywołaniu metody post.select(:content) 9](#_Toc436935119)

[Listing 8 Log po wywołaniu metody post.offset(2).limit(10) 9](#_Toc436935120)

[Listing 9 Log po wywołaniu metody Post.where("id=?",2).order(:created\_at).first 9](#_Toc436935121)

[Listing 10 Podejście z zmianą wartości określonych parametrów 9](#_Toc436935122)

[Listing 11 Aktualizacja za pomocą parametru 10](#_Toc436935123)

[Listing 12 Usuwanie obiektu za pomocą metody destroy 10](#_Toc436935124)

[Listing 13 Definicja przykładowego stanu 11](#_Toc436935125)

[Listing 14 Konstrukcja kontrolera 11](#_Toc436935126)

[Listing 15 Przykładowa trasa API 55](#_Toc436935127)

[Listing 16 Zagnieżdżanie zasobu w zasobie routes.rb 56](#_Toc436935128)

[Listing 17 Plik routes.rb i dodanie metody "members" konkretnego zespołu 56](#_Toc436935129)

[Listing 18 Wynik stworzenia trasy 56](#_Toc436935130)

[Listing 19 Plik routes.rb i dodanie metody do kolekcji "members" 56](#_Toc436935131)

[Listing 20 Wynik stworzenia trasy 56](#_Toc436935132)

[Listing 21 Konstrukcja serwisu z wstrzykniętą usługą $http 57](#_Toc436935133)

[Listing 22 Wykorzystanie odpowiedzi na wywołanie metody getPosts 58](#_Toc436935134)

[Listing 23 Fragment kodu obsługi wyszukiwarki po stronie serwera 62](#_Toc436935135)

[Listing 24 Szablon trasy API 62](#_Toc436935136)

[Listing 25 Wymagane dane do tworzenia postu 63](#_Toc436935137)

[Listing 26 Wynik stworzenia postu na profilu użytkownika 63](#_Toc436935138)

[Listing 27 Wymagane dane do usunięcia komentarza 63](#_Toc436935139)

[Listing 28 Wynik operacji po usunięciu komentarza 63](#_Toc436935140)

[Listing 29 Wynik operacji pobrani wszystkich miast 64](#_Toc436935141)

[Listing 30 Dane opcjonalne do wyszukiwania zespołów 64](#_Toc436935142)

[Listing 31 Wynik wyszukiwania zespołów 64](#_Toc436935143)

[Listing 32 Dane wymagane do zmiany gatunków muzycznych związanych z wydarzeniem. 64](#_Toc436935144)

[Listing 33 Wynik zmiany gatunków muzycznych związanych z wydarzeniem 65](#_Toc436935145)

[Listing 34 Wymagane dane do nadania/usunięcia praw organizatora 65](#_Toc436935146)

[Listing 35 Wynik nadania/usunięcia praw organizatora 65](#_Toc436935147)

[Listing 36 Dane wymagane do pobrania danych danego zespołu 66](#_Toc436935148)

[Listing 37 Wynik pobrania zespołu 66](#_Toc436935149)

[Listing 38 Wymagane dane do aktualizacji wydarzenia. 79](#_Toc436935150)

[Listing 39 Wynik aktualizacji wydarzenia 79](#_Toc436935151)

[Listing 40 Wymagane dane do usunięcia wydarzenia 79](#_Toc436935152)

[Listing 41 Wynik operacji usunięcia zespołu 79](#_Toc436935153)

[Listing 42 Wymagane dane do potwierdzenia przez zespół wzięcia udziału w wydarzeniu 80](#_Toc436935154)

[Listing 43 Wynik operacji potwierdzenia zaproszenia dla zespołu 80](#_Toc436935155)

[Listing 44 Wymagane dane do wyszukiwania grup 80](#_Toc436935156)

[Listing 45 Wynk operacji wyszukiwania 80](#_Toc436935157)

[Listing 46 Wymagane dane do ustawiania awatarów dla grupy 81](#_Toc436935158)

[Listing 47 Wynik operacji ustawienia awataru dla grupy 81](#_Toc436935159)

[Listing 48 Wymagane dane do dodania zalogowanego użytkownika do grupy 81](#_Toc436935160)

[Listing 49 Wynik operacji 81](#_Toc436935161)

[Listing 50 Wymagane dane do usununięcia użytkownika grupy 81](#_Toc436935162)

[Listing 51 Wynik usunięcia użytkownika z grupy 81](#_Toc436935163)

[Listing 52 Wymagane dane do nadania praw administratora grupy 82](#_Toc436935164)

[Listing 53 Wynik operacji 82](#_Toc436935165)

[Listing 54 Wymagane dane do pobrania postów grupy 82](#_Toc436935166)

[Listing 55 Wynik operacji 82](#_Toc436935167)

[Listing 56 Dane wymagane do pobrania zdjęć grupy 83](#_Toc436935168)

[Listing 57 Wynik operacji 83](#_Toc436935169)

[Listing 58 Wymagane dane do pobrania wydarzeń grupy 84](#_Toc436935170)

[Listing 59 Wynik operacji 84](#_Toc436935171)

[Listing 60 Wymagane dane do tworzenia wydarzenia 85](#_Toc436935172)

[Listing 61 Wymagane dane 85](#_Toc436935173)

[Listing 62 Wymagane dane do tworzenia grupy 85](#_Toc436935174)

[Listing 63 Wynik operacji 85](#_Toc436935175)

[Listing 64 Wymagane dane do pobrania grupy 85](#_Toc436935176)

[Listing 65 Wynik operacji 86](#_Toc436935177)

[Listing 66 Wymagane dane do aktualizacji grupy 86](#_Toc436935178)

[Listing 67 Wymagane dane 86](#_Toc436935179)

[Listing 68 Wymagane dane do usunięcia grupy 86](#_Toc436935180)

[Listing 69 Wynik operacji 87](#_Toc436935181)

[Listing 70 Wymagane dane do dodania użytkownika do konwersacji 87](#_Toc436935182)

[Listing 71 Wynik operacji 87](#_Toc436935183)

[Listing 72 Wymagane dane do usunięcia użytkownika z konwersacji 87](#_Toc436935184)

[Listing 73 Wymagane dane 87](#_Toc436935185)

[Listing 74 Wynik operacji pobrania wszystkich konwersacji 88](#_Toc436935186)

[Listing 75 Wymagane dane do tworzenia konwersacji 88](#_Toc436935187)

[Listing 76 Wynik operacji 88](#_Toc436935188)

[Listing 77 Wymagane dane do pobrania konwersacji 89](#_Toc436935189)

[Listing 78 Wynik operacji 89](#_Toc436935190)

[Listing 79 Wymagane dane do aktualizacji nazwy konwersacji 90](#_Toc436935191)

[Listing 80 Wynik operacji 90](#_Toc436935192)

[Listing 81 Wymagane dane do pobrania miniaturki zdjęcia 90](#_Toc436935193)

[Listing 82 Wynik operacji 90](#_Toc436935194)

[Listing 83 Wymagane dane do pobrania oryginału zdjęcia 90](#_Toc436935195)

[Listing 84 Wynik operacji 90](#_Toc436935196)

[Listing 85 Wymagane dane do dodania komentarza do zdjęcia 91](#_Toc436935197)

[Listing 86 Wynik operacji 91](#_Toc436935198)

[Listing 87 Wymagane dane do tworzenia zdjęć 91](#_Toc436935199)

[Listing 88 Wynik operacji 91](#_Toc436935200)

[Listing 89 Wymagane dany do pobrania zdjęć 92](#_Toc436935201)

[Listing 90 Wynik operacji 92](#_Toc436935202)

[Listing 91 Wymagane dane do aktualizacji opisu zdjęcia 93](#_Toc436935203)

[Listing 92 Wynik operacji 93](#_Toc436935204)

[Listing 93 Wymagane dane do usunięcia zdjęcia 93](#_Toc436935205)

[Listing 94 Wynik operacji 93](#_Toc436935206)

[Listing 95 Opcjonalne dane do wyszukiwania użytkowników 93](#_Toc436935207)

[Listing 96 Wynik operacji 94](#_Toc436935208)

[Listing 97 Opcjonalne dane do wyszukiwania użytkowników do zespołu 94](#_Toc436935209)

[Listing 98 Wynik operacji 94](#_Toc436935210)

[Listing 99 Wymagane dane do ustawienia awataru dla użytkownika 95](#_Toc436935211)

[Listing 100 Wynik operacji 95](#_Toc436935212)

[Listing 101 Wymagane dane do tworzenia wydarzenia użytkownika 95](#_Toc436935213)

[Listing 102 Wymagane dane 95](#_Toc436935214)

[Listing 103 Wymagane dane do aktualizacji danych użytkownika 96](#_Toc436935215)

[Listing 104 Wynik operacji 96](#_Toc436935216)

[Listing 105 Wymagane dane do aktualizacji instrumentów użytkownika 96](#_Toc436935217)

[Listing 106 Wynik operacji 97](#_Toc436935218)

[Listing 107 Wymagane dane do pobrania postów użytkownika 97](#_Toc436935219)

[Listing 108 Wynik operacji 97](#_Toc436935220)

[Listing 109 Wymagande dane do pobrania znajomych użytkownika 98](#_Toc436935221)

[Listing 110 Wynik operacji 98](#_Toc436935222)

[Listing 111 Wymagane dane do pobrania zdjęć użytkownika 98](#_Toc436935223)

[Listing 112 Wynik operacji 98](#_Toc436935224)

[Listing 113 Wymagande dane do pobrania zespołów użytkownika 99](#_Toc436935225)

[Listing 114 Wynik operacji 99](#_Toc436935226)

[Listing 115 Wymagane dane do pobrania wydarzeń użytkownika 99](#_Toc436935227)

[Listing 116 Wynik operacji 99](#_Toc436935228)

[Listing 117 Wymagane dane do pobrania użytkownika 100](#_Toc436935229)

[Listing 118 Wynik operacji 100](#_Toc436935230)

[Listing 119 Wymagane dane do dodania komentarza do postu 101](#_Toc436935231)

[Listing 120 Wymagane dane do pobrania postów na stronę główną 101](#_Toc436935232)

[Listing 121 Wynik operacji 101](#_Toc436935233)

[Listing 122 Wymagande dane do zaakceptowania zaproszenia do znajomych 102](#_Toc436935234)

[Listing 123 Wynik operacji 102](#_Toc436935235)

[Listing 124 Wymagane dane do pobrania niezaakceptowanych znajomości 103](#_Toc436935236)

[Listing 125 Wynik operacji 103](#_Toc436935237)

[Listing 126 Wymagane dane do wysłania zaproszenia do znajomych 103](#_Toc436935238)

[Listing 127 Wynik operacji 103](#_Toc436935239)

[Listing 128 Wymagane dane do usuwania znajomości 103](#_Toc436935240)

[Listing 129 Wynik operacji 103](#_Toc436935241)

[Listing 130 Wymagane dane do zmiany gatunków muzycznych 104](#_Toc436935242)

[Listing 131 Wynik operacji 104](#_Toc436935243)

[Listing 132 Wymagane dane do pobrania użytkowników, którzy obserwują zespół 104](#_Toc436935244)

[Listing 133 Wynik operacji 104](#_Toc436935245)

[Listing 134 Wymagane dane do ustawienia awatara dla zespołu 105](#_Toc436935246)

[Listing 135 Wynik operacji 105](#_Toc436935247)

[Listing 136 Wymagane dane do dodania użytkownika do składu zespołu 105](#_Toc436935248)

[Listing 137 Wynik operacji 105](#_Toc436935249)

[Listing 138 Wymagane dane do rozpoczęcia obserwacji zespołu 105](#_Toc436935250)

[Listing 139 Wynik operacji 106](#_Toc436935251)

[Listing 140 Wymagane dane do usunięcia użytkownika ze składu zespołu 106](#_Toc436935252)

[Listing 141 Wynik operacji 106](#_Toc436935253)

[Listing 142 Wymagane do zaprzestania obserwacji zespołu 106](#_Toc436935254)

[Listing 143 Wynik operacji 107](#_Toc436935255)

[Listing 144 Wymagane dane do pobrania postów zespołu 107](#_Toc436935256)

[Listing 145 Wynik operacji 107](#_Toc436935257)

[Listing 146 Wymagane dane do pobrania zdjęć zespołu 108](#_Toc436935258)

[Listing 147 Wynik operacji 108](#_Toc436935259)

[Listing 148 Wymagane dane do utworzenia wydarzenia zespołu 108](#_Toc436935260)

[Listing 149 Wynik operacji 109](#_Toc436935261)

[Listing 150 Wymagane dane do pobrana wydarzeń zespołu 109](#_Toc436935262)

[Listing 151 Wynik operacji 109](#_Toc436935263)

[Listing 152 Wymagane dane do tworzenia zespołu 110](#_Toc436935264)

[Listing 153 Wynik operacji 110](#_Toc436935265)

[Listing 154 Wymagane dane do aktualizacji informacji o zespole 110](#_Toc436935266)

[Listing 155 Wynik operacji 110](#_Toc436935267)

[Listing 156 Wymagane dane do usunięcia zespołu 111](#_Toc436935268)

[Listing 157 Wynik operacji 111](#_Toc436935269)

[Listing 158 Wymagane dane do tworzenia postu na tronie zespołu 111](#_Toc436935270)

[Listing 159 Wynik operacji 111](#_Toc436935271)

[Listing 160 Wymagane dane do tworzenia postu dla konkretnej grupy 112](#_Toc436935272)

[Listing 161 Wynik operacji 112](#_Toc436935273)

[Listing 162 Wymagane dane do usuwania postu 113](#_Toc436935274)

[Listing 163 Wynik operacji 113](#_Toc436935275)

[Listing 164 Wymagane dane do aktualizacji postu 113](#_Toc436935276)

[Listing 165 Wynik operacji 113](#_Toc436935277)

[Listing 166 Wynik operacji 114](#_Toc436935278)

[Listing 167 Wynik operacji 114](#_Toc436935279)

[Listing 168 Wymagane dane do wyszukiwania wydarzeń 114](#_Toc436935280)

[Listing 169 Wynik opercji 114](#_Toc436935281)

[Listing 170 Wymagane dane do dodania zespołu do wydarzenia 115](#_Toc436935282)

[Listing 171 Wynik operacji 115](#_Toc436935283)

[Listing 172 Wymagane dane do usnięcia zespołu z wydarzenia 116](#_Toc436935284)

[Listing 173 Wynik operacji 116](#_Toc436935285)

[Listing 174 Wymagane dane do dodania użytkownika do wydarzenia 116](#_Toc436935286)

[Listing 175 Wynik operacji 116](#_Toc436935287)

[Listing 176 Wymagane dane do usunięcia użytkownika z wydarzenia 117](#_Toc436935288)

[Listing 177 Wynik operacji 117](#_Toc436935289)

[Listing 178 Wymagane dane do aktualizacji komentarza 117](#_Toc436935290)

[Listing 179 Wynik operacji 117](#_Toc436935291)

[Listing 180 Wymagane dane do tworzenia zespołu 117](#_Toc436935292)

[Listing 181 Wynik operacji 118](#_Toc436935293)

# Załącznik Dokumentacja API

**Trasa: /api/v1/events/:id**

**Opis:** Trasę wykorzystujemy do aktualizacji wydarzenia**.** Zalogowany użytkownik musi być organizatorem wydarzenia.

**Metoda:** PUT

**Wymagane dane:**

Listing 38 Wymagane dane do aktualizacji wydarzenia.

*{*

*event\_id identyfikator wydarzenia.*

*event:{ (opcjonalne parametry)*

*name: nazwa wydarzenia.*

*description: opis wydarzenia.*

*start\_date: data i czas rozpoczęcia wydarzenia.*

*end\_date: data i czas końca wydarzenia.*

*city\_id: identyfikator miasta, w którym odbywa się wydarzenie.*

*}*

*}*

**Wynik:**

Listing 39 Wynik aktualizacji wydarzenia

*{*

*id: identyfikator wydarzenia,.*

*name: nazwa wydarzenia.*

*description: opis wydarzenia.*

*start\_date: data i czas rozpoczęcia wydarzenia.*

*end\_date: data i czas końca wydarzenia.*

*city*

*{*

*id: identyfikator miasta, w którym odbywa się wydarzenie.*

*name : nazwa miasta, w którym odbywa się wydarzenie.*

*}*

*}*

**Trasa: /api/v1/events/:id**

**Opis:** Trasę wykorzystujemy do usunięcia wydarzenia.

**Metoda:** Delete

**Wymagane dane:**

Listing 40 Wymagane dane do usunięcia wydarzenia

*{*

*event\_id: identyfikator wydarzenia.*

*}*

**Wynik:**

Listing 41 Wynik operacji usunięcia zespołu

*{*

*id: identyfikator wydarzenia.*

*deleted: wynik operacji.*

*}*

**Trasa: /api/v1/events/:id/accept\_band**

**Opis:** Trasę wykorzystujemy do potwierdzenia przez zespół wzięcia udziału w wydarzeniu. Zalogowany użytkownik, musi być w składzie zespołu.

**Metoda:** POST

**Wymagane dane:**

Listing 42 Wymagane dane do potwierdzenia przez zespół wzięcia udziału w wydarzeniu

*{*

*event\_id: identyfikator wydarzenia.*

*band\_id: identyfikator zespołu.*

*}*

**Wynik:**

Listing 43 Wynik operacji potwierdzenia zaproszenia dla zespołu

*{*

*event\_id: identyfikator wydarzenia.*

*accept: wynik operacji.*

*}*

**Trasa: /api/v1/groups/query?name=someValue**

**Opis:** Trasę wykorzystujemy do wyszukiwania grup.

**Metoda**: GET

**Wymagane dane:**

Listing 44 Wymagane dane do wyszukiwania grup

*{*

*name: nazwa grupy.*

*}*

**Wynik:**

Listing 45 Wynk operacji wyszukiwania

*{*

*id: identyfikator grupy.*

*name: nazwa grupy.*

*is\_member: czy użytkownik uczestniczy w grupy.*

*is\_admin: czy użytkownik jest administratorem grupy.*

*avatar\_original: awatar grupy.*

*}*

**Trasa: /api/v1/groups/:group\_id/setAvatar**

**Opis:** Trasę wykorzystujemy do ustawienia awatara dla grupy. Zalogowany użytkownik musi być administratorem grupy.

**Metoda**: PUT

**Wymagane dane:**

Listing 46 Wymagane dane do ustawiania awatarów dla grupy

{

group\_id: identyfikator grupy.

photo:{

id: identyfikator zdjęcia.

}

}

**Wynik:**

Listing 47 Wynik operacji ustawienia awataru dla grupy

*{*

*id: identyfikator grupy.*

*avatar\_original: awatar grupy.*

*}*

**Trasa: /api/v1/groups/:group\_id/add\_me**

**Opis:** Trasę wykorzystujemy do dodania zalogowanego użytkownika do grupy.

**Metoda:** POST

**Wymagane dane:**

Listing 48 Wymagane dane do dodania zalogowanego użytkownika do grupy

*{*

*group\_id: identyfikator grupy,*

*}*

**Wynik:**

Listing 49 Wynik operacji

*{*

*id: identyfikator grupy.*

*users[]:{*

*id: identyfikator użytkownika.*

*name: nazwa użytkownika*

*avatar\_original: avatar użytkownika*

*}*

*}*

**Trasa: /api/v1/groups/:group\_id/delete\_me**

**Opis:** Trasę wykorzystujemy do usunięcia zalogowanego użytkownika z grupy.

**Metoda:** POST

**Wymagane dane:**

Listing 50 Wymagane dane do usununięcia użytkownika grupy

*{*

*group\_id: identyfikator grupy,*

*}*

**Wynik:**

Listing 51 Wynik usunięcia użytkownika z grupy

*{*

*id: identyfikator grupy.*

*users[]:{*

*id: identyfikator użytkownika.*

*name: nazwa użytkownika.*

*avatar\_original: avatar użytkownika.*

*}*

*}*

**Trasa: /api/v1/groups/:group\_id/set\_member**

**Opis:** Trasę wykorzystujemy do nadania/usunięcia praw administratora dla konkretnego użytkownika. Możemy to zrealizować tylko, jeśli użytkownik nadający/usuwający uprawnienia jest administratorem grupy.

**Metoda:** PUT

**Wymagane dane:**

Listing 52 Wymagane dane do nadania praw administratora grupy

*{*

*group\_id: identyfikator grupy,*

*admin: czy użytkownik ma być administratorem.*

*user\_id: identyfikator użytkownika, który jest w grupie.*

*}*

**Wynik:**

Listing 53 Wynik operacji

*{*

*id: identyfikator grupy*

*users[]:{*

*id: identyfikator użytkownika.*

*name: nazwa użytkownika.*

*avatar\_original: awatar użytkownika.*

*is\_admin: czy użytkownik ma być organizatorem.*

*}*

*}*

**Trasa: /api/v1/groups/:group\_id/posts**

**Opis:** Trasę wykorzystujemy do pobrania postów grupy.

**Metoda:** GET

**Wymagane dane:**

Listing 54 Wymagane dane do pobrania postów grupy

*{*

*group\_id: identyfikator grupy.*

*page: paginacja po postach.*

*}*

**Wynik:**

Listing 55 Wynik operacji

*{*

*{id: identyfikator postu.*

*content: treść postu.*

*created\_at: data stworzenia postu.*

*is\_author: czy zalogowany użytkownik jest autorem.*

*comments[]:{*

*id: identyfikator komentarza.*

*content: treść komentarza.*

*created\_at : data stworzenia postu.*

*is\_author: czy zalogowany użytkownik jest autorem komentarza.*

*user:{*

*id: identyfikator użytkownika.*

*name: nazwa użytkownika.*

*avatar\_original: avatar użytkownika.*

*}*

*}*

*user:{*

*id: identyfikator użytkownika.*

*name: nazwa użytkownika.*

*avatar\_original: awatar użytkownika.*

*}*

*}*

*group: {*

*id: identyfikator grupy.*

*name: nazwa grupy.*

*is\_member: czy użytkownik uczestniczy w grupie.*

*is\_admin: czy użytkownik jest administratorem grupy.*

*avatar\_original: awatar grupy.*

*count: ilość postów w grupie.*

*}*

*}*

**Trasa: /api/v1/groups/:group\_id/photos**

**Opis:** Trasę wykorzystujemy do pobrania zdjęć grupy.

**Metoda:** GET

**Wymagane dane:**

Listing 56 Dane wymagane do pobrania zdjęć grupy

*{*

*group\_id: identyfikator grupy.*

*page: paginacja po zdjęciach.*

*}*

**Wynik:**

Listing 57 Wynik operacji

*{*

*{id: identyfikator zdjęcia.*

*description: opis zdjęcia.*

*created\_at: data dodania zdjęcia.*

*photo\_url: adres zdjęcia.*

*group {*

*id: identyfikator grupy.*

*count: ilość zdjęć w grupie.*

*}*

*}*

**Trasa: /api/v1/groups/:group\_id/events**

**Opis:** Trasę wykorzystujemy do pobrania wydarzeń stworzonych w ramach grupy.

**Metoda:** GET

**Wymagane dane:**

Listing 58 Wymagane dane do pobrania wydarzeń grupy

*{*

*group\_id: identyfikator grupy.*

*page: paginacja po wydarzeniach.*

*}*

**Wynik:**

Listing 59 Wynik operacji

*{*

*{*

*id: identyfikator wydarzenia.*

*name: nazwa wydarzenia.*

*start\_date: data i czas rozpoczęcia wydarzenia.*

*end\_date: data i czas końca wydarzenia.*

*city*

*{*

*id: identyfikator miasta w którym odbywa się wydarzenie.*

*name : nazwa miasta w którym odbywa się wydarzenie.*

*}*

*music\_generes[]*

*{*

*id: identyfikator gatunku muzycznego związanego z wydarzeniem.*

*name : nazwa gatunku muzycznego związanego z wydarzeniem.*

*}*

*is\_member: czy użytkownik uczestniczy w wydarzeniu.*

*is\_organizer: czy użytkownik jest organizatorem wydarzenia.*

*group:{*

*id: identyfikator grupy.*

*name: nazwa grupy.*

*is\_member: czy użytkownik uczestniczy w grupy.*

*is\_admin: czy użytkownik jest administratorem grupy.*

*avatar\_original: awatar grupy.*

*count: ilość wydarzeń grupie.*

*}*

*}*

*}*

**Trasa: /api/v1/groups/:group\_id/events**

**Opis:** Trasę wykorzystujemy do tworzenia wydarzenia w ramach grupy

**Metoda:** POST

**Wymagane dane:**

Listing 60 Wymagane dane do tworzenia wydarzenia

*{*

*group\_id: identyfikator grupy.*

*event: {*

*name: nazwa wydarzenia.*

*}*

*}*

**Wynik:**

Listing 61 Wymagane dane

***{***

*id:**identyfikator wydarzenia.*

*name:**nazwa wydarzenia.*

***}***

**Trasa: /api/v1/groups**

**Opis:** Trasę wykorzystujemy do tworzenia grupy.

**Metoda:** POST

**Wymagane dane:**

Listing 62 Wymagane dane do tworzenia grupy

*{*

*group: {*

*name: nazwa grupy.*

*}*

*}*

**Wynik:**

Listing 63 Wynik operacji

***{***

*id:**identyfikator grupy.*

*name:**nazwa grupy.*

***}***

**Trasa: /api/v1/groups/:id**

**Opis:** Trasę wykorzystujemy do pobrania grupy.

**Metoda:** GET

**Wymagane dane:**

Listing 64 Wymagane dane do pobrania grupy

*{*

*id: identyfikator grupy.*

*}*

**Wynik:**

Listing 65 Wynik operacji

*{*

*id: identyfikator grupy.*

*name: nazwa grupy.*

*is\_member: czy użytkownik uczestniczy w grupy.*

*is\_admin: czy użytkownik jest administratorem grupy.*

*avatar\_original: awatar grupy.*

*description: opis grupy.*

*}*

**Trasa: /api/v1/groups/:group\_id**

**Opis:** Trasę wykorzystujemy do aktualizacji grupy. Zalogowany użytkownik musi być administratorem grupy.

**Metoda:** PUT

**Wymagane dane:**

Listing 66 Wymagane dane do aktualizacji grupy

*{*

*group\_id: identyfikator grupy.*

*group:{ (opcjoalne parametry)*

*name: nazwa grupy.*

*description: opis grupy.*

*}*

*}*

**Wynik:**

Listing 67 Wymagane dane

{

*id: identyfikator grupy.*

*name: nazwa grupy.*

*description: opis grupy.*

*}*

**Trasa: /api/v1/groups/:id**

**Opis**: Trasę wykorzystujemy do usunięcia grupy

**Metoda**: Delete

**Wymagane dane**:

Listing 68 Wymagane dane do usunięcia grupy

{

group\_id: identyfikator grupy.

}

**Wynik**:

Listing 69 Wynik operacji

{

id: identyfikator grupy.

deleted: wynik operacji

**}**

**Trasa: /api/v1/conversations/:conversation\_id/addUserToConversation**

**Opis**: Trasę wykorzystujemy do dodania użytkownika do konwersacji .

**Metoda**: PUT

**Wymagane dane**:

Listing 70 Wymagane dane do dodania użytkownika do konwersacji

{

conversation\_id: identyfikator konwersacji.

user{

id: identyfikator użytkownika.

}

}

**Wynik**:

Listing 71 Wynik operacji

{

id: identyfikator konwersacji.

users

{

id: identyfikator użytkownika.

nickname : nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

**}**

**Trasa: /api/v1/conversations/:conversation\_id/deleteUserFromConversation**

**Opis**: Trasę wykorzystujemy do usunięcia użytkownika z konwersacji

**Metoda**: PUT

**Wymagane dane**:

Listing 72 Wymagane dane do usunięcia użytkownika z konwersacji

{

conversation\_id: identyfikator konwersacji.

user{

id: identyfikator użytkownika.

}

}

**Wynik**:

Listing 73 Wymagane dane

{

id: identyfikator konwersacji.

users

{

id: identyfikator użytkownika

nickname : nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

**}**

**Trasa: /api/v1/conversations**

**Opis:** Trasę wykorzystujemy dopobrania wszystkich konwersacji zalogowanego użytkownika.

**Metoda:** GET

**Wymagane parametry:** brak.

**Wynik:**

Listing 74 Wynik operacji pobrania wszystkich konwersacji

{

id: identyfikator konwersacji.

name: nazwa konwersacji.

created\_at: data rozpoczęcia konwersacji.

users[]:

{

id: identyfikator użytkownika.

nickname : nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

}

**Trasa: /api/v1/conversations**

**Opis:** Trasę wykorzystujemy do tworzenia konwersacji

**Metoda:** POST

**Wymagane parametry:**

Listing 75 Wymagane dane do tworzenia konwersacji

{

name; nazwa konwersacji.

users[]: {

id: identyfikator użytkownika.

}

}

**Wynik:**

Listing 76 Wynik operacji

{

id: identyfikator konwersacji.

name: nazwa konwersacji.

created\_at: data rozpoczęcia konwersacji.

users[]:

{

id: identyfikator użytkownika.

nickname : nazwa użytkownika.

awatar\_original: awatar użytkownika.

}

}

**Trasa: /api/v1/conversations/:id**

**Opis:** Trasę wykorzystujemy do pobrania konwersacji.

**Metoda:** POST

**Wymagane parametry:**

Listing 77 Wymagane dane do pobrania konwersacji

{

conversation\_id: identyfikator konwersacji.

}

**Wynik:**

Listing 78 Wynik operacji

{

id: identyfikator konwersacji.

name: nazwa konwersacji.

created\_at: data rozpoczęcia konwersacji.

users[]:

{

id: identyfikator użytkownika.

nickname : nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

message[]:{

id: identyfikator wiadomości.

conent: treść wiadomości.

created\_at: data rozpoczęcia wiadomości.

user

{

id: identyfikator użytkownika.

nickname : nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

}

}

**Trasa: /api/v1/conversations/:id**

**Operacja:** Trasę wykorzystujemy do aktualizacji nazwy konwersacji

**Metoda:** PUT

**Wymagane parametry:**

Listing 79 Wymagane dane do aktualizacji nazwy konwersacji

{

conversation\_id: identyfikator konwersacji.

name: nazwa konwersacji.

}

**Wynik:**

Listing 80 Wynik operacji

{

id: identyfikator konwersacji.

name: nazwa konwersacji.

}

**Trasa: /api/v1/photos/:photo\_id/download**

**Opis:** Trasę wykorzystujemy dopobrania miniaturki.

**Metoda:** GET

**Wymagane dane:**

Listing 81 Wymagane dane do pobrania miniaturki zdjęcia

{

photo\_id: identyfikator zdjęcia.

}

**Wynik**:

Listing 82 Wynik operacji

{

adres: adres zdjęcia.

}

**Trasa: /api/v1/photos/:photo\_id/download\_org**

**Opis:** Trasę wykorzystujemy do pobrania oryginału zdjęcia.

**Metoda:** GET

**Wymagane dane:**

Listing 83 Wymagane dane do pobrania oryginału zdjęcia

{

photo\_id: identyfikator

}

**Wynik**:

Listing 84 Wynik operacji

{

adres: adres zdjęcia

}

**Trasa: /api/v1/photos/:photo\_id/comments**

**Opis:** Trasę wykorzystujemy do dodania komentarza do zdjęcia.

**Metoda:** POST

**Wymagane dane:**

Listing 85 Wymagane dane do dodania komentarza do zdjęcia

{

photo\_id: identyfikator zdjęcia.

comment: {

content: treść komentarza.

}

}

**Wynik**:

Listing 86 Wynik operacji

{

comments[]:{

id: id komentarza.

conent: treść komentarza.

created\_at: data dodania komentarza.

user:{

id: identyfikator użytkownika.

name: nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

}

}

**Trasa: /api/v1/photos**

**Opis:** Trasę wykorzystujemy do tworzenia zdjęcia. Użytkownik musi być właścicielem profilu lub być w składzie zespołu lub być uczestnikiem grupy.

**Metoda:** POST

**Wymagane dane:**

Listing 87 Wymagane dane do tworzenia zdjęć

{

description: opis zdjęcia.

group\_id/user\_id/band\_id: identyfikator dodania przez użytkownika.

photo: plik ze zdjęciem.

}

**Wynik:**

Listing 88 Wynik operacji

{

photo\_id: id zdjęcia.

photo\_url: adres zdjęcia.

created\_at: data dodania.

}

**Trasa: /api/v1/photos/:id**

**Opis:** Trasę wykorzystujemy do pobrania zdjęcia.

**Metoda:** GET

**Wymagane dane:**

Listing 89 Wymagane dany do pobrania zdjęć

{

photo\_id: identyfikator zdjęcia.

}

**Wynik:**

Listing 90 Wynik operacji

{

id identyfikator zdjęcia.

description opis zdjęcia.

photo\_url adres zdjęcia.

comments:{

id: identyfikator komentarza.

content: treść komentarza.

created\_at : data stworzenia komentarza.

is\_author: czy użytkownik jest autorem komentarza.

user:{

id: identyfikator użytkownika.

name: nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

}

//W zależności od tego kto dodał zdjecie (zespół, grupa, użytkownik)

user

{

id: identyfikator użytkownika.

nickname : nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

group

{

id: identyfikator grupy.

name: nazwa grupy.

awatar\_original: awatar grupy.

}

band

{

id: identyfikator zespołu.

name : nazwa zespołu.

avatar\_original: awatar zespołu.

}

}

**Trasa: /api/v1/photos/:id**

**Opis:** Trasę wykorzystujemy do aktualizacji treści zdjęcia.

**Metoda:** PUT

**Wymagane parametry:**

Listing 91 Wymagane dane do aktualizacji opisu zdjęcia

{

photo\_id: identyfikator zdjęcia.

photo:{

description: opis zdjęcia.

}

}

**Wynik:**

Listing 92 Wynik operacji

{

photo\_id: identyfikator zdjęcia.

description: opis zdjęcia.

}

**Trasa: /api/v1/photos/:id**

**Opis:** Trasę wykorzystujemy do usunięcia zdjęcia.

**Metoda:** DELETE

**Wymagane dane**:

Listing 93 Wymagane dane do usunięcia zdjęcia

{

photo\_id: identyfikator zdjęcia.

}

**Wynik**:

Listing 94 Wynik operacji

{

photo\_id: identyfikator .

deleted: wynik operacji

}

**Trasa: /api/v1/users/query?nickname=someValue&plays[]=someInstrument&cities[]=someCity**

**Opis:** Trasę wykorzystujemy do wyszukiwania użytkowników.

**Metoda:** GET

**Wymagane dane:**

Listing 95 Opcjonalne dane do wyszukiwania użytkowników

{

nickname: nazwa użytkownika.

plays[]: tablica z nazwami instrumentów.

cities[]: tablica z nazwami miast .

page: paginacja po wynikach.

}

**Wynik:**

Listing 96 Wynik operacji

{

id: identyfikator użytkownika.

name: nazwa użytkownika.

avatar\_original: awatar użytkownika.

user\_instruments[]:{

id: identyfikator instrumentu

name : nazwa instrumentu

}

city: {

id: identyfikator miasta

name: nazwa miasta

}

}

**Trasa: /api/v1/users/query\_for\_band?nickname=someValue&plays[]=someGenere&cities[]=someCity**

**Opis:** Trasę wykorzystujemy do wyszukiwania użytkowników do zespołów

**Metoda:** GET

**Wymagane dane:**

Listing 97 Opcjonalne dane do wyszukiwania użytkowników do zespołu

{

nickname: nazwa użytkownika,

plays[]: tablica z nazwami instrumentów.

cities[]: tablica z nazwami miast .

page: paginacja po wynikach.

}

**Wynik:**

Listing 98 Wynik operacji

{

id: identyfikator użytkownika.

nickname: nazwa użytkownika.

avatar\_original: awatar użytkownika.

user\_instruments[]:{

id: identyfikator instrumentu.

name : nazwa instrumentu.

}

city: {

id: identyfikator miasta.

name: nazwa miasta.

}

about: opis użytkownika.

}

**Trasa: /api/v1/users/:user\_id/setAvatar**

**Opis**: Trasę wykorzystujemy do ustawienia awatara dla użytkownika .

**Metoda**: PUT

**Wymagane dane:**

Listing 99 Wymagane dane do ustawienia awataru dla użytkownika

{

user\_id: identyfikator użytkownika .

photo:{

id: identyfikator zdjęcia.

}

}

**Wynik:**

Listing 100 Wynik operacji

{

id: identyfikator użytkownika.

avatar\_original: awatar użytkownika.

}

**Trasa: /api/v1/users/:user\_id/events**

**Opis: T**rasę wykorzystujemy do tworzenia wydarzenia użytkownika.

**Metoda:** POST

**Wymagane dane:**

Listing 101 Wymagane dane do tworzenia wydarzenia użytkownika

{

user\_id: identyfikator użytkownika.

event: {

name: nazwa wydarzenia.

}

}

**Wynik:**

Listing 102 Wymagane dane

{

id:identyfikator wydarzenia

name:nazwa wydarzenia

}

**Trasa: /api/v1/users/:user\_id/setAddonationalInfo**

**Opis:** Trasę wykorzystujemy do aktualizacji danych użytkownika.

**Metoda:** PUT

**Wymagane dane:**

Listing 103 Wymagane dane do aktualizacji danych użytkownika

{

user\_id: identyfikator użytkownika.

user:{ (opcjonalne parametry)

about informacje o użytkowniku.

birthday data urodzin.

my\_history historia użytkownika.

search\_for\_band: czy poszukuje zespołu.

my\_bands: dotychczasowa kariera użytkownika.

city\_id : identyfikator miasta.

}

}

**Wynik:**

Listing 104 Wynik operacji

{

user\_id: identyfikator użytkownika.

user:{

about informacje o użytkowniku.

birthday data urodzin.

my\_history historia użytkownika.

search\_for\_band: czy poszukuje zespołu.

my\_bands: dotychczasowa kariera użytkownika.

city: {

id: identyfikator miasta.

name: nazwa miasta.

}

}

}

**Trasa: /api/v1/users/:user\_id/plays**

**Metoda:** PUT

**Wymagane dane:**

Listing 105 Wymagane dane do aktualizacji instrumentów użytkownika

{

user\_id: identyfikator użytkownika,

instrument{

toAppend[]:{

id: identyfikator instrumentu do dodania.

}

toDelete[]:{

id: identyfikator instrumentu do usunięcia.

}

}

}

**Wyniki:**

Listing 106 Wynik operacji

{

id: identyfikator użytkownika.

play;{

id: identyfikator instrumentu.

name: nazwa instrumentu.

}

}

**Trasa: /api/v1/users/:user\_id/posts**

**Opis:** Trasę wykorzystujemy do pobrania postów użytkownika.

**Metoda:** GET

**Wymagane dane:**

Listing 107 Wymagane dane do pobrania postów użytkownika

{

user\_id: identyfikator użytkownika

page: paginacja po postach

}

**Wynik:**

Listing 108 Wynik operacji

{

{id: identyfikator postu.

content: treść posta.

created\_at: data stworzenia postu.

is\_author: czy użytkownik jest autorem.

comments[]:{

id: identyfikator komentarza.

content: treść komentarza.

created\_at : data stworzenia komentarza.

is\_author: czy użytkownik jest autorem komentarza.

user:{

id: identyfikator użytkownika.

name: nazwa użytkownika.

avatar\_original: avatar użytkownika.

}

}

user:{

id: identyfikator użytkownika.

name: nazwa użytkownika.

avatar\_original: avatar użytkownika.

}

}

**Trasa: /api/v1/users/:user\_id/buddyships**

**Opis:** Trasę wykorzystujemy do pobrania znajomych użytkownika.

**Metoda:** GET

**Wymagane dane:**

Listing 109 Wymagande dane do pobrania znajomych użytkownika

{

user\_id: identyfikator grupy

}

**Wynik**:

Listing 110 Wynik operacji

{

id: znajomości

buddy[]:{

id: identyfikator użytkownika.

nickname: nazwa użytkownika.

}

}

**Trasa: /api/v1/users/:user\_id/photos**

**Opis:** Trasę wykorzystujemy do pobrania zdjęć użytkownika

**Metoda:** GET

**Wymagane dane:**

Listing 111 Wymagane dane do pobrania zdjęć użytkownika

{

user\_id: identyfikator użytkownika.

page: paginacja po zdjęciach.

}

**Wynik:**

Listing 112 Wynik operacji

{

{id: identyfikator zdjęcia.

description: opis zdjęcia.

created\_at: data dodania zdjęcia.

photo\_url: adres zdjęcia.

user{

id: identyfikator użytkownika.

count: ilość zdjęć użytkownika.

}

}

**Trasa: /api/v1/users/:user\_id/bands**

**Opis:** Trasę wykorzystujemy dopobrania zespołów użytkownika.

**Metoda:** GET

**Wymagane dane:**

Listing 113 Wymagande dane do pobrania zespołów użytkownika

{

user\_id: identyfikator użytkownika.

page: paginacja po zdjęciach.

}

**Wynik:**

Listing 114 Wynik operacji

{

bands[]:

{

id: identyfikator zespołu

name : nazwa zespołu

avatar\_original: awatar zespołu

}

}

**Trasa: /api/v1/users/:user\_id/events**

**Opis:** Trasę wykorzystujemy do pobrania wydarzeń użytkownika.

**Metoda:** GET

**Wymagane dane:**

Listing 115 Wymagane dane do pobrania wydarzeń użytkownika

{

user\_id: identyfikator użytkownika.

page: paginacja po wydarzeniach.

}

**Wynik:**

Listing 116 Wynik operacji

{

{

id: identyfikator wydarzenia.

name: nazwa wydarzenia.

start\_date: data i czas rozpoczęcia wydarzenia.

end\_date: data i czas końca wydarzenia.

city

{

id: identyfikator miasta, w którym odbywa się wydarzenie.

name : nazwa miasta, w którym odbywa się wydarzenie.

}

music\_generes[]

{

id: identyfikator gatunku muzycznego związanego z wydarzeniem.

name : nazwa gatunku muzycznego związanego z wydarzeniem.

}

is\_member: czy użytkownik uczestniczy w wydarzeniu.

is\_organizer: czy użytkownik jest organizatorem wydarzenia.

user:{

id: identyfikator użytkownika.

nickname: nazwa użytkownika.

avatar\_original: awatar użytkownika.

count: ilość wydarzeń użytkownika.

}

}

}

**Trasa: /api/v1/users/:id**

**Opis:** Trasę wykorzystujemy do pobrania użytkownika

**Metoda:** GET

**Wymagane dane:**

Listing 117 Wymagane dane do pobrania użytkownika

**{**

user\_id**:** identyfikator użytkownika.

**}**

**Wynik:**

Listing 118 Wynik operacji

users[]

{

id: identyfikator użytkownika.

nickname : nazwa użytkownika.

avatar\_original: awatar użytkownika.

user\_instruments:[]{

id: identyfikator instrumentu.

name : nazwa instrumentu.

}

city: {

id: identyfikator miasta.

name: nazwa miasta.

}

is\_friends: czy użytkownik jest znajomym użytkownika.

is\_me: czy profil należy do użytkownika.

about informacje o użytkowniku

birthday data urodzin

my\_history historia użytkownika

search\_for\_band: czy poszukuje zespołu

my\_bands: dotychczasowa kariera użytkownika

}

**Trasa: /api/v1/posts/:post\_id/comments**

**Opis:** Trasę wykorzystujemy do dodania komentarza do postu.

**Metoda:** POST

**Wymagane dane:**

Listing 119 Wymagane dane do dodania komentarza do postu

{

post\_id: identyfikator postu

comment{

content: treść komentarza

}

}

**Wynik**:

Listing 119 Wynik operacji

{

comments[]:{

id: id komentarza

content: treść komentarza

created\_at: data dodania komentarza

user:{

id: identyfikator użytkownika

name: nazwa użytkownika

avatar\_original: avatar użytkownika

}

}

}

**Trasa: /api/v1/posts/get\_main\_site**

**Opis:** Trasę wykorzystujemy do pobranie postów, które może zalogowany użytkownik zobaczyćna stronie głównej**.**

**Metoda:** GET

**Wymagane dane:**

Listing 120 Wymagane dane do pobrania postów na stronę główną

{

page: paginacja, pobiera po 10 obiektów z x strony.

}

**Wynik:**

Listing 121 Wynik operacji

{

{id: identyfikator postu.

content: treść posta.

created\_at: data stworzenia postu.

is\_author: czy użytkownik jest autorem postu.

comments[]:{

id: identyfikator komentarza.

content: treść komentarza.

created\_at : data stworzenia postu.

is\_author: czy użytkownik jest autorem komentarza.

user:{

id: identyfikator użytkownika.

nickname: nazwa użytkownika

avatar\_original: avatar użytkownika.

}

}

W zależności od tego kto i gdzie dodał post

user:{

id: identyfikator użytkownika.

nickname: nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

group:{

id: identyfikator grupy.

name: nazwa grupy.

avatar\_original: awatar grupy.

}

band:{

id: identyfikator zespołu.

name: nazwa zespołu.

avatar\_original: awatar zespołu.

}

}

**Trasa: /api/v1/buddyships/accept**

**Opis:** Trasę wykorzystujemy do zaakceptowania znajomości.

**Metoda:** POST

**Wymagane dane:**

Listing 122 Wymagande dane do zaakceptowania zaproszenia do znajomych

{

buddy\_id: identyfikator użytkownika.

}

**Wynik:**

Listing 123 Wynik operacji

**{**

accept: wynik operacji.

**}**

**Trasa: /api/v1/buddyships/notaccepted**

**Opis:** Trasę wykorzystujemy dopobrania niezaakceptowanych znajomości zalogowanego użytkownika.

**Metoda:** GET

**Wymagane dane:**

Listing 124 Wymagane dane do pobrania niezaakceptowanych znajomości

{

user\_id: identyfikator użytkownika.

}

**Wynik:**

Listing 125 Wynik operacji

**{**

id: identyfikator użytkownika.

nickname : nazwa użytkownika.

avatar\_original: awatar użytkownika.

**}**

**Trasa: /api/v1/buddyships/requested**

**Opis:** Trasę wykorzystujemy do wysłania zaproszenia do znajomych

**Metoda:** POST

**Wymagane dane:**

Listing 126 Wymagane dane do wysłania zaproszenia do znajomych

{

buddy\_id: identyfikator użytkownika

}

**Wynik:**

Listing 127 Wynik operacji

**{**

Send: true

**}**

**Trasa: /api/v1/buddyships/:id**

**Opis:** Trasę wykorzystujemy do usuwania znajomości zalogowanego użytkownika..

**Metoda:** DELETE

**Wymagane dane:**

Listing 128 Wymagane dane do usuwania znajomości

{

buddy\_id: identyfikator użytkownika.

}

**Wynik:**

Listing 129 Wynik operacji

{

deleted: powodzenie operacji.

}

**Trasa: /api/v1/bands/:band\_id/set\_music\_generes Metoda: POST**

**Opis:** Trasę wykorzystujemy do zmiany gatunków muzycznych związanych z zespołem

**Metoda:** POST

**Wymagane dane:**

Listing 130 Wymagane dane do zmiany gatunków muzycznych

{

band\_id: identyfikator zespołu,

toAppend[]: {

id: identyfikator gatunku muzycznego do dodania

}

toDelete[]:{

id: identyfikator gatunku muzycznego do usunięcia

}

}

**Wynik:**

Listing 131 Wynik operacji

{

id: identyfikator zespołu.

music\_generes[]:{

id: identyfikator gatunku muzycznego.

name : nazwa gatunku muzycznego.

}

}

**Trasa: /api/v1/bands/:band\_id/fallows**

**Opis:** Trasę wykorzystujemy pobrania użytkowników, którzy obserwują zespół

**Metoda:** GET

**Wymagane dane:**

Listing 132 Wymagane dane do pobrania użytkowników, którzy obserwują zespół

{

band\_id: identyfikator zespołu,

}

**Wynik:**

Listing 133 Wynik operacji

{

id: identyfikator użytkownika.

nickname : nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

**Trasa: /api/v1/bands/:band\_id/setAvatar**

**Opis:** Trasę wykorzystujemy do ustawienia awatara dla zespołu. Zalogowany użytkownik, musi być w składzie zespołu.

**Metoda**: PUT

**Wymagane dane:**

Listing 134 Wymagane dane do ustawienia awatara dla zespołu

{

band\_id: identyfikator zespołu.

photo:{

id: identyfikator zdjęcia.

}

}

**Wynik:**

Listing 135 Wynik operacji

{

id: identyfikator zespołu.

avatar\_original: awatar zespołu.

}

**Trasa: /api/v1/bands/:band\_id/add\_member**

**Opis:** Trasę wykorzystujemy do dodanie użytkownika do składu zespołu.

**Metoda:** POST

**Wymagane dane:**

Listing 136 Wymagane dane do dodania użytkownika do składu zespołu

{

user\_id: identyfikator użytkownika,.

band\_id: identyfikator zespołu.

}

**Wynik:**

Listing 137 Wynik operacji

{

id: identyfikator wydarzenia.

users[]:{

id: identyfikator użytkownika.

nickname : nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

}

**Trasa:** /api/v1/bands/:band\_id/add\_fallower

**Opis:** Trasę wykorzystujemy do dodanie zalogowanego użytkownika do zespołu jako obserwatora.

**Metoda:** POST

**Wymagane dane:**

Listing 138 Wymagane dane do rozpoczęcia obserwacji zespołu

{

band\_id: identyfikator zespołu.

}

**Wynik:**

Listing 139 Wynik operacji

{

id: identyfikator wydarzenia.

users[]{

id: identyfikator użytkownika.

nickname : nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

}

**Trasa: /api/v1/bands/:band\_id/delete\_member**

**Opis:** Trasę wykorzystujemy do usunięcia użytkownika ze składu zespołu.

**Metoda:** POST

**Wymagane dane:**

Listing 140 Wymagane dane do usunięcia użytkownika ze składu zespołu

{

user\_id: identyfikator użytkownika.

band\_id: identyfikator zespołu.

}

**Wynik:**

Listing 141 Wynik operacji

{

id: identyfikator zespołu.

users[]{

id: identyfikator użytkownika.

nickname : nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

}

**Trasa: /api/v1/bands/:band\_id/delete\_fallower**

**Opis:** Trasę wykorzystujemy do usunięcia zalogowanego użytkownika z zespołu jako obserwatora

**Metoda:** POST

**Wymagane dane:**

Listing 142 Wymagane do zaprzestania obserwacji zespołu

{

band\_id: identyfikator zespołu

}

**Wynik:**

Listing 143 Wynik operacji

{

id: identyfikator wydarzenia

users[]{

id: identyfikator użytkownika

nickname : nazwa użytkownika

avatar\_original: awatar użytkownika

}

}

**Trasa: /api/v1/bands/:band\_id/posts**

**Opis:** Trasę wykorzystujemy do pobrania postów zespołu.

**Metoda:** GET

**Wymagane dane:**

Listing 144 Wymagane dane do pobrania postów zespołu

{

band\_id: identyfikator zespołu.

page: paginacja po postach.

}

**Wynik:**

Listing 145 Wynik operacji

{

{id: identyfikator postu.

content: treść posta.

created\_at: data stworzenia postu.

is\_author: czy użytkownik jest autorem.

comments[]:{

id: identyfikator komentarza.

content: treść komentarza.

created\_at : data stworzenia postu.

is\_author: czy użytkownik jest autorem komentarza.

user:{

id: identyfikator użytkownika.

name: nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

}

user(twórcy):{

id: identyfikator użytkownika.

name: nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

}

band{

id: identyfikator zespołu.

name: nazwa zespołu.

is\_member: czy użytkownik uczestniczy zespołu.

avatar\_original: awatar zespołu.

count: ilość postów w zespole.

}

}

**Trasa: /api/v1/bands/:band\_id/photos**

**Opis:** Trasę wykorzystujemy do pobrania zdjęć zespołu.

**Metoda:** GET

**Wymagane dane:**

Listing 146 Wymagane dane do pobrania zdjęć zespołu

{

band\_id: identyfikator zespołu.

page: paginacja po zdjęciach.

}

**Wynik:**

Listing 147 Wynik operacji

{

{id: identyfikator zdjęcia.

description: opis zdjęcia.

created\_at: data dodania zdjęcia.

photo\_url: adres zdjęcia.

band{

id: identyfikator zepsołu.

count: ilość zdjęć w zepsołu.

}

}

**Trasa: /api/v1/bands/:band\_id/events**

**Opis:** Trasę wykorzystujemy do tworzenia wydarzenia zespołu.

**Metoda:** POST

**Wymagane dane:**

Listing 148 Wymagane dane do utworzenia wydarzenia zespołu

{

band\_id: identyfikator zespołu.

event: {

name: nazwa wydarzenia.

}

}

**Wynik:**

Listing 149 Wynik operacji

**{**

id**:** identyfikator wydarzenia.

name**:** nazwa wydarzenia.

**}**

**Trasa: /api/v1/bands/:band\_id/events**

**Opis:** Trasę wykorzystujemy do pobrania wydarzeń zespołu.

**Metoda:** GET

**Wymagane dane:**

Listing 150 Wymagane dane do pobrana wydarzeń zespołu

{

band\_id: identyfikator zespołu.

page: paginacja po wydarzeniach.

}

**Wynik:**

Listing 151 Wynik operacji

{

{

id: identyfikator wydarzenia.

name: nazwa wydarzenia.

start\_date: data i czas rozpoczęcia wydarzenia.

end\_date: data i czas końca wydarzenia.

city

{

id: identyfikator miasta, w którym odbywa się wydarzenie.

name : nazwa miasta, w którym odbywa się wydarzenie.

}

music\_generes[]:

{

id: identyfikator gatunku muzycznego związanego z wydarzeniem.

name : nazwa gatunku muzycznego związanego z wydarzeniem.

}

is\_member: czy użytkownik uczestniczy w wydarzeniu.

is\_organizer: czy użytkownik jest organizatorem wydarzenia.

band:{

id: identyfikator zespołu.

name: nazwa zespołu.

is\_member: czy użytkownik uczestniczy w wydarzeniu.

avatar\_original: awatar zespołu.

count: ilość wydarzeń zespołu.

}

}

}

**Trasa: /api/v1/bands**

**Opis:** Trasę wykorzystujemy do tworzenia zespołu.

**Metoda:** POST

**Wymagane dane:**

Listing 152 Wymagane dane do tworzenia zespołu

{

band: {

name: nazwa zespołu.

}

}

**Wynik:**

Listing 153 Wynik operacji

{

id: identyfikator zespołu.

name: nazwa zespołu.

}

**Trasa: /api/v1/bands/:id**

**Opis:** Trasę tą wykorzystujemy do aktualizacji informacji o zespole. Zalogowany użytkownik musi być w składzie zespołu.

**Metoda:** PUT

**Wymagane parametry**:

Listing 154 Wymagane dane do aktualizacji informacji o zespole

{

band\_id: identyfikator zespołu

band:{ (Opcjonalne parametry)

name -nazwa zespołu.

history - historia zespołu.

members - opis członków zespołu.

start\_date - data rozpoczęcia działalności.

description - opis zespołu.

city\_id: identyfikator miasta.

}

}

**Wynik**:

Listing 155 Wynik operacji

{

id - identyfikator zespołu.

name -nazwa zespołu.

history - historia zespołu.

members - opis członków zespołu.

start\_date - data rozpoczęcia działalności.

description - opis zespołu.

city {

id: identyfikator miasta.

name : nazwa miasta.

}

}

**Trasa: /api/v1/bands/:id**

**Opis**: Trasę wykorzystujemy do usuwania zespołu. Użytkownik musi być w składzie zespołu.

**Metoda:** DELETE

**Wymagane dane**:

Listing 156 Wymagane dane do usunięcia zespołu

{

band\_id: identyfikator zespołu.

}

**Wynik**:

Listing 157 Wynik operacji

{

id: identyfikator zespołu.

deleted: wynik operacji.

}

**Trasa: /api/v1/band/:band\_id/post**

**Opis:** Trasę wykorzystujemy do tworzenia postu na stronie zespołu. Zalogowany użytkownik musi być w składzie zespołu.

**Metoda:** POST

**Wymagane dane:**

Listing 158 Wymagane dane do tworzenia postu na tronie zespołu

{

band\_id: identyfikator zespołu.

post:{

content: treść posta .

}

}

**Wynik:**

Listing 159 Wynik operacji

{

{

id: id posta.

content: treść posta.

created\_at: data stworzenia posta.

is\_author: czy obecny użytkownik jest autorem postu.

user (twórca posta): {

id: id użytkownika

nickname: ksywka użytkownika.

avatar\_original: adres awatara.

}

band: {

id: zespołu.

name; nazwa zespołu.

is\_member: czy użytkownik jest członkiem zespołu.

count: ilość wszystkich postów.

avatar\_original: adres awatara.

}

}

}

**Trasa: /api/v1/groups/:group\_id/posts**

**Opis:** Trasę wykorzystujemy do tworzenia postu dla konkretnej grupy. Zalogowany użytkownik musi należeć do grupy.

**Metoda:** POST

**Wymagane dane:**

Listing 160 Wymagane dane do tworzenia postu dla konkretnej grupy

{

group\_id: identyfikator grupy.

post:{

content: treść postu.

}

}

**Wynik:**

Listing 161 Wynik operacji

{

{

id: identyfikator postu.

content: treść postu.

created\_at: data stworzenia postu.

is\_author: czy obecny użytkownik jest autorem postu.

user (twórca posta): {

id: id użytkownika.

nickname: ksywka użytkownika.

avatar\_original: adres awatara użytkownika.

}

group: {

id: identyfikator grupy.

name; nazwa grupy.

is\_member: czy użytkownik jest członkiem grupy.

is\_admin: czy użytkownik jest administratorem grupy.

count: ilość wszystkich postów.

avatar\_original: adres awatara grupy.

}

}

}

**Trasa: /api/v1/posts/:post\_id**

**Opis:** Trasę wykorzystujemy do usuwania postu, jeśli użytkownik ma do tego odpowiednie uprawnienia.

**Metoda:** DELETE

**Wymagane dane:**

Listing 162 Wymagane dane do usuwania postu

{

post\_id: identyfikator postu

}

**Wynik:**

Listing 163 Wynik operacji

{

deleted: powodzenia operacji

}

**Trasa: /api/v1/posts/:post\_id**

**Opis:** Trasę wykorzystujemy do aktualizacji postu, jeśli zalogowany użytkownik ma do tego odpowiednie uprawnienia

**Metoda:** PUT

**Wymagane dane:**

Listing 164 Wymagane dane do aktualizacji postu

{

post\_id: identyfikator postu

post{

content: treść postu

}

}

**Wynik:**

Listing 165 Wynik operacji

{

id: identyfikator postu

content: zaktualizowana treść postu

}

**Trasa: /api/v1/utils/get\_music\_generes**

Opis: Trasę wykorzystujemy do pobrania wszystkich gatunków muzycznych.

**Metoda:** GET

**Wymagane dane:** brak

**Wynik:**

Listing 166 Wynik operacji

{

{

id: id gatunku

name: nazwa gatunku muzycznego np. “rock”

}

...

}

**Trasa: /api/v1/utils/get\_user\_instrument**

Opis: Trasę wykorzystujemy do pobrania wszystkich instrumentów muzycznych.

**Metoda:** GET

**Wymagane dane:** brak

**Wynik:**

Listing 167 Wynik operacji

{

{

id: identyfikator instrumentu muzycznego.

name: nazwa instrumentu muzycznego np. gitara elektryczna.

}

...

}

**Trasa: /api/v1/events/query?name=someValue&plays[]=someGenere&cities[]=someCity**

**Opis:** Trasę wykorzystujemy do wyszukiwania wydarzeń.

**Metoda**: GET

**Wymagane dane:**

Listing 168 Wymagane dane do wyszukiwania wydarzeń

{

name: nazwa wydarzenia.

plays[]: tablica z nazwami gatunków muzycznych np. “Rock”.

cities[]: tablica z nazwami miast.

page: paginacja po wynikach.

}

**Wynik:**

Listing 169 Wynik opercji

{

{id: identyfikator wydarzenia.

name: nazwa wydarzenia.

music\_generes[]:{

id: identyfikator gatunku muzycznego.

name : nazwa gatunku muzycznego.

}

city: {

id: identyfikator miasta.

name: nazwa miasta.

}

user(twórca wydarzenia): {

id: identyfikator wydarzenia.

nickname: nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

lub

band(twórca wydarzenia): {

id: identyfikator wydarzenia.

name: nazwa zespołu.

avatar\_original: awatar zespołu.

}

lub

group(twórca wydarzenia): {

id: identyfikator wydarzenia.

name: nazwa grupy.

avatar\_original: awatar grupy.

}

is\_member: czy użytkownik uczestniczy w wydarzeniu.

is\_admin: czy użytkownik jest organizatorem wydarzenia.

}

}

**Trasa: /api/v1/events/:event\_id/add\_band**

**Opis:** Trasę wykorzystujemy do dodania zespołu do wydarzenia.

**Metoda:** POST

**Wymagane dane:**

Listing 170 Wymagane dane do dodania zespołu do wydarzenia

{

event\_id: identyfikator wydarzenia.

band\_id: identyfikator zespołu.

}

**Wynik:**

Listing 171 Wynik operacji

{

id: identyfikator wydarzenia.

bands[]: {

id: identyfikator zespołu.

name: nazwa zespołu.

avatar\_original: awatar zespołu.

}

}

**Trasa: /api/v1/events/:event\_id/delete\_band**

**Opis:** Trasę wykorzystujemy do usunięcia zespołu z wydarzenia.

**Metoda:** POST

**Wymagane dane:**

Listing 172 Wymagane dane do usnięcia zespołu z wydarzenia

{

event\_id: identyfikator wydarzenia.

band\_id: identyfikator zespołu.

}

**Wynik:**

Listing 173 Wynik operacji

{

id: identyfikator wydarzenia

bands[]: {

id: identyfikator zespołu

name: nazwa zespołu

avatar\_original: awatar zespołu

}

}

**Trasa: /api/v1/events/:event\_id/add\_me**

**Opis:** Trasę wykorzystujemy do dodania zalogowanego użytkownika do wydarzenia.

**Metoda:** POST

**Wymagane dane:**

Listing 174 Wymagane dane do dodania użytkownika do wydarzenia

{

event\_id: identyfikator wydarzenia.

}

**Wynik:**

Listing 175 Wynik operacji

{

id: identyfikator wydarzenia.

users[]:{

id: identyfikator użytkownika.

name: nazwa użytkownika.

avatar\_original: awatarużytkownika.

}}

**Trasa: /api/v1/events/:event\_id/delete\_me**

**Opis:** Trasę wykorzystujemy do usunięcie zalogowanego użytkownika z wydarzenia.

**Metoda:** POST

**Wymagane dane:**

Listing 176 Wymagane dane do usunięcia użytkownika z wydarzenia

{

event\_id: identyfikator wydarzenia.

}

**Wynik:**

Listing 177 Wynik operacji

{

id: identyfikator wydarzenia.

users[]:{

id: identyfikator użytkownika.

name: nazwa użytkownika.

avatar\_original: awatar użytkownika.

}

}

**Trasa: /api/v1/comments/:id**

**Opis:** Trasę wykorzystujemy doaktualizacji komentarza.

**Metoda:** PUT

**Wymagane dane:**

Listing 178 Wymagane dane do aktualizacji komentarza

{

comment:{

id: identyfikator komentarza.

content: Treść zaktualizowanego komentarza.

}

}

**Wynik:**

Listing 179 Wynik operacji

{

id: identyfikator komentarza.

content: Treść zaktualizowanego komentarza.

}

**Trasa: /api/v1/events**

**Opis:** Trasę wykorzystujemy do tworzenia wydarzenia.

**Metoda:** POST

**Wymagane dane:**

Listing 180 Wymagane dane do tworzenia zespołu

{

event: {

name: nazwa wydarzenia.

}

}

**Wynik:**

Listing 181 Wynik operacji

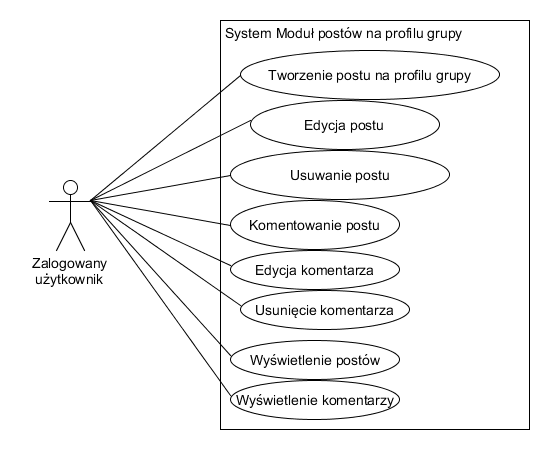
{

id: identyfikator wydarzenia.

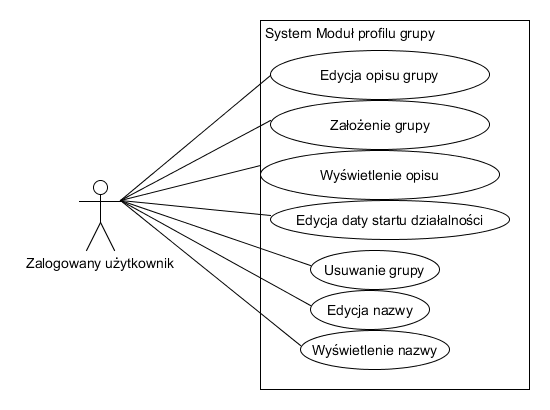
name: nazwa wydarzenia.

}

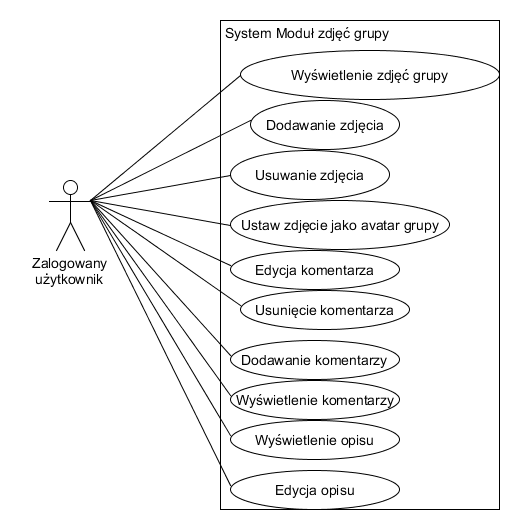
# Załącznik Pozostałe Diagramy przypadków użycia



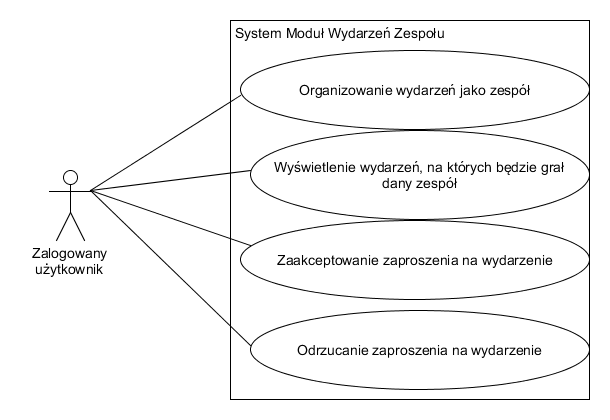
Rysunek B.1 Moduł postów na profilu grupy



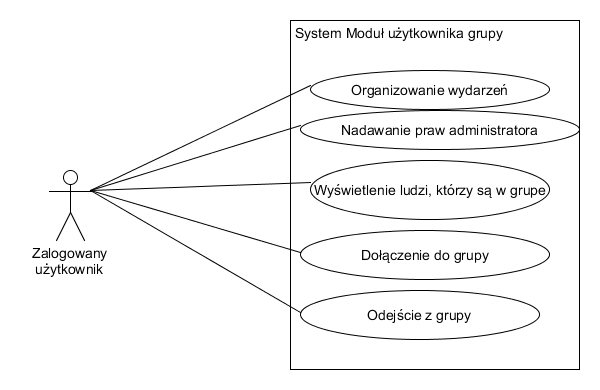
Rysunek B.2 Moduł profilu grupy



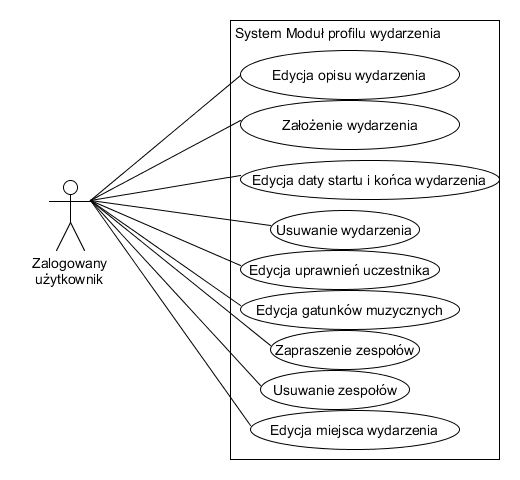
Rysunek B.3 Moduł zdjęć grupy



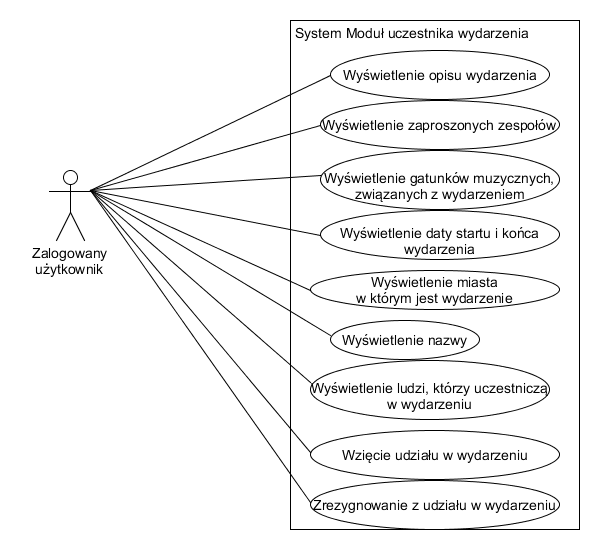
Rysunek B.4 Moduł wydarzeń zespołu



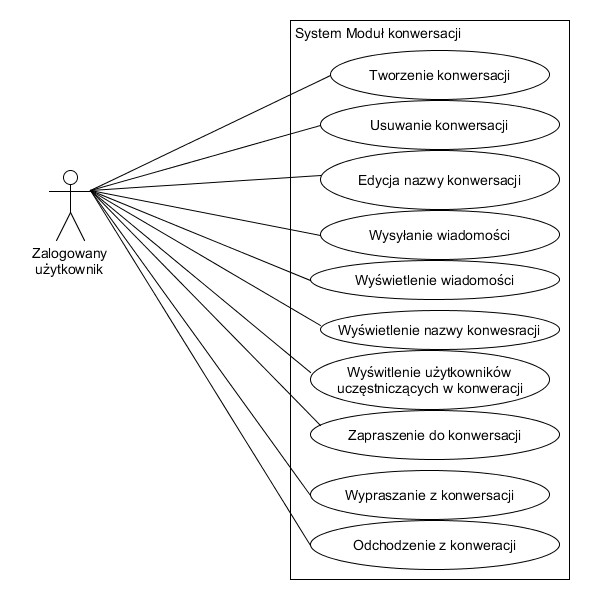
Rysunek B.5 Moduł użytkownika grupy



Rysunek B.6 Moduł profilu wydarzenia



Rysunek B.7 Moduł uczestnika wydarzenia



Rysunek B.8 Moduł konwersacji