Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Systemy Informacyjno-Decyzyjne

Budowa repozytorium informacji o wydarzeniach lokalnych, wzbogaconego danymi z serwisów społecznościowych

Mateusz Klimaszewski
Numer albumu 261426

promotor
dr inż. Mariusz Kamola

Warszawa 2017

Streszczenie

Tytuł: **Budowa repozytorium informacji o wydarzeniach lokalnych, wzbogaconego danymi z serwisów społecznościowych**

Celem pracy było zbudowanie repozytorium wydarzeń lokalnych korzystając z dostępnych źródeł informacji, w tym serwisów społecznościowych. W pracy została zawarta analiza różnych portali pod kątem możliwości wydobywania z nich rzetelnych i jasno określonych wydarzeń na terenie Polski. W ramach analizy zostały uwypuklone ograniczenia i trudności przy pozyskiwaniu informacji z serwisów społecznościowych. Praca zawiera również opis problemu dopasowywania do siebie tych samych wydarzeń, które pochodzą z różnych źródeł i krótkie omówienie narzędzi potrzebnych do budowy aplikacji i integracji z systemami zewnętrznymi. Na końcu zamieszczono wyniki i statystyki z otrzymanego repozytorium.

Słowa kluczowe: *serwisy społecznościowe, przetwarzanie danych, facebook api, meetup api.*

Abstract

Title: **Building repository of informations about local events, enriched in the data from social networking sites.**

The main purpose of this bachelor thesis was to build repository of informations about local events using available sources of informations, including social networking sites. This topic contains different websites analysis for extraction of reliable and clearly defined events in Poland. Within the analysis there are highlighted difficulties and limitations during obtaining informations from social networking sites. The thesis also includes description of matching events from another sources and a brief presentation of tools needed to build a web application and to integrate with external systems. To sum up there are results and statistics from achieved repository.

Keywords: *social networking sites, data processing, facebook api, meetup api.*

załącznik do zarządzenia nr 28/2016 r.
Rektora PW

Politechnika Warszawska

„załącznik nr 3 do zarządzenia nr 24/2016 Rektora PW

.....
miejscowość i data

.....
imię i nazwisko studenta

.....
numer albumu

.....
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta"

Spis treści

1. Wstęp

1.1. Cel pracy

Zamierzeniem podjętym w ramach pracy inżynierskiej było stworzenie repozytorium informacji o wydarzeniach lokalnych, wspomagając się różnymi serwisami społecznościowymi. Aplikacja internetowa stworzona w tym celu miałaby za zadanie wyszukiwanie danych w wspomnianych wcześniej portalach dla Polski z uwzględnieniem regionalnych wydarzeń, starając się, aby skupić się na całym obszarze kraju, a nie tylko na głównych ośrodkach. Wyszukiwanie wydarzeń polegałoby na przeglądzie dostępnych platform społecznościowych i wykorzystywaniu udostępnionych danych w celu znalezienia interesujących, w kontekście pracy, zagadnień.

Zanim przystąpiłem do przeglądu danych otrzymywanych z serwisów zdecydowałem się na analizę które dane są kluczowe, aby można było informację zakwalifikować jako wydarzenie. Podstawowym kryterium jest obecność tytułu bądź nazwy, bez tej podstawowej danej użytkownik repozytorium miałby problemy z wykorzystaniem zebranych danych. Kolejną istotną sprawą jest lokalizacja, wydarzenia bez jej podania mogą być niezwykle ciężkie do powiązania z konkretnym miejscem. Następnie ważna jest data początkowa wydarzenia. Ze względu na pojawiające się w portalach fikcyjne wydarzenia brak tej podstawowej danej sprawia, iż istnieje prawdopodobieństwo, że przedstawiona tam treść nie jest prawdziwa. Korzystając z tych założeń w rozdziale 2 przedstawiam przegląd dostępnych serwisów społecznościowych.

Dzięki zawartości repozytorium, pierwszym przychodzącym na myśl zastosowaniem jest wyszukiwanie ciekawych, interesujących imprez, koncertów. Istnieją także inne zastosowania takich zbiorów danych. Dzięki zgromadzonym informacjom na temat wydarzeń lokalnych w serwisach społecznościowych, firmy zajmujące się marketingiem mogą mieć dane statystyczne, w których miejscach proceder reklamowania wydarzeń poprzez serwisy społecznościowe jest najbardziej popularny. Inne branże, na przykład hotelarska, mogły by dzięki temu przewidywać, gdzie i kiedy zaistnieje prawdopodobnie wzrost zainteresowania ich usługami. Dzięki takim analizom, podane branże, ale też wiele innych, mogłyby działać sprawniej, szybciej i skuteczniej.

1.2. Przegląd istniejących rozwiązań

Przystępując do tworzenia pracy nie znałem żadnego rozwiązania, które spełniałoby jej kryteria lub byłoby bliskie tego celu. Jako osobie uważającej się za zaznajomioną i korzystającą z różnych portali społecznościowych, wydało mi się dziwne, iż nie istnieje produkt stworzony dokładnie i wyłącznie w takim celu. Założyłem jednak, biorąc pod uwagę szybką zmianę tej branży, iż bardzo prawdopodobne jest, że takie serwisy pozostają niezauważone przy gigantach tego specyficznego rynku.

Okazało się, że takie platformy istnieją, co więcej, są przykłady portali posiadających spore ilości użytkowników i cieszące się popularnością. W trakcie pracy poznałem aplikacje internetowe, które, choć nie w pełni, to jednak częściowo spełniają kryteria pod jakimi można oceniać serwis mający być celem tej pracy. Poniżej zamieściłem dwie najbardziej według mnie interesujące strony internetowe, które są zdecydowanie ciekawym przykładem jak aplikacje stworzone w celu prezentowania wydarzeń lokalnych powinny wyglądać.

1.2.1. Eventful

Eventful (`http://eventful.com/`) jest to serwis społecznościowy do wspierania rozrywek i wydarzeń na żywo. Chwali się on działaniem blisko partnerów z branży muzyki i rozrywki oraz wykonywaniem kampanii mailowych, telefonicznych i społecznościowych, które wspierają i poszerzają zainteresowanie m.in. lokalnymi koncertami oraz filmami [2]. Ważną informacją pochodzącą z ich strony jest fakt, iż posiadają aż 22 miliony zarejestrowanych użytkowników, co wydaje się sporą sumą, jak na tak zawężony krąg działalności.

Serwis ten spełnia wiele wymagań pod względem tematu tej pracy. Gromadzi dużo informacji, ma wielu użytkowników, do tego chwali się wydarzeniami lokalnymi. Niestety nie spełnia jednego kluczowego kryterium, czyli ma nikłą działalność w naszym kraju. Gdy w wyszukiwarce wpiszemy miasto stanowe Stanów Zjednoczonych, otrzymamy wiele stron wyników. Postanawiając poszukać wydarzeń w mieście wojewódzkim w Polsce, np Białymstoku otrzymamy dużo mniej, czasem wręcz zero (przy pisaniu pracy i przetestowaniu ilości otrzymałem jedno wydarzenie, które było w innym mieście, Kolnie).

Pomimo niespełniania tego kluczowego wymagania w kontekście wydarzeń w Polsce, *Eventful* może być interesujące nie tylko dla użytkowników, lecz także dla informatyków, gdyż udostępnia interfejs programistyczny i pozwala na pobieranie wielu informacji. Są to przede wszystkim wydarzenia, ale także ich miejsca, użytkownicy portalu, czy też dane o artystach. Jest to ciekawe źródło, które może stanowić atrakcyjną bazę dla wielu analiz, statystyk i różnych badań.

1.2.2. Kiwiportal.pl

Serwis `http://www.kiwiportal.pl/` to polski produkt, stworzony w ramach projektu *Imprezownia – internetowy serwis wydarzeń kulturalnych, rozrywkowych i sportowych*. Środki na jego realizację pochodziły z *Europejskiego Funduszu Rozwoju Regionalnego*. Na podstronie */okiwi* można przeczytać, iż w przeciwieństwie do innych portali, nie ogranicza się on do publikowania informacji z jednej dziedziny bądź regionu [32]. Można także uzyskać tam informację, że na `kiwiportal.pl` użytkownicy znajdą wydarzenia otwarte, kinowe, teatralne, popularne i o wiele więcej, wszystko na obszarze Polski.

Łatwo zatem zauważyć, iż serwis spełnia praktycznie wszystkie wymagania jakie użytkownik oczekiwałby od platformy mającej na celu przedstawiać wydarzenia lokalne w Polsce. Jednakże istnieje pewien mankament, niezauważalny na pierwszy rzut oka. Na podstronie `http://www.kiwiportal.pl/wydarzenia/m`, gdzie zaprezentowana jest lista województw z miastami, istnieje tylko po sześć miast dla każdego z województwa. Gdy stronę z wydarzeń któregoś z tych miast, można zauważyć pojedyncze wydarzenia dla okolicznych miejscowości. Na przykład po

przejrzeniu ponad dwustu wydarzeń dla Warszawy, znalazłem tam tylko po jednym przypadku zajęć z Wołominie, kabaretu w Łomiankach oraz spotkania na tematy historyczne w Piasecznie. Dlatego w mojej pracy zdecydowałem się postarać o uzyskiwanie informacji nawet dla mniejszych miasteczek, aby zbudować repozytorium, które będzie unikalnym rozwiązaniem tematyki pracy.

2. Przegląd i analiza serwisów społecznościowych

Przed przystąpieniem ustalenia architektury aplikacji i wyboru technologii głównym zadaniem było zdecydowanie, które serwisy społecznościowe posiadają informacje o wydarzeniach lokalnych. Głównym kryterium była ilość wydarzeń, poprawność ze względu na lokalizację, a także kompletność danych, gdyż informacja typu "Koncert" z samą datą, bez miejsca, godziny i dodatkowych informacji jest niezwykle ciężka do skojarzeniem z odbywającym się lub planowanym wydarzeniem. W tym rozdziale znajduje się opis różnych portali, sposób korzystania z każdego z nich oraz powody dla których zdecydowałem się na ich wykorzystanie bądź odrzucenie. Wszystkie serwisy zostały przetestowane korzystając z skryptów pisanych w języku *Python*, korzystając z poleceń linii komend (m.in. *curl*) bądź korzystając z dostępnych w ramach dokumentacji portali narzędzi do testowania.

2.1. Wykorzystane serwisy

2.1.1. Facebook

Facebook jako najbardziej popularny serwis społecznościowy [31] dostarcza możliwość wydobywania wielu informacji poprzez interfejs programistyczny (ang. *Application Programming Interface, API*) nazywany *Graph-Api* [6] oraz poprzez zestawy narzędzi (ang. *Software development kit, SDK*) przygotowanych specjalnie pod konkretną platformę bądź język programowania (m.in. Android SDK, iOS SDK, JavaScript SDK). Pierwsza możliwość opiera się o protokół *HTTP* (ang. *Hypertext Transfer Protocol*), a cały interfejs zaprojektowany jest w oparciu o architekturę REST (ang. *Representational State Transfer*). Rezultatami zapytań są obiekty w postaci wykorzystującej format *JSON* (ang. *JavaScript Object Notation*). Sprawia to, iż informacje są dostępne niezależnie od platformy (wymagane jest jedynie, aby mogła ona posługiwać się zapytaniami protokołu *HTTP*). Do korzystania potrzebne jest także zarejestrowanie aplikacji w celu uzyskania zestawu kluczy z których najważniejszy jest klucz dostępu (ang. *access token*). Powoduje to, iż nasza aplikacja jest związana z kluczem i przy generacji zbyt dużej ilości zapytań *HTTP* może ona zostać czasowo lub całkowicie zablokowana.

Niestety wraz z przejściem *Graph-Api* na wersję 2.0, Facebook odszedł od tzw. *Facebook Query Language*, który umożliwiał zadawanie zapytań w stylu języka *SQL* (ang. *Structured Query Language*) do poruszania się po *grafie* informacji udostępnianych przez serwis i dostawania się do danych, które nie były bezpośrednio udostępnione w interfejsie programistycznym. Migracja na wyższą wersję spowodowała, iż znalezienie informacji o wydarzeniach w podanej okolicy stało się odrobinę bardziej skomplikowane i powoduje dodatkowe problemy. W chwili obecnej (zgodnie z wersją 2.6), aby znaleźć wydarzenie (ang. *event*) należy wykorzystać

udostępnioną w interfejsie metodę wyszukiwania podając nazwę rzeczy której szukamy oraz dodać parametr mówiący o tym, że interesują nas tylko wydarzenia. Ze względu, iż jest to jedyna metoda, aby wydobyć dane z serwisu *Facebook* należało się zastanowić, jakie zapytania wpisywać w wyszukiwarkę, aby dostać jak najwięcej wydarzeń. Po różnych próbach zdecydowałem się na zastosowanie dwóch podejść:

- wyszukiwanie poprzez nazwę miasta
- wyszukiwanie poprzez słowo kluczowe

Pierwsze z rozwiązań polega na wykorzystaniu listy miast i dla każdej nazwy szukamy wydarzeń. Jest to dosyć naiwne rozwiązanie, ale algorytmy kryjące się wewnątrz serwisu społecznościowego dają nie najgorszą liczbę wydarzeń, ulokowanych w okolicach podawanego miasta. Drugie rozwiązanie polegało na ręcznym przetestowaniu i przemyśleniu pewnej liczby słów kluczowych, które mogą zaprowadzić do interesujących wyników. Poza najprostszymi wyrazami jak *wydarzenie* postanowiłem dodać bardziej specyficzne, skierowane mocno w jedną z podgrup wydarzeń, np. *mecz*, *koncert* czy też *konferencja*.

Przykładowe zapytanie wyszukujące wydarzeń w Warszawie:

https://graph.facebook.com/search?q=Warszawa&type=event
&access_token=example_token

Należy liczyć się z tym, że to rozwiązanie nie gwarantuje nam, iż udało się wydobyć z serwisu wszystkie istniejące wydarzenia i że uzyskane w przedstawiony sposób wyniki są w stu procentach prawidłowe. Istnieje możliwość, że lokalizacja tych wydarzeń zupełnie mija się z wyszukiwanym miastem, bądź też nie jest to prawdziwe wydarzenie, gdyż każda osoba korzystająca z serwisu może stworzyć takie wydarzenie i nie jest prowadzona żadna walidacja przy dodawaniu. Między innymi te problemy zostały opisane w rozdziale 3.4.

2.1.2. Meetup

Meetup jest to portal związany bezpośrednio z lokalnymi wydarzeniami. Pozwala on tworzyć grupy i organizować wydarzenia, które automatycznie powiadamiają członków. Grupy są publicznie dostępne, więc każda osoba która np. zmienia miejsce zamieszkania i szuka nowych znajomych, którzy mają takie same zainteresowania, może wykorzystać możliwości tego portalu. W Polsce serwis nie jest na tyle popularny, żeby każde miasto znajdujące się na stronie serwisu [28] miało przynajmniej jedną grupę, ale wiele większych miast ma od kilku do kilkudziesięciu grup.

Z programistycznego punktu widzenia Meetup jest również bardzo wygodny w obsłudze. Udostępnia interfejs programistyczny [29] oparty o architekturę REST i bezpośrednią możliwość wyszukiwania wydarzeń ze względu na ich długość i szerokość geograficzną.

Zapytanie wyszukujące wydarzeń w serwisie Meetup:

http://api.meetup.com/events?lon=longitude&lat=latitude&key=apiKey

Warto podkreślić, że odpowiedzi z *API* zawierają wszystkie potrzebne do budowy repozytorium informacje (a nawet o wiele więcej) w formacie *JSON*. Jedynym problemem jest zbyt duży obszar jaki jest brany pod uwagę wokół podanej długości i szerokości. Często zdarza się tak, że szukając wyników dla współrzędnych miasta powiatowego, możemy nagle otrzymać wydarzenia, które odbywają się w mieście wojewódzkim. Na szczęście bogate w dane odpowiedzi z interfejsu programistycznego zawierają dokładną lokalizację spotkania grupy, dzięki czemu można przeprowadzić walidację otrzymywanych danych.

2.1.3. Otwarte dane

Niektóre większe miasta w Polsce udostępniają publicznie niektóre dane. Wśród nich następujące miasta w swym interfejsie programistycznym udostępniają możliwość pobrania informacji na temat wydarzeń:

- Gdańsk [1]
- Poznań [34]
- Wrocław [36]

Ze względu na nieotrzymanie klucza dostępu do interfejsu programistycznego udostępnionego przez miasto Wrocław, wykorzystanie zostały tylko dwa pierwsze. Warto podkreślić, że informacje pochodzące z oficjalnych portali miast, można traktować jako zaufane informacje, w przeciwieństwie do typowych portali społecznościowych, na których każda osoba może założyć grupę, wydarzenie lub innego rodzaju zdarzenie, zależnie od możliwości strony. Niestety wykorzystanie takich źródeł informacji, dostępnych tylko dla pojedynczych miast, zaburza końcowe podsumowanie. Należy się zatem liczyć z faktem, iż dla dwóch poniższych miast, ilość zgromadzonych wydarzeń może wydawać się zwodnicza.

Gdańsk


```
{
  "id": 26772,
  "place": {
    "id": 9,
    "subname": "Winda",
    "name": "Gdański Archipelag Kultury"
  },
  "endDate": "2016-11-11T23:59:59+0100",
  "name": "Taniec towarzyski dla dorosłych",
  "urls": {
    "www": "http://winda.gda.pl"
  },
  "attachments": [],
  "descLong": "<p>Program zajęć:</p><p> </p><p>nauka podstawow
  "categoryId": 77,
  "startDate": "2016-11-11T00:00:00+0100",
  "organizer": {
    "id": 5,
    "designation": "Gdański Archipelag Kultury"
  },
  "active": 1,
  "descShort": "Program zajęć: nauka podstawowych kroków tańcó
  "tickets": {
```


Rysunek 2.1: *Część odpowiedzi na zapytanie otwartych danych Gdańsk*

Serwis *otwartygdansk.pl* udostępnia dane podzielone na osiem różnych kategorii i tyle samo typów plików - od gotowych spakowanych w ZIP (format kompresji bezstratnej i archiwizacji danych), przez format CSV (ang. *comma-separated values*) po JSON. Dane dotyczące wydarzeń przygotowane są przez *Instytut Kultury Miejskiej* na specjalnej stronie [5].

Zapytanie wyszukujące wydarzeń w Gdańsku:

http://planer.info.pl/api/rest/events.json?start_date=startDate&end_date=endDate

Usunięcie z adresu kawałka *.json* powoduje otrzymanie danych w formacie XML (ang. *Extensible Markup Language*) zamiast JSON. Część przykładowego rezultatu zapytania jest przedstawiona na rysunku 2.1.

Poznań

```
▾<operation name="getEvents">
   <input wsam:Action="http://www.poznan.pl/web-service/events/events/getEventsRequest" message="tns:getEvents"/>
   <output wsam:Action="http://www.poznan.pl/web-service/events/events/getEventsResponse" message="tns:getEventsResponse"/>
   <fault message="tns:Exception" name="Exception" wsam:Action="http://www.poznan.pl/web-service/events/events/getEvents/Fault/Exception"/>
 </operation>
▾<operation name="getEventsToDate">
   <input wsam:Action="http://www.poznan.pl/web-service/events/events/getEventsToDateRequest" message="tns:getEventsToDate"/>
   <output wsam:Action="http://www.poznan.pl/web-service/events/events/getEventsToDateResponse" message="tns:getEventsToDateResponse"/>
   <fault message="tns:Exception" name="Exception" wsam:Action="http://www.poznan.pl/web-service/events/events/getEventsToDate/Fault/Exception"/>
 </operation>
▾<operation name="getDayEvents">
   <input wsam:Action="http://www.poznan.pl/web-service/events/events/getDayEventsRequest" message="tns:getDayEvents"/>
   <output wsam:Action="http://www.poznan.pl/web-service/events/events/getDayEventsResponse" message="tns:getDayEventsResponse"/>
   <fault message="tns:Exception" name="Exception" wsam:Action="http://www.poznan.pl/web-service/events/events/getDayEvents/Fault/Exception"/>
 </operation>
▾<operation name="getCurrentDayEvents">
   <input wsam:Action="http://www.poznan.pl/web-service/events/events/getCurrentDayEventsRequest" message="tns:getCurrentDayEvents"/>
   <output wsam:Action="http://www.poznan.pl/web-service/events/events/getCurrentDayEventsResponse" message="tns:getCurrentDayEventsResponse"/>
   <fault message="tns:Exception" name="Exception" wsam:Action="http://www.poznan.pl/web-service/events/events/getCurrentDayEvents/Fault/Exception"/>
 </operation>
▾<operation name="getEventsFromDate">
   <input wsam:Action="http://www.poznan.pl/web-service/events/events/getEventsFromDateRequest" message="tns:getEventsFromDate"/>
   <output wsam:Action="http://www.poznan.pl/web-service/events/events/getEventsFromDateResponse" message="tns:getEventsFromDateResponse"/>
   <fault message="tns:Exception" name="Exception" wsam:Action="http://www.poznan.pl/web-service/events/events/getEventsFromDate/Fault/Exception"/>
 </operation>
```

Rysunek 2.2: *Operacje dotyczące wydarzeń dostępne w Poznań API opisane w pliku wsdl*

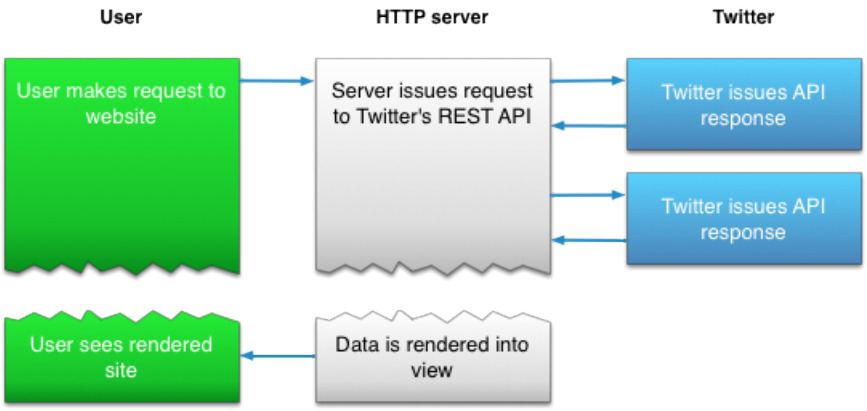
W przeciwieństwie do większości przedstawionych w tej pracy serwisów, otwarte dane w Poznaniu dotyczące wydarzeń opiera się na protokole *SOAP* (ang. *Simple Object Access Protocol*). W tym przypadku opiera się on na protokole *HTTP*, ale nie jest to jedyne technicznie możliwe rozwiązanie. Dokumentacja *API* pokazuje opis interfejsu korzystając ze sposobu opisu funkcjonalności jaki zapewnia *WSDL* (ang. *Web Services Description Language*) wykorzystujących sposób zapisu danych *XML* (ang. *Extensible Markup Language*). Plik typu *.xsd* przedstawiony w dokumentacji [15] (ang. *XML Schema Definition*) daje informacje jak będzie wyglądał schemat odpowiedzi na zapytania.

2.2. Odrzucone serwisy

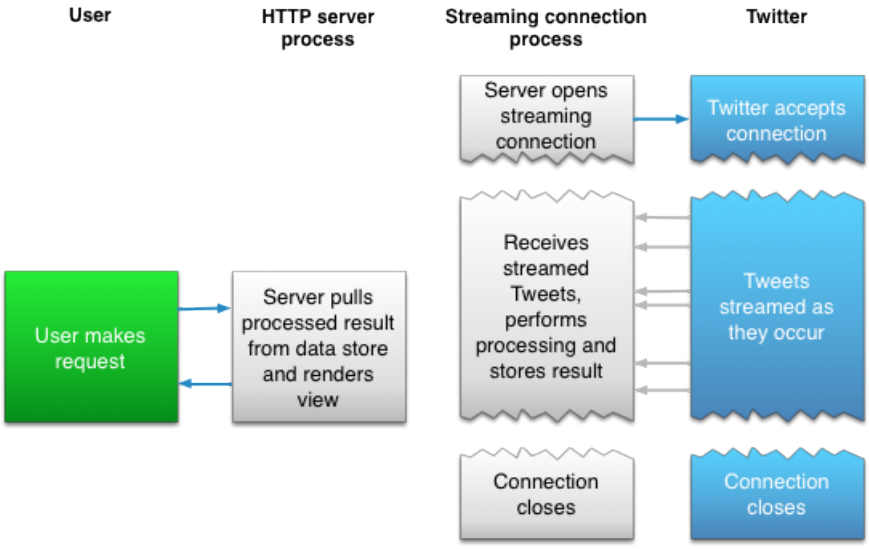
Niestety spora ilość portali społecznościowych nie nadawała się do wykorzystania przy tworzeniu repozytorium wydarzeń lokalnych. Każde z tych potencjalnych

źródeł danych dawało przesłanki, iż będzie można uzyskać ciekawe i wartościowe informacje. Niestety przeprowadzone testy i analiza ich wyników, które wraz z krótkim opisem znajdują się poniżej, ukazały ukryte mniej lub bardziej jawnie wady tych serwisów w kontekście przeprowadzanych badań.

2.2.1. Twitter



Rysunek 2.3: *Działanie REST API w przypadku Twittera [13]*



Rysunek 2.4: *Działanie Streaming API w przypadku Twittera [14]*

Twitter podobnie jak Facebook udostępnia interfejs programistyczny oparty o protokół *HTTP* i architekturę REST, jednakże ma dodatkowo *Streaming API*. Podejścia te zostały przedstawione na rysunkach 2.3 i 2.4.

Największym problemem przy korzystaniu z *REST API* które udostępnia *Twitter* są limity dotyczące zapytań. *Twitter* restrykcyjnie podchodzi do wielokrotnych zapytań, niektóre usługi są znacznie ograniczone, aż do 15 zapytań w okienku 15

minutowym. Przy 919 miastach w Polsce do obsłużenia daje to niezbyt satysfakcjonujący wynik.

Streaming API zezwala na jednorazowe podłączenie i otrzymywanie ciągle nowych wiadomości z Twittera tzw. *tweetów*. Niestety geolokalizacja jest na niezbyt dużym procencie komunikatów, do tego jeśli już jest to nie leży w granicach Polski. Przy przeprowadzonych testach, na ponad 10 tyś. wiadomości tylko 5 było z Polski i żaden z nich nie był wartościową, w kontekście tematu pracy, informacją.

2.2.2. Flickr i Instagram

Flickr [23] i Instagram [25] są to portale społecznościowe opierające się głownie na udostępnianiu zdjęć. Zarówno jeden jak i drugi serwis udostępnia interfejs programistyczny oparty o architekturę REST [24] [26]. Obydwa portale umożliwiają pobranie komunikatów ze względu na położenie geograficzne. Jednakże po przetestowaniu jakości udostępnianych informacji okazało się, że opisy samych zdjęć są niewystarczające do uzyskania wiedzy na temat wydarzeń lokalnych. Zwykle są to pojedyncze słowa z dodatkiem wielu tagów poprzedzonych symbolem # (ang. *hashtag*). Przy tak znikomych opisach ciężko wywnioskować czy zdjęcie jest związane z jakimś wydarzeniem.

2.2.3. Portal naszemiasto.pl

Portal naszemiasto.pl chwali się udostępnianiem informacji lokalnych. Na swojej stronie głównej udostępniają obiecującą listę miast [33]. W teorii wygląda to całkiem obiecująco, tym bardziej, iż po wejściu na stronę wybranego miasta możemy przejść do zakładki *regionalne* co sugerowałoby, iż znajdziemy tam regionalne wydarzenia.

W celu sprawdzeniu jakości danych z owego serwisu napisałem skrypt pobierający tytuły wydarzeń z zakładki *regionalne* dla każdego z miast. Niestety po porównaniu dwóch miast z tego samego województwa okazało się że liczba unikalnych dla tych miast wydarzeń jest bliska zeru. Mimo nazwy zakładki *regionalne* wydarzenia lokalne są tam praktycznie nieobecne. Większość wydarzeń pochodzi miasta wojewódzkiego, Warszawy lub są to informacje na szczeblu krajowym bądź międzynarodowym, które w kontekście pracy nie są wydarzeniami poszukiwanymi.

2.2.4. Lokalne serwisy informacyjne

Przez lokalne serwisy informacyjne w tej pracy rozumiem mniejsze strony internetowe o zasięgu maksymalnie powiatowym. Taki serwisy są często prowadzone przez niezależne osoby bądź w ramach jakiejś innej działalności, np. gazety lokalnej. Niestety, pomimo tego, iż takie portale mają lokalne wiadomości i jest w nich sporo informacji, których nie znajdziemy w innych źródłach, jest niezwykle ciężko pozyskać z nich dane ze względu na brak udostępnionego interfejsu programistycznego, a także różne struktury każdej z takich stron.

W celu wydobycia informacji należy stworzyć narzędzie analizujące kod *HTML* (ang. *HyperText Markup Language*). Jednakże większość portali jest robiona za pomocą różnych narzędzi i w różniący się od siebie sposób, dlatego niemożliwe

jest stworzenie uniwersalnego narzędzia do wyciągania informacji z każdego z portali. Jedynym rozwiązaniem byłoby stworzenie dedykowanego narzędzia dla każdego serwisu, bądź próba podziału i grupowania różnych stron na podstawie wykorzystanych technologii do ich stworzenia. Jednakże ilość stron bądź grup stron dla których trzeba by było przygotować takie rozwiązanie sprawia, iż to zadanie znacznie wykracza poza ramy czasowe pracy inżynierskiej. Do tego dochodzi fakt utrzymania takiego rozwiązania, gdyż ma się do utrzymania nie kilka, lecz kilkanaście bądź kilkadziesiąt (o ile nie kilkaset, jeśli grupowanie okaże się nieskuteczne) narzędzi. Ilość miast powoduje, że takich stron będzie w przybliżeniu tyle samo, więc zmiany na takiej ilości portali będą na porządku dziennym. Sprawia to, że do utrzymania takiego przedsięwzięcia potrzebny jest wręcz zespół programistów.

Na podstawie mojej analizy wygląda jednak na to, że jest to jedyna słuszna droga w celu wydobywania maksymalnie możliwej ilości informacji o wydarzeniach lokalnych. Jeżeli rozwiązanie zaprezentowane w tej pracy okazałoby się potrzebne i korzystne, jest to wysoce prawdopodobna ścieżka rozwoju aplikacji.

3. Realizacja pracy i rozwiązywanie problemów

3.1. Wstępne założenia

Głównym założeniem pracy było stworzenie aplikacji która będzie działała dobrze na przeglądarkach dla komputerów osobistych i urządzeń mobilnych. Ma ona prezentować repozytorium wydarzeń lokalnych tylko dla Polski, gromadząc dane z jak największej ilości różnych serwisów społecznościowych. Użytkownik powinien mieć także możliwość filtrowania wydarzeń względem daty, miejsca, a także źródła pochodzenia informacji. W razie możliwości powinien otrzymać także sposobność przejścia do serwisu, gdzie jest oryginalnie umiejscowione informacja o wydarzeniu. W razie wyłączenia bądź błędów leżących po stronie zewnętrznych serwisów z których aplikacja będzie pobierać wydarzenia, całość powinna działać bez problemów, dając użytkownikom możliwość przeglądania zebranych wcześniej informacji.

Dodatkowym miejscem, gdzie należało przyjąć założenia, były dane pochodzące z różnych serwisów. Każdy portal oferuje różniące się od siebie dane, względem kompletności i jakości. Zdecydowałem się na stosunkowo rygorystyczne podejście i założyłem, iż będę starał się akceptować tylko takie dane, które posiadają wymagane minimum niezbędnych informacji (m.in. datę rozpoczęcia, tytuł) oraz będą posiadały lokalizację w niezbyt dużej odległości od punktu centralnego miasta. Ze względu na wydarzenia przychodzące z różną ilością podstawowych danych nawet w obrębie jednego źródła, zdecydowałem się (jak tylko to będzie możliwe) przyjąć jednakowe założenia względem wszystkich pobieranych informacji i pozbywać się niepoprawnych lub niepełnych.

3.2. Wybór technologii

3.2.1. Warstwa prezentacji

Zgodnie w założeniami warstwa prezentacji (ang. *frontend*) w aplikacji miała być dostosowana nie tylko do przeglądarek dla komputerów osobistych, lecz także dla urządzeń mobilnych. Aby zrealizować to założenie, zdecydowałem się na wykorzystać platformę programistyczną (ang. *framework*) opartą na języku *Javascript* *VueJS* w wesji 2 dodatkowo korzystając z przystosowanych do niej komponentów typu *Material Design* [16] [21], których założeniem jest uniwersalność pomiędzy różnymi platformami. Dzięki temu nie musiałem się martwić, iż napisane własnoręcznie style CSS (ang. *Cascading Style Sheets*) będą działały poprawnie tylko na niektórych urządzeniach.

VueJS jest to framework oparty na wzorcu projektowym *MVVM* (ang. *model–view–view-model*). Wraz z dodatkowymi bibliotekami umożliwia budowanie pełnoprawnych aplikacji *SPA* (ang. *single-page application*). Przewagą tej platformy

programistycznej nad innymi jest jej prostota i uniwersalność. Może zostać zastosowana w celu budowy całej aplikacji (do tego zadania dostępny jest szereg dodatkowych i łatwych do skonfigurowania bibliotek), jednakże może zostać dodana tylko po to, aby zbudować tylko jakąś funkcjonalność niedostępną bądź trudną do wykonania w innych technologiach. Dodatkowo *VueJS* daje możliwość wygenerowania z linii komend prostej aplikacji gotowej do startu z dodanymi bibliotekami (które wybieramy w czasie konfiguracji) i możliwością pisania z wersji *es2016* języka *JavaScript* co znacząco usprawnia pracę.

W celach estetycznych na stronie głównej aplikacji użytkownik otrzymuje mapę Polski w postaci *mapy cieplnej* (ang. *heatmap*). Została ona wykonana korzystając z *Google Maps JavaScript API* [7]. Dzięki interfejsowi dostarczanemu przez firmę *Google* otrzymywana jest mapa na którą zostały wprowadzone koordynaty wszystkich zgromadzonych w bazie wydarzeń.

3.2.2. Serwer

Ze względu na znajomość języka Java uzyskaną w czasie studiów i pracy zawodowej zdecydowałem się na wybór tego języka do głównej części pracy. Wybrałem do tego najnowszą, ósmą wersję *Javy* ze względu na dodanie do niej elementów programowania funkcyjnego, zaletami takiej metodyki programowania są m.in. [39]:

- bezstanowość (ang. *statelessness*)
- przejrzystość referencyjna (ang. *referential transparency*)
- niezmienność danych (ang. *immutability*).

Całość postanowiłem oprzeć na otwartej platformie programistycznej utrzymywanej przez firmę *Privotal* jaką jest *Spring*. Zdecydowałem się na wybór *Spring Boot* ponad klasycznym *Spring MVC* ze względu na automatyczną konfigurację wielu rzeczy (m.in. połączenia do bazy danych, metryk monitorujących stan aplikacji) jaką oferuje *Spring Boot* i wygodę jaką daje brak konieczności korzystania z serwera aplikacyjnego *JEE* (ang. *Java Enterprise Edition*). Warto wspomnieć, iż *Spring MVC* i *Spring Boot* to najbardziej popularne frameworki dostępne na wirtualną maszynę Javy (ang. *Java Virtual Machine, JVM*) [30].

Do przechowywania pobranych informacji zdecydowałem się zastosować relacyjną bazę danych. Wybór padł na *PostgreSQL* ze względu na otwartość kodu i wygodną obsługę w konsoli. Jest to jedna z najbardziej popularnych baz danych i najbardziej popularna w kategorii darmowych baz danych [4]. *PostgreSQL* może być zainstalowany na wielu systemach operacyjnych (m.in. *Linux, Windows, OS X*). Istnieją też biblioteki pozwalające korzystać z bazy używając różnych języków programowania (m.in. *Java, C++, Python*).

3.2.3. Pozostałe narzędzia

Całość pracy została wykonana korzystając z systemu operacyjnego *Ubuntu 14.04 LTS* ze względu na moją znajomość tego systemu uzyskaną na studiach i w czasie pracy zawodowej, lecz wszystkie elementy aplikacji powinny bez problemów zadziałać na praktycznie każdej dystrybucji systemu *Linux* na której da się zainstalować wymagane narzędzia.

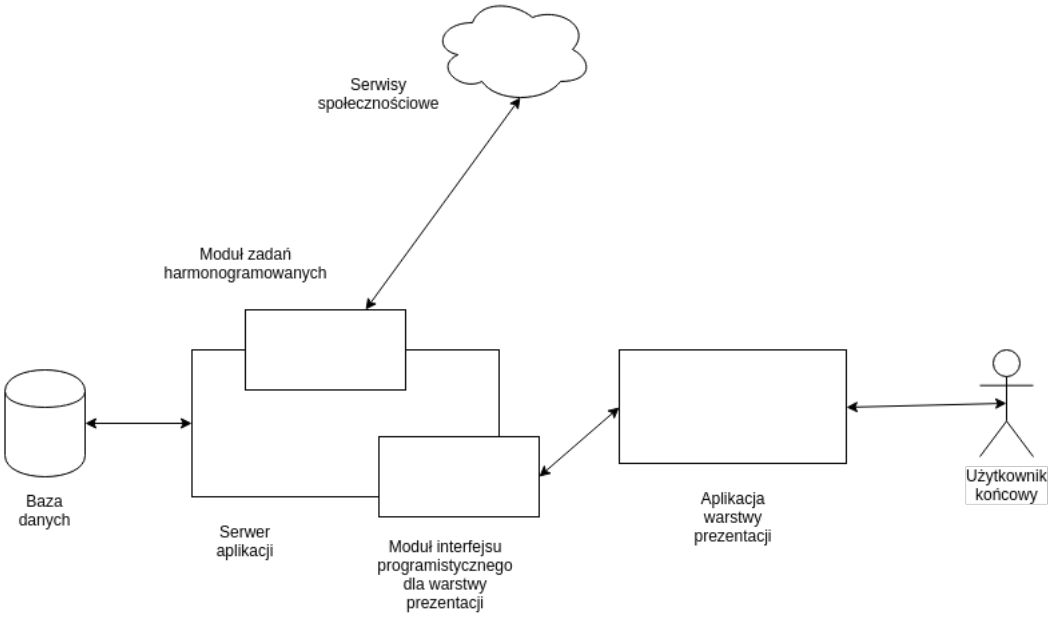
Do pisania pracy zdecydowałem się wykorzystać trzy różne zintegrowane środowiska programistyczne (ang. *Integrated Development Environment, IDE*), wszystkie będące narzędziami firmy Jetbrains [27]. Przy pisaniu skryptów w języku *Python*

- *Pycharm*, przy tworzeniu aplikacji będącej warstwą prezentacji - *Webstorm*. Do tworzenia głównej części pracy, czyli logiki serwera - *IntelliJ IDEA*.

W celu uniezależnienia kodu od środowiska programistycznego i ułatwienia korzystania z aplikacji skorzystałem z dwóch systemów budowania. Po stronie części opartej na języku *JavaScript* użyty został *npm*. Przy głównej części pracy opartej na języku *Java* wybór padł *Gradle*, z którego można korzystać poprzez komendy w terminalu. System ten posiada także wsparcie w większości IDE co ułatwia korzystanie z niego.

Ostatnim, ale nie najmniej ważnym narzędziem jest *Git*, czyli rozproszony system kontroli wersji. Pozwala na bezproblemową zmianę kontekstu, podział na gałęzie (ang. *branch*), pracę opartą na właściwościach (np. nowa gałąź dla każdej nowej właściwości), jednorazowe eksperymentowanie (gałęzie są łatwe do tworzenia i usuwania). Te cechy wraz z przewagą wydajnościową nad innymi systemami kontroli wersji sprawiają, że *Git* jest jednym z podstawowych narzędzi przy tworzeniu oprogramowania [10].

3.3. Architektura rozwiązania



Rysunek 3.1: *Poglądowy schemat architektury systemu*

Mając wyraźnie widoczne cele, architektura systemu wyklarowała się błyskawicznie. W ramach serwera aplikacyjnego zdecydowałem się na wydzielenie dwóch głównych modułów. Pierwszy z nich to część odpowiedzialna za interfejs programistyczny udostępniany dla warstwy prezentacji i oparty o architekturę *REST*, drugi zaś byłby odpowiedzialny za pobieranie wydarzeń w postaci zadań harmonogramowanych (ang. *cron jobs*). Poglądowy schemat architektury został zaprezentowany na rysunku 3.1.

3.3.1. Warstwa prezentacji

Zgodnie z możliwościami *VueJS* zdecydowałem się na architekturę opartą na *SPA*. Stworzony został jeden komponent, który jest stały dla każdego okna aplikacji. Zwiera on panel górny i boczny aplikacji. Przy zmianach ścieżki dostępu do zasobu (ang. *path*) w linku zmianie ulega tylko część której nie zajmują owe panele. Dzięki takiemu podejściu tworzenie widoków na konkretnych ścieżkach polegało na uzupełnieniu treści pomiędzy tagami *main-layout* co łatwo zobaczyć na przykładzie kodu *HTML* ścieżki */about* 3.1.

```
<template>
    <div>
        <main-layout>
            <div class="centered">
                <div class="md-display-3">
                    Bachelor thesis: Local Events Repository<br/>
                    Author: Mateusz Klimaszewski<br/>
                </div>
                <div class="md-display-1">
                    Warsaw University of Technology,<br/>
                    Faculty of electronics and information technology<br/>
                </div>
                <div class="md-display-1">
                    Winter semester 2016/2017
                </div>
            </div>
        </main-layout>
    </div>
</template>
```

Wydruk 3.1: Wykorzystanie komponentu *MainLayout* na podstronie.

Dzięki takiemu podejściu rozbiłem aplikację na kilka ścieżek:

- /
- /about
- /error
- /event/:id
- /province/:name

Każda z tych ścieżek składa się z jednego komponentu wykorzystującego tag *main-layout*, który zawiera specyficzne dla danego widoku inne komponenty. Dwie ostatnie z tych ścieżek posiadają dynamiczny parametr, odpowiednio *id* oraz *name*. *Id* jest to unikalny identyfikator wydarzenia, zaś *name* to po prostu nazwa województwa. Błędnie podany dynamiczny parametr, niezależnie czy w pierwszym czy w drugim przypadku skutkuje natychmiastowym przekierowaniem na podstronę oznaczoną w powyższej liście jako *error* na której użytkownik dostanie informację, iż podstrona którą podał jest nieprawidłowa. Także w przypadku wejścia ścieżkę, która wcale nie została zdefiniowana, skutek będzie ten sam - przekierownanie na *error*. Na stronie każdego z województw użytkownik ma możliwość filtrowania wyników po różnych parametrach, takich jak miasto (w obrębie danego województwa) czy data wydarzeń. Tabela w której wyświetlają się przeszukiwane wydarzenia zawiera także przycisk przekierowania do szczegółów, czyli *event/:id* i przejrzenia

wszystkich danych jakie udało się zgromadzić w repozytorium na temat danej imprezy.

3.3.2. Serwer

Zgodnie z tym co zostało przedstawione w paragrafie 3.3, zdecydowałem się w ramach serwera na wydzielenie dwóch głównych części:

- REST API
- harmonogramowane zadania

Pierwsza cześć służy jako źródło danych dla warstwy prezentacji. Udostępnia interfejs programistyczny który moduł widoku wykorzystuje, aby zaprezentować odpowiednie dane. Do wystawienia interfejsu programistycznego wykorzystałem dwie biblioteki rozszerzające framework *Spring Boot* czyli *Spring Boot Data Rest* i *Spring Boot Data Jpa*. Pierwsza z nich zezwala na implementację własnych metod w interfejsie programistycznym, druga umożliwia wykorzystanie metod z klas wykorzystujących wzorzec projektowy repozytorium (ang. *repository*). Biblioteka samodzielnie przekształca takie metody do postaci API, o ile nie jest jawnie w kodzie zaznaczone, że ma tego nie robić. Usprawnia to pracę programisty, który nie musi pisać zbędnego, schematycznego kodu.

Druga cześć jest dużo bardziej skomplikowana. Zawiera one zadania wykonywane z określoną częstotliwością. Zadaniami tymi jest integracja z odpowiednim portalem społecznościowym, pobranie danych z serwisu, przefiltrowanie ich względem określonych parametrów i zapis do bazy danych. Harmonogramowanie zadań jest udostępnione w *Springu* na podobnej zasadzie jak w systemach typu *Linux*.

REST API

Ze względu na wykorzystanie wcześniej wspomnianej technologii *Spring Boot Data JPA*, większość interfejsu programistycznego wystawionego dla warstwy prezentacji została automatycznie stworzona przy tworzeniu repozytoriów. Obok nich, zdecydowałem się na dodanie trzech dodatkowych

- /coords - koordynaty wszystkich wydarzeń
- /event/:id - wydarzenie o podanym *id* z wszystkimi potrzebnymi warstwie prezentacji danymi
- /province/:name/events - wszystkie wydarzenia dla województwa (parametr dynamiczny *name* dla nazwy województwa)
- /province/:name/events/:city wszystkie wydarzenia dla miasta w województwie (*name* jak wyżej, *city* jako nazwa miasta)

Dzięki tak prostemu podziałowi, warstwa widoku może bez problemu pobierać dane. Dodatkowo w przypadku pierwszej z nich *coords*, zdecydowałem się na umieszczenie wyniku operacji w do pamięci podręcznej (ang. *cache*) przyspieszając czas odpowiedzi. Dzięki wykorzystaniu platformy *Spring* taki zamysł w architekturze okazał się być prosty do zastosowania, co widać na wydruku 3.2. Dzięki takiemu rozwiązaniu odpowiedź nie jest za każdym razem wyliczana tylko wyciągana z pamięci podręcznej i przesyłana. W wypadku dodania nowego wydarzenia, część pamięci odpowiedzialna za przechowywanie wyniku tego zapytania jest czyszczona

i przy zapytaniu zapełniana nowymi wartościami. Jest to element kluczowy, gdyż wyciąganie z bazy danych wszystkich wydarzeń i iteracjach po nich jest kosztownym działaniem, dlatego najlepiej robić to jak najrzadziej.

Wydruk 3.2: Metoda interfejsu programistycznego wystawiona do użytku dla warstwy widoku z wykorzystaniem przechowywania wyniku zapytania w pamięci podręcznej.

```
1 @Cacheable(value = CacheConfig.COORDS_CACHE)
2 @RequestMapping(value = "/coords", method = RequestMethod.GET)
3 public List<Tuple> eventCoordinates() {
4     return StreamSupport.stream(eventRepository.findAll().spliterator(),
5             false)
6             .map(e -> Tuple.from(e.getCity().getLatitude(), e.getCity()
7             .getLongitude()))
8             .collect(Collectors.toList());
9 }
```

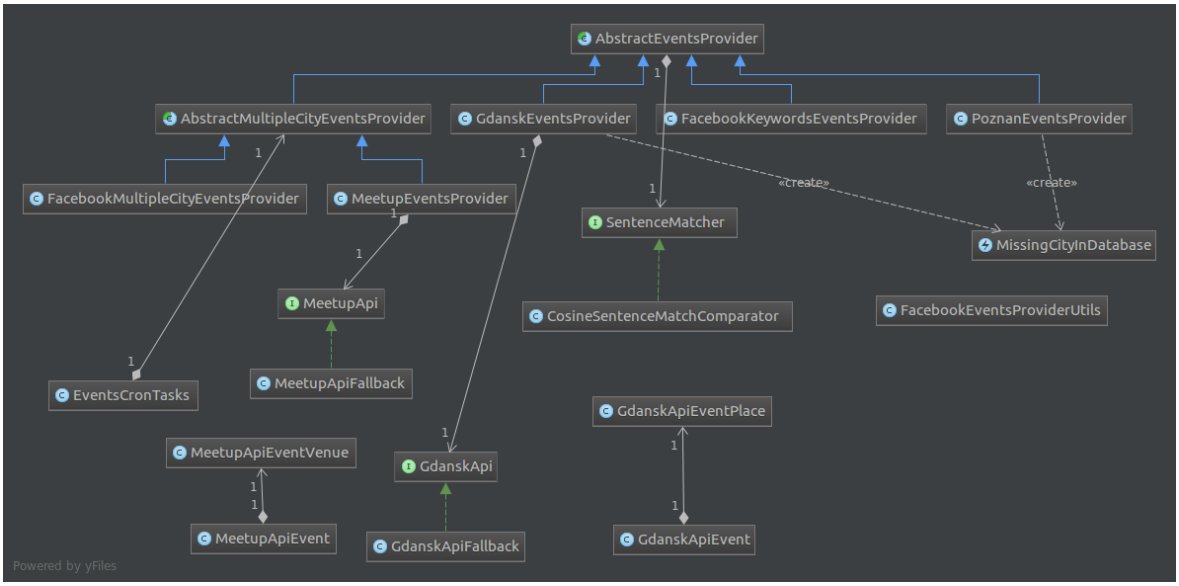
Harmonogramowane zadania

Mając do wyboru cztery różne źródła danych (*Meetup, Facebook, otwarte dane Gdańsk i Poznań*) z różnymi limitami zdecydowałem się na architekturę w której dla każdego źródła będzie istniała możliwość ustalenia częstotliwości wykonywania zdania przy starcie aplikacji. Ze względu na to, że przy każdym takim zadaniu oczekiwana jest grupa tych samych problemów, klasy musiały być jak najbardziej dostosowane do rozszerzenia o nowe źródła danych. Dlatego zdecydowałem się na podział źródeł danych na dwie podgrupy, gdzie pierwszą są *Facebook* i *Meetup* oparte na implementacjach tego samego *interfejsu*, a druga otwarte dane, oparte na *klasie abstrakcyjnej*. Taki podział jest optymalny, gdyż te grupy w domyśle będą napotykać podobne problemy i rozwiązanie w ten sposób daje swobodę implementacji. Schemat *UML* (ang. *Unified Modeling Language*), wykonany przy pomocy wtyczki do zintegrowanego środowiska programistycznego *Intellij*, został przedstawiony na rysunku 3.2.

3.4. Rozwiązywanie problemów

3.4.1. Lista miast z lokalizacjami

Do pobierania wydarzeń z serwisów *Meetup* i *Facebook* wymagana była lista miast wraz ze współrzędnymi geograficznymi. Dane wszystkich miast, wsi i o wiele więcej udostępnia na swojej stronie Główny Urząd Statystyczny [35]. Wszystkie pliki zwierające potrzebne informacje były w formacie *XML*, brakowało w nich jedynie położenia geograficznego miast. Do przetworzenia danych i zdobycia brakujących wykorzystałem język programowania *Python* oraz bibliotekę *geopy* [9], która pozwoliła na szybkie uzyskanie współrzędnych każdej z lokalizacji. Zdecydowałem się na napisanie skryptu który przetworzy dane, pobierze brakujące i przetransformuje je na odpowiedni format obsługiwany przez bazę danych, czyli plik formatu *SQL* w postaci operacji dodających dane do bazy danych (ang. *INSERT*), którego fragment został przedstawiony na wydruku 3.3. Dzięki temu wynik skryptu można



Rysunek 3.2: *Diagram UML harmonogramowanych zadań*

```
insert into public.city VALUES (717, 51.3220058, 21.947372, 'Kazimierz Dolny',
'LUBELSKIE');
insert into public.city VALUES (753, 50.3217212, 17.5801041, 'Prudnik',
'OPOLSKIE');
insert into public.city VALUES (794, 51.9509817, 14.7272197, 'Gubin',
'LUBUSKIE');
```

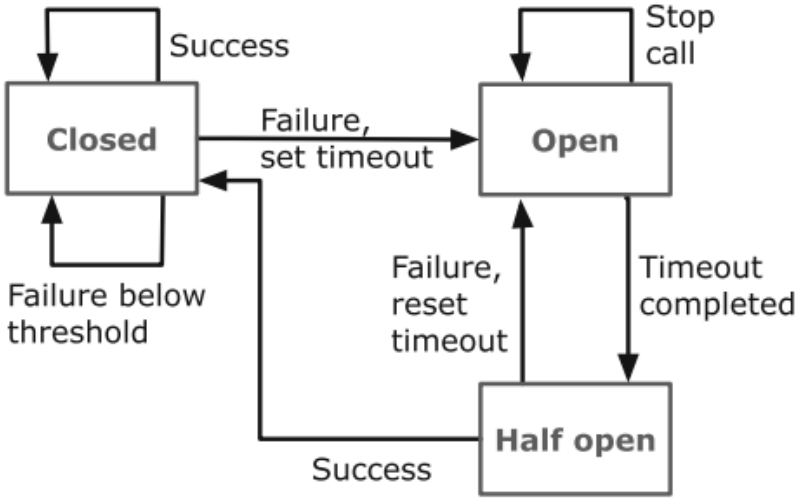
Wydruk 3.3: Przykład danych z pliku dodającego miasta do bazy danych.

wprowadzić bezpośrednio do bazy danych bądź podać plik w odpowiedni sposób wewnątrz kodu aplikacji i spowodować, aby został on wykonany przy starcie aplikacji.

3.4.2. Bezpieczeństwo integracji - Hystrix

Zgodnie z założeniami aplikacja miała być dostępna dla klienta niezależnie od dostarczania nowych wydarzeń. Z tego względu istniała potrzeba zabezpieczenia integracji z systemami zewnętrznymi, gdyż w chwili gdy np. *Facebook API* przestanie odpowiadać lub będzie odpowiadać z olbrzymim opóźnieniem, każde zapytanie z aplikacji będzie trwało bardzo długo, co powoduje oczekiwanie wątków na odpowiedź. Przy dużej ilości zapytań jakie wykonuje aplikacja może to spowodowa jej zawieszenie lub też wyłączenie przez co frontend nie będzie otrzymywał informacji lub będzie powodowało duże opóźnienia dla klienta i warstwa prezentacji będzie nieprzyjemna do korzystania dla użytkowników. Z tego względu zdecydowałem się wykorzystać bibliotekę *Hystrix*, a dokładniej jej dostosowaną do *Springa* wersję w ramach projektu *Spring Cloud Netflix* [3].

Hystrix to biblioteka stworzona przez firmę *Netflix*, która zawiera implementację wzorca projektowego *wyłącznik obwodu* (ang. *circuit breaker*). *Circuit breaker*



Rysunek 3.3: *Diagram stanów dla generycznej wersji wzorca circuit breaker [37]*

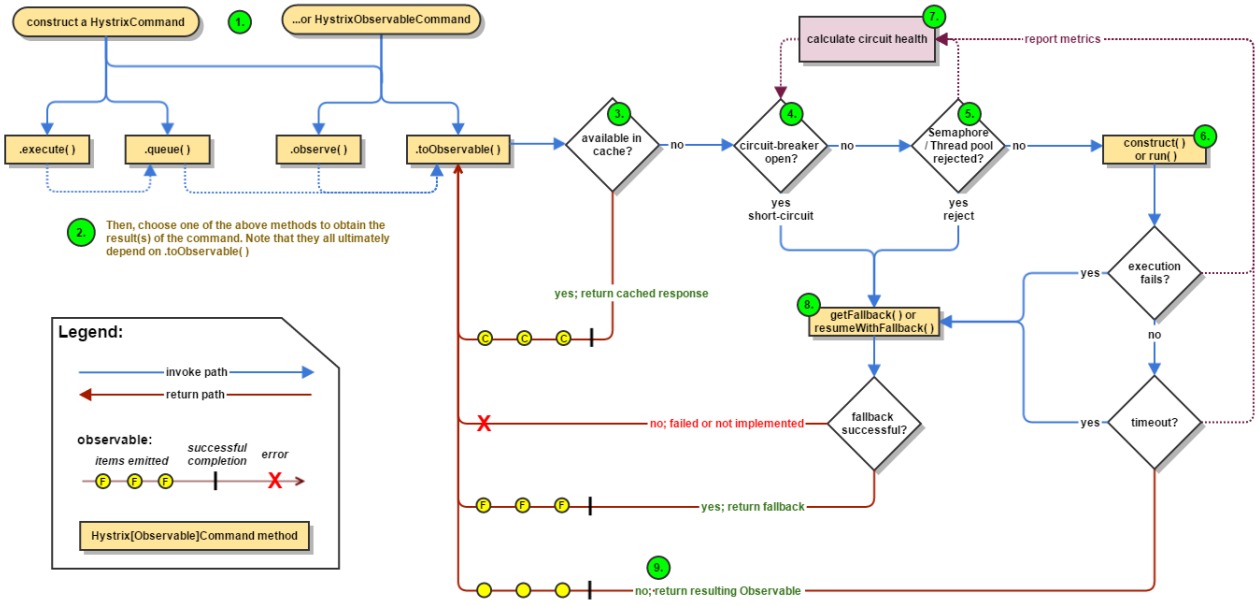
jest rozwiązaniem mającym na celu zabezpieczenie integracji z zewnętrznymi serwisami. Jego działanie prezentuje się następująco: cześć kodu odpowiedzialna za integracją (np. metoda) zostaje otoczona przez obiekt bezpiecznika, który monitoruje błędy pojawiające się w odpowiedziach. Gdy liczba błędów przekroczy pewien poziom (*Hystrix* ustawia poziom domyślnie na 20 błędów w czasie 5 sekund, ale można to łatwo skonfigurować wedle własnych potrzeb), bezpiecznik jest otwierany i wszystkie kolejne zapytania zwracają natychmiast błąd (lub podaną wartość, np. pustą listę jako odpowiedź), bez wysyłania zapytania do zewnętrznego systemu. Następnie co określony czas bezpiecznik próbuje wysłać zapytanie do systemu zamiast zwracać błąd. Jeżeli zapytanie jest zakończone sukcesem, bezpiecznik wraca do stanu zamknięty i wracamy do punktu początkowego [11]. Rozszerzony schemat działania na przykładzie *Hystrixa* znajduje się na rysunku 3.4.

Projekt *Spring Cloud Netflix* pozwala na stosunkowo proste otoczenie metody odpowiedzialnej za integracje z zewnętrznym system we wzorzec projektowy *circuit breaker*. Poza możliwością implementacji *klasy abstrakcyjnej - HystrixCommand<R>* daje możliwość dodania *adnotacji* nad istniejącą metodą. Konfiguracja w ten sposób sprawia, że można niskim kosztem wdrożyć bibliotekę *Hystrix* do istniejących projektów, bez wprowadzania olbrzymich zmian w istniejącym kodzie. Wykorzystanie *adnotacji* na zaimplementowanej wcześniej metodzie integracji z otwartymi danymi z Poznania znajduje się na wydruku 3.4.

Ważnym elementem jaki daje *Hystrix* jest monitoring zawarty w dodatkowej bibliotece. Pozwala on śledzić w czasie rzeczywistym stan wszystkich bezpieczników i ich metryki (m.in. ilość udanych, zakończonych błędem lub pozytywnie, lecz z dużym opóźnieniem zapytań), które dodamy do naszego pulpitu. Opierając się na tym można wykorzystać to gotowe rozwiązanie do np. powiadomień SMS,

Wydruk 3.4: Wykorzystanie adnotacji HystrixCommand na zaimplementowanej wcześniej metodzie zawierającej integrację z serwisem zewnętrznym.

```
@Override
@Scheduled(cron = "${cron.poznan.scheduled}")
@HystrixCommand(commandKey = "poznan-open-data",
    fallbackMethod = "storeEventsFallback")
public void storeEvents() {
    city = cityRepository.findByName(CITY_NAME)
            .orElseThrow(() -> new MissingCityInDatabase(CITY_NAME));
    Hibernate.initialize(city.getEvents());
    try {
        ((Root) JAXBContext.newInstance(Root.class)
                .createUnmarshaller()
                .unmarshal(new ByteArrayInputStream(new EventsService()
                .getEventsPort()
                .getEventsFromDate(LocalDate.now().toString())
                .getBytes(StandardCharsets.UTF_8))))
                .getEvent()
                .stream()
                .filter(this::hasImportantData)
                .forEach(e -> saveOrUpdateEvent(e,
                        this::findExistingEvent, this::updateEvent,
                        this::findSameEvent, this::saveEvent));
    } catch (Exception | JAXBException e) {
        log.error("Error connecting/parsing webservice Poznan: {}", e);
    }
}
```



Rysunek 3.4: *Schemat działania wzorca circuit breaker w bibliotece Hystrix [18]*

jeżeli integracja z jakimś system jest kluczowa dla działania aplikacji. Wygląd przykładowego monitoringu został zaprezentowany na rysunku 3.5, a opis na 3.6.

3.4.3. Problem duplikatów wydarzeń

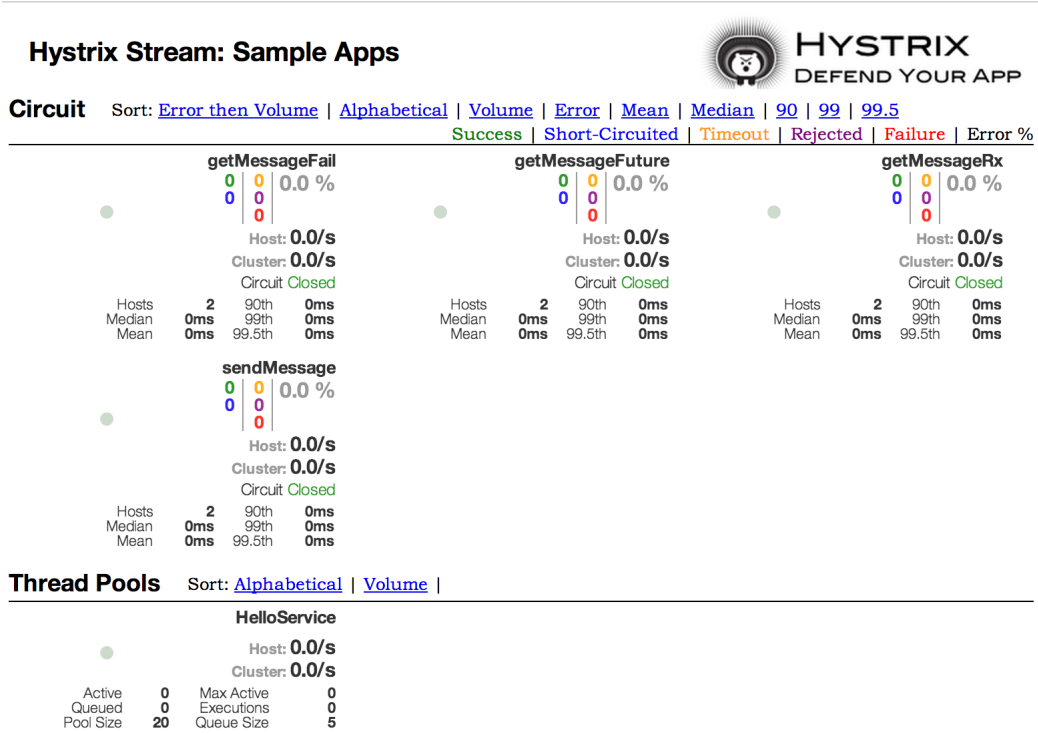
Ze względu na pobieranie wydarzeń z różnych serwisów istnieje problem dopasowania tych samych wydarzeń do siebie. Wewnątrz wybranych portali wydarzenia traktowane są jako unikalne (identyfikowane poprzez unikalną wartość *ID* zwracanego przez interfejs programistyczny), ale jeżeli z różnych źródeł mamy tę samą lokalizację i datę istnieje podejrzenie, że jest to duplikat.

W celu dopasowania do siebie wydarzeń wybrałem kolejną daną, która po ID, dacie wydarzenia i lokalizacji jest istotna, czyli tytuł. Bardzo precyzyjne porównywanie zdań, szczególnie w języku polskim, gdzie w przeciwieństwie do np. języka angielskiego istnieje wiele form fleksyjnych, jest nietrywialnym zadaniem. Na szczęście, po przeprowadzonych testach, jeżeli wydarzenie pojawia się na różnych serwisach, jego tytuł ulega bardzo małym zmianom lub jest identyczny. Dzięki temu zdecydowałem się zastosować pewne uproszczenie, ignorując zawiłość języka polskiego i zastosować porównywanie tytułów wykorzystując miarę odległości kosinusowej (ang. *cosine similarity*) pomiędzy nimi.

Odległość kosinusową otrzymujemy z następującego wzoru: [38]

$$SIM_C(\vec{t_a}, \vec{t_b}) = \frac{\vec{t_a} \cdot \vec{t_b}}{|| \vec{t_a} || \times || \vec{t_b} ||} \tag{3.1}$$

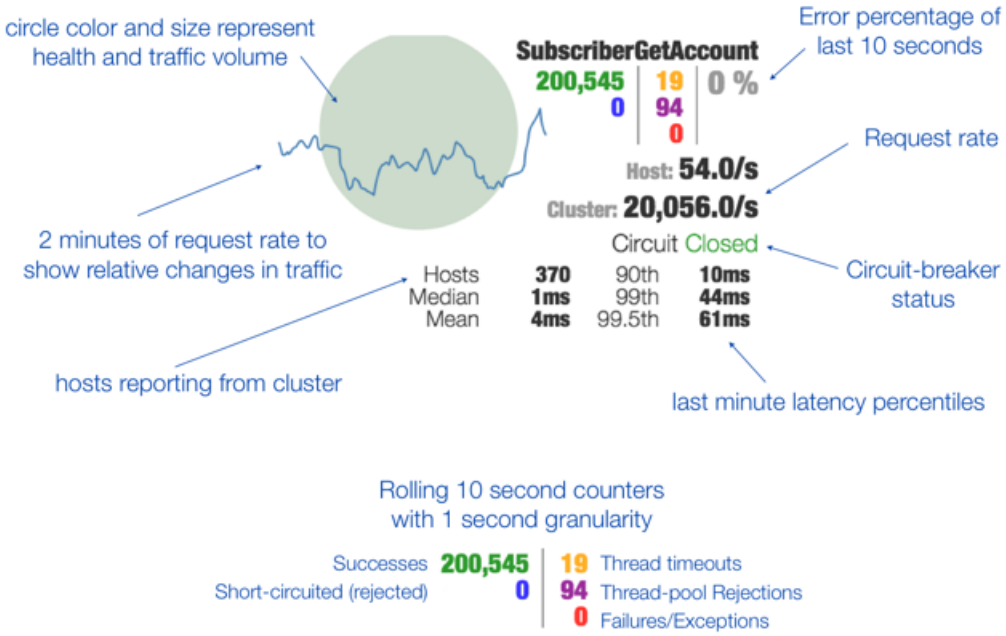
gdzie $\vec{t_a}$ i $\vec{t_b}$ to wektory do których porównywane tytuły zostały sprowadzone.



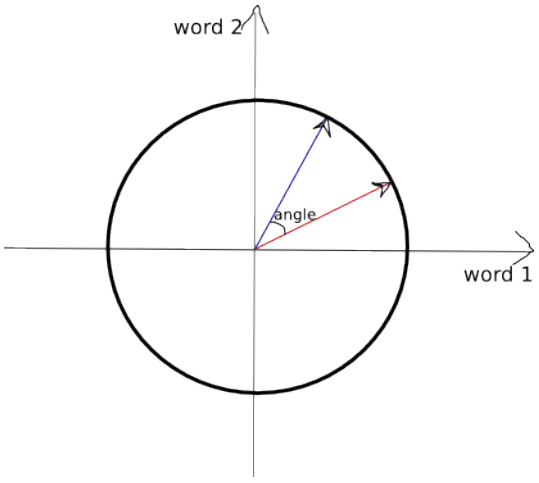
Rysunek 3.5: *Monitoring bezpieczników Hystrix [3]*

W przypadku zdań, przedstawienie geometryczne odległości kosinusowej byłoby nieczytelne, ale na rysunku 3.7 przedstawiona jest reprezentacja na podstawie porównania dwóch słów. Okrąg naniesiony na osie ma promień o długości 1. Zatem rezultatem obliczenia odległości kosinusowej jest, jak nazwa sugeruje, wartość kosinusa, czyli wynik jest między 1, a 0, gdzie 1 oznacza identyczność porównywanych zdań (bądź całych dokumentów, gdyż wykorzystanie odległości kosinusowej nie ogarnicza się tylko do zdań), a 0 całkowitą rozbieżność. W aplikacji wykorzystałem gotową implementację algorytmu zawartą w bibliotece *simmetrics* [12] wraz z najprostszą tokenizacją - opartą na spacjach. Wydzielenie wyrazów ze zdań jest niezbędną czynnością do stworzenia wektora z tytułu.

Tak obliczona odległość kosinusowa daje w zastosowaniu dobre rezultaty. Istnieje jednak prawdopodobieństwo, że przy większej ilości wydarzeń, gdy sytuacja duplikacji zdarza się na większą skalę, była to miara mająca niewystarczającą poprawność w kontekście osiągnięcia celu. Pierwszym krokiem w celu ulepszenia tego rozwiązania, byłoby zastosowanie lepsze jakości tokenizacji, sprowadzanie do wektora tylko słów kluczowych, a także testy innych algorytmów, np. indeksu *Jaccarda*. Dodatkowo dobrym pomysłem mogłoby okazać się zastosowanie bardziej zaawansowanych technik przetwarzania języka naturalnego, na przykład wydzielanie z tytułów słów kluczowych.



Rysunek 3.6: *Opis monitoringu pojedynczego bezpiecznika Hystrix [17]*

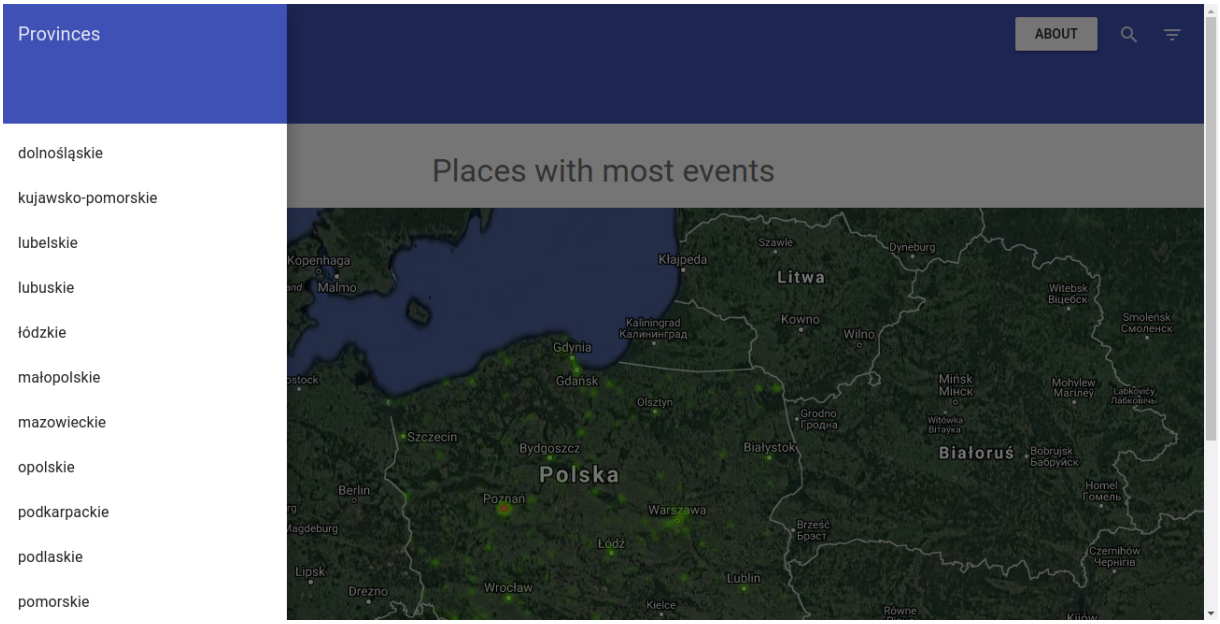


Rysunek 3.7: *Odległość kosinusowa pomiędzy dwoma wyrazami [20]*

4. Podsumowanie pracy

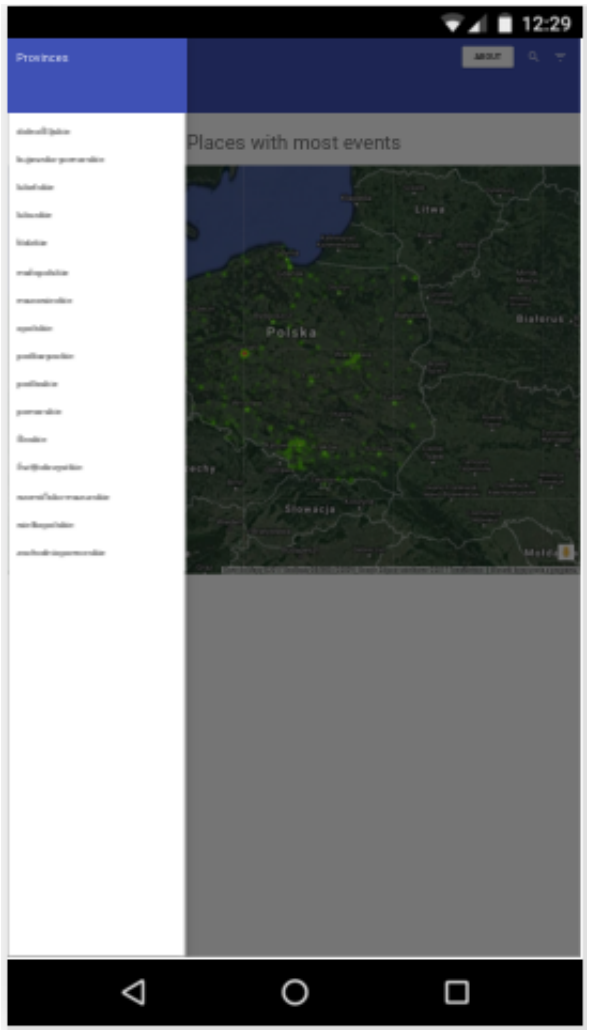
4.1. Prezentacja wyników

W wyniku pracy inżynierskiej powstał serwis internetowy oparty o dwie części: warstwy prezentacji i serwera. Warstwa prezentacji spełnia wymagania nowoczesnego projektowania stron internetowych, wygląda poprawnie na różnych przeglądarkach, tych działających na urządzeniach stacjonarnych i tych, z których użytkownik korzysta na urządzeniach mobilnych. Ze względu na prostotę zaprojektowanego interfejsu użytkownika, a także wykorzystanie komponentów do których użytkownicy są przyzwyczajeni, poruszanie się po portalu nie stanowi żadnego problemu. Na rysunku 4.1 ukazuje wygląd strony z perspektywy przeglądarki urządzenia stacjonarnego, z kolei na rysunku 4.2 prezentuje oblicze portalu na urządzeniu mobilnym (wykonane przy pomocy narzędzi dostępnych w przeglądarce *Chrome*, mającej opcję naśladowania niektórych urządzeń mobilnych).



Rysunek 4.1: *Wygląd strony głównej z otwartym panelem bocznym na przeglądarce komputera stacjonarnego*

Jednakże główną częścią pracy była nie warstwa prezentacji, a zbudowane repozytorium i moduł zajmujący się dostarczaniem informacji do bazy danych. Podsumowując, wydarzenia były szukane tylko dla Polski, a dokładnie dla 919 miast znajdujących się w obszarze państwa. Po wykonaniu jednorazowo wszystkich akcji dla zadań harmonogramowanych, wielkość repozytorium to



Rysunek 4.2: *Wygląd strony głównej z otwartym panelem bocznym na przeglądarce naśladującej telefon Nexus 5X*

ponad 3500 odnotowanych wydarzeń. Liczba ta może wydawać się niezbyt duża, lecz istotne było przyjęcie założeń, które spowodowało, że wiele z wydarzeń zostało odrzuconych, ze względu na brakujące dane lub nieprawidłowe położenie geograficzne. Zdecydowałem się na stosunkowo rygorystyczne kryteria odrzucania wydarzeń, aby mieć pewność, że zdecydowana większość umieszczonych w repozytorium danych będzie prawdziwa i będzie miała swoje odzwierciedlenie w realnym świecie. Istotnym punktem jest fakt, iż korzystając z otwartych danych dla miast Gdańsk i Poznań zaburzyłem statystyki ukazane w poniższej tabeli i trzeba wziąć na to poprawkę analizując to podsumowanie. Warto zauważyć, iż mimo tego, kolejność województw wg. liczby zgromadzonych wydarzeń jest całkiem podobna do kolejności województw wg. ludności w niej mieszkającej [19]. Można zatem stwierdzić, iż wykorzystanie serwisów społecznościowych (a przynajmniej części, którą ta praca wzięła pod uwagę) w branym pod uwagę obszarze Polski jest zbliżone do wprost proporcjonalnego względem liczby ludności.

Tablica 4.1: Liczba wydarzeń w repozytorium w każdym z województw

Nazwa województwa	Liczba zgromadzonych wydarzeń
Wielkopolskie	574
Śląskie	556
Mazowieckie	477
Dolnośląskie	398
Małopolskie	330
Pomorskie	276
Warmińsko-mazurskie	205
Zachodniopomorskie	194
Łódzkie	188
Kujawsko-pomorskie	186
Podkarpackie	167
Lubelskie	156
Lubuskie	146
Świętokrzystkie	140
Podlaskie	128
Opolskie	124

W trakcie tworzenia pracy wielokrotnie poddawałem analizie zgromadzone wyniki, aby obserwować pojawiające się zmiany wraz z rozwojem aplikacji i dorzucaniem bądź zdejmowaniem pewnych ograniczeń. Ciekawą zależność mogłem zauważyć, gdy w grudniu, korzystając z niefinalnej wersji aplikacji postanowiłem przeprowadzić analizę zgromadzonych danych. Patrząc na rozkład danych zaprezentowany na wykresie 4.2 można łatwo zauważyć anomalię w jednym dniu. Jest to sylwester 31.12, więc ilość wydarzeń osiąga rezultat, który mógłby być tam oczekiwany, znaleziono wielokrotnie więcej wydarzeń, niż w innych dniach na przestrzeni ponad miesiąca. Dodatkowo można spostrzec, iż liczba wydarzeń zależy od dnia tygodnia, w okolicach weekendów mamy ich więcej, niż w inne dni. Początek wykresu, mający najmniej znalezionych wydarzeń może sugerować, że okres adwentu w naszym kraju ma wpływ na liczbę organizowanych imprez. Dopiero w ostatnim okresie bezpośrednio poprzedzającym święta Bożego Narodzenia (czyli czas gdzie potencjalnie sporo osób może wziąć dodatkowe wolne od pracy) następuje skok w numerze wyszukanych przed aplikację informacji.

W finalnej wersji aplikacji liczba wydarzeń każdego dnia jest wyższa względem tego co zostało pokazane na wykresie 4.2. Wyniki z przełomu stycznia i lutego roku 2017 przedstawione są na rysunku 4.3 i zostały one zebrane dziesiątego stycznia 2017 po jednokrotnej iteracji po wszystkich zadaniach harmonogramowanych. Widać wyraźnie jak nagromadzenie wydarzeń w serwisach społecznościowych przejawia się w weekendy. Idąc dalej, poza zakres przedstawiony na wykresie, liczba wydarzeń będzie malała, jest to związane z tym, że nie każda informacja na portalach pojawia się z dużym wyprzedzeniem, czasami dopiero dzień czy dwa przed datą rozpoczęcia, co skutkuje tym, że czasami można znaleźć informację na stronie, dopiero na krótką chwilę przed datą rozpoczęcia. Pierwszy rzut oka na wykres, może spowodować sugestię, że dzień 15.01, niedziela, nie jest dniem, który z trzech dni z wolnymi wieczorami (dla ludzi pracujących w systemie poniedziałek - piątek), będzie dominował jeśli chodzi o ilość wydarzeń. Podkreśla to jeszcze fakt, że w dalszej części wykresu ponad każdą niedzielą, jeśli chodzi o

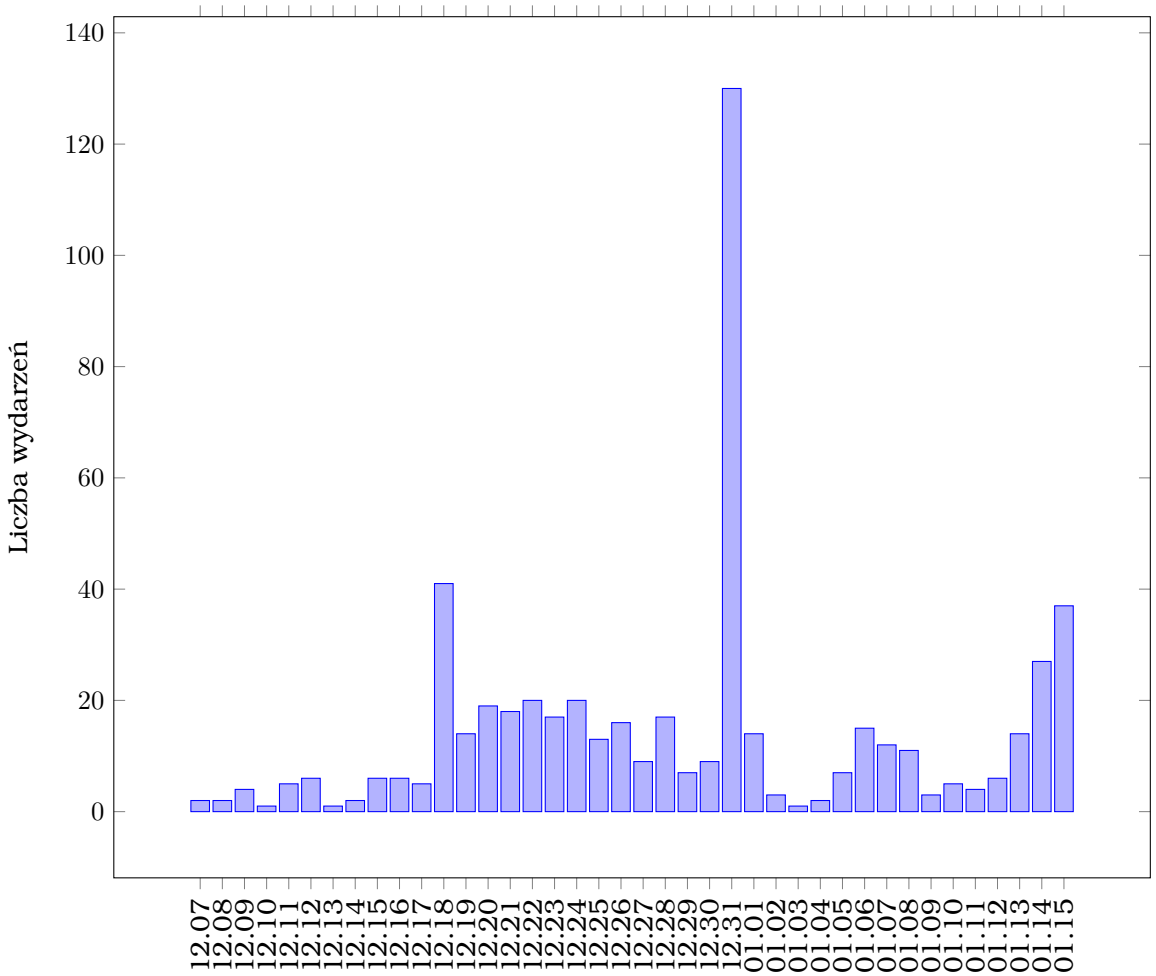
ilość zgromadzonych danych, jest sobota. Jednakże okazuje się, że jest całkiem dobre wyjaśnienie tego fenomenu, podkreślające skuteczność systemu. Dzień 15.01.2017 to *25. Finał Wielkiej Orkiestry Świątecznej Pomocy*, z tego względu można oczekiwać, że w tym celu powstanie sporo wydarzeń na obszarze Polski i część z nich będzie miała swoje odzwierciedlenie w serwisach społecznościowych.

Kwestią kluczową do przedstawienia jest też aspekt lokalności wydarzeń. Korzystając z danych zebranych, jak we wcześniejszym akapicie, w dniu 10.01.2017, dla województwa mazowieckiego znaleziono 472 wydarzenia, z czego 355 nie jest w największym mieście, Warszawie. Kolejnymi miastami w kolejności są: Radom (21 wydarzeń), Płock (18), Wołomin (17) i Legionowo (14). Dla trzydziestu miast w tym województwie znaleziono co najmniej 5 wydarzeń. Widać, że pomimo stosunkowo małej ilości pobieranych danych, jednak został w pewnym stopniu osiągnięty aspekt lokalności danych.

Rysunek 4.3: *Wydarzenie w Suwałkach ze strony o którym brakowało informacji na lokalnych stronach internetowych, a znalazło się w repozytorium [22]*

Korzystając z danych zebranych w dniu 19.01.2017, których było więcej niż we wcześniejszych testach (ponad 6500 wydarzeń), przeprowadziłem porównanie zebranych informacji dla miasta Suwałki z informacjami dostępnymi na portalach o tym mieście (*http://um.suwalki.pl/*, *http://www.suwalki24.pl/*, *http://niebywalesuwalki.pl/*). Portale te zostały wybrane w sposób subiektywny, korzystając kilku z pierwszych wyników pojawiających się w wyszukiwarce *www.google.com*. Korzystając z wydarzeń zebranych do końca miesiąca stycznia, sprawdziłem czy w repozytorium znajdują się wydarzenia na temat których nie ma informacji na podanych stronach. Okazało się, że wynik był pozytywny, dwa wydarzenia (impreza w kręgielni i spotkanie na temat wolności zgromadzeń i prawa do

Tablica 4.2: Liczba wydarzeń na przełomie grudnia i stycznia 2016/2017 zgromadzonych przy użyciu niefinalnej wersji aplikacji

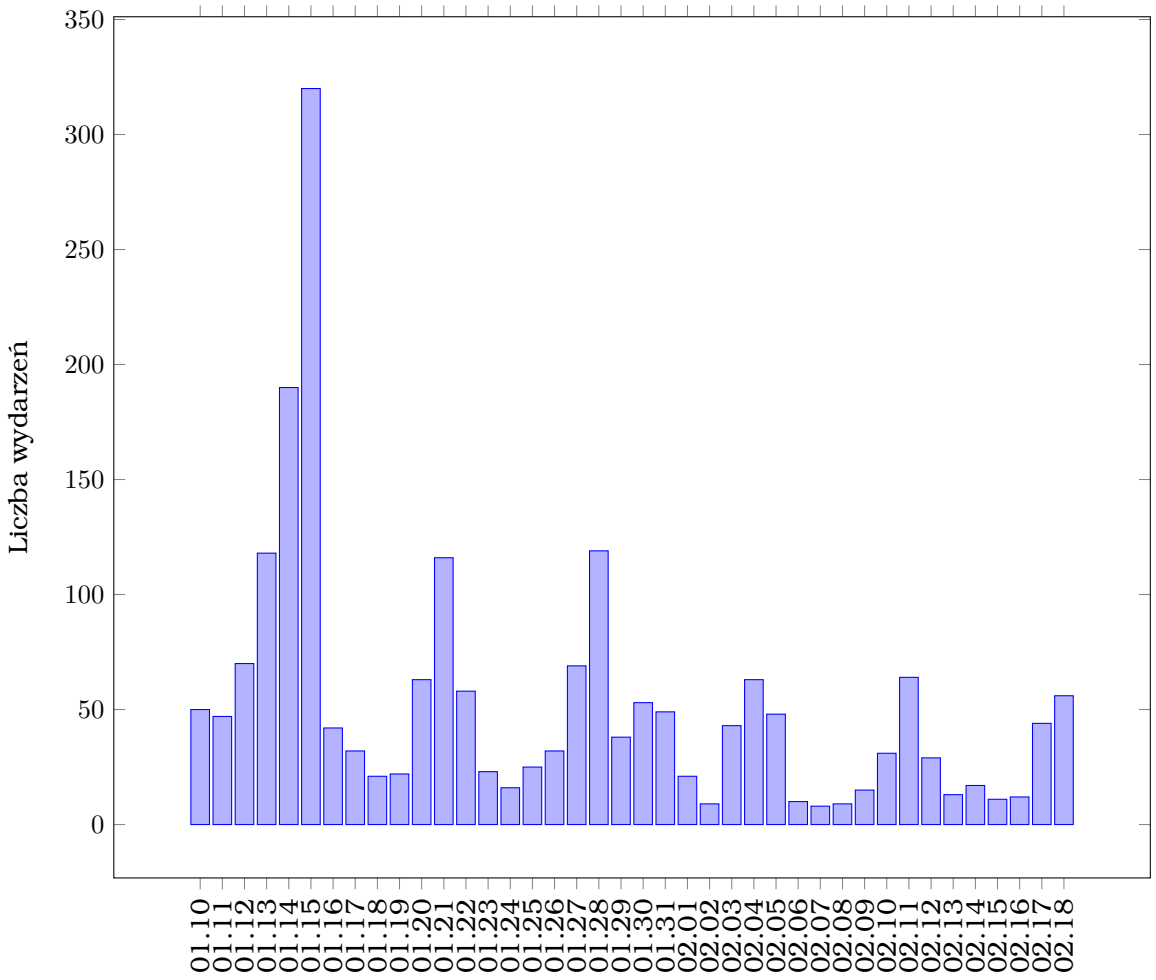


protestu, które cieszyło się sporym zainteresowaniem - przedstawione na fragmencie zrzutu ekranu 4.3) nie pojawiły się na żadnym z portali lub nie można ich było znaleźć (szukając na różne sposoby, m.in. po słowach kluczowych, korzystając z wyszukiwarki na stronach, przechodząc na odpowiednie podstrony o grupujące informacje o zbliżonej tematyce). Jednakże po przejrzeniu podanych wcześniej portali można było zauważyć, że znajduje tam więcej wydarzeń, niż w repozytorium. Gdy wykona się przecięcie zbiorów informacji zgromadzonych na podanych portalach i w repozytorium widać, iż unikalnych wydarzeń w pierwszym z wymienionych źródeł jest wielokrotnie więcej. Potwierdza to tezę w analizie 2.2.4, że przetwarzanie takich stron, to dobry kierunek rozwoju tej aplikacji.

4.2. Wnioski

Udało mi się zrealizować wszystkie założenia i cele jakie zostały zdefiniowane w ramach pracy. Stworzyłem repozytorium wydarzeń lokalnych i warstwę prezentacji dla użytkowników do przeglądania dostępnych danych zdobytych z portali społecznościowych. Spotkało mnie więcej problemów niż spodziewałem się wybierając ten

Tablica 4.3: Liczba wydarzeń na przełomie stycznia i lutego 2017 zgromadzonych przy użyciu ostatecznej wersji aplikacji



temat, przy elemencie, który uważałem za stosunkowo łatwy czyli zdobyciu odpowiednich źródeł danych

Istotnym punktem przy pracy z serwisami społecznościowymi jest ich ciągała zmiana. Portale, jak i wystawiane przez nie interfejsy programistyczne, ulegają ciągłym zmianom i modyfikacjom, gdyż to użytkownicy wpływają na definicję pod której sztandarem będą one dalej działać. Sprawia to, że pracując nad nimi trzeba przewidywać zmiany bądź szybko na nie reagować i nie być uzależnionym od jednego z nich, gdyż w społeczności internetowej trendy zmieniają się błyskawicznie, a portale zdobywają popularność i ją tracą. Takie *źródło danych* dobre jednego dnia, może wręcz z dnia na dzień zacząć przegrywać walkę z konkurencją i nie być tak interesujące jak było wcześniej. Dlatego praca z portalami społecznościowymi to nieustanna nauka i wychodzenie poza schematy w celu znalezienia sposobu na wydobycie z nich tego czego człowiek szuka. Kolejnym zanotowanym w trakcie pracy wnioskiem jest problematyczność uzyskania dużej ilości i jednocześnie wysokiej jakości danych. Biorą pod uwagę liczbę użytkowników poszczególnych serwisów wydaje się to niewiarygodne, jednak otrzymywane z nich informacje mogę być różnorakie. Dane wyglądające dobrze na pierwszy rzut oka, mogą okazać się całkowicie bezużyteczne, czego sam doświadczyłem pisząc tę pracę.

Bibliografia

[1] *Otwarte dane Gdańsk*, http://otwartygdansk.pl/.

[2] *Informacje o serwisie Eventful*, http://about.eventful.com/.

[3] *Spring Cloud Netflix*, http://cloud.spring.io/spring-cloud-netflix/spring-cloud-netflix.html.

[4] *Ranking popularności baz danych*, http://db-engines.com/en/ranking.

[5] *Dokumentacja REST API Gdańsk*, http://planer.info.pl/Dokumentacja_REST_API.pdf.

[6] *Dokumentacja Facebook Graph-Api*, https://developers.facebook.com/docs/graph-api.

[7] *Google Maps JavaScript API*, https://developers.google.com/maps/documentation/javascript/.

[8] *Dokumentacja Twitter API*, https://dev.twitter.com/docs.

[9] *Dokumentacjia biblioteki geopy*, https://geopy.readthedocs.io/en/1.10.0/.

[10] *Git dokumentacja*, https://git-scm.com/about.

[11] *Hystrix wiki*, https://github.com/Netflix/Hystrix/wiki/.

[12] *Biblioteka simmetrics*, https://github.com/Simmetrics/simmetrics.

[13] *Rysunek REST API dla serwisu Twitter*, https://g.twimg.com/dev/documentation/image/streaming-intro-1_1.png.

[14] *Rysunek Streaming API dla serwisu Twitter*, https://g.twimg.com/dev/sites/default/files/images_documentation/streaming-intro-2_1.png.

[15] *Poznań API events.xsd*, http://sit-wlkp.eu/xmlrepo/events.xsd.

[16] *Biblioteka Material Design*, https://material.io/.

[17] *Opis monitoringu pojedynczego bezpiecznika Hystrix*, https://raw.githubusercontent.com/wiki/Netflix/Hystrix/images/dashboard-annoted-circuit-640.png.

[18] *Schemat działania wzorca circuit breaker w bibliotece Hystrix*, https://raw.githubusercontent.com/wiki/Netflix/Hystrix/images/hystrix-command-flow-chart.png.

[19] *Ludność według województw*, http://stat.gov.pl/statystyka-regionalna/rankingi-statystyczne/ludnosc-wedlug-wojewodztw/.

[20] *Prezentacja graficzna odległości kosinusowej pomiędzy dwoma wyrazami*, http://stefansavev.com/assets/images/cosine/cosine.png.

[21] *Biblioteka vuematerial*, https://vuematerial.github.io/.

[22] *Strona wydarzenia na temat wolności zgromadzeń i prawa do protestu na portalu Facebook*, https://www.facebook.com/events/1802315236689455/.
[23] *Strona portalu społecznościowego Flickr*, https://www.flickr.com/.
[24] *Dokumentacja Flickr API*, https://www.flickr.com/services/api/.
[25] *Strona portalu społecznościowego Instagram*, https://www.instagram.com/.
[26] *Dokumentacja Instagram API*, https://www.instagram.com/developer/.
[27] *IDE firmy Jetbrains*, https://www.jetbrains.com/.
[28] *Lista miast serwisu Meetup*, https://www.meetup.com/cities/pl/?all=1.
[29] *Dokumentacja Meetup API*, https://www.meetup.com/meetup_api/docs/.
[30] *Popularność platform programistycznych Java*, https://zeroturnaround.com/rebellabs/most-popular-java-frameworks-tools-and-libraries-2016/.
[31] *Ranking serwisów społecznościowych*, http://www.ebizmba.com/articles/social-networking-websites.
[32] *Informacje o serwisie kiwiportal.pl*, http://www.kiwiportal.pl/okiwi.
[33] *Lista miast portalu naszemiasto.pl*, http://www.naszemiasto.pl/lista_miejscowosci/.
[34] *Otwarte dane Poznań*, http://www.poznan.pl/api/.
[35] *Dane udostępniane przez Główny Urząd Statystyczny*, http://www.stat.gov.pl/broker/access/prefile/listPreFiles.jspa.
[36] *Otwarte dane Wrocław*, http://www.wroclaw.pl/open-data/.
[37] Kjell Jørgen Hole. *Anti-fragile ICT Systems*. SpringerOpen, 2016.
[38] Anna Huang. Similarity measures for text document clustering. *Proceedings of the sixth new zealand computer science research student conference*, 2008.
[39] Konrad Siek. Wprowadzenie. *Metody Bezpiecznego Programowania (Programowanie Funkcyjne)*.

A. Wykaz symboli i skrótów

HTML - ang. Hypertext Markup Language
HTTP - ang. Hypertext Transfer Protocol
REST - ang. Representational State Transfer
API - ang. Application Programming Interface
SDK - ang. Software development kit
JSON - ang. JavaScript Object Notation
SQL - ang. Structured Query Language
XML - ang. Extensible Markup Language
CSS - ang. Cascading Style Sheets
MVVM - ang. Model-View-View-Model
SPA - ang. Single-page application
JEE - ang. Java Enterprise Edition
UML - ang. Unified Modeling Language

B. Spis rysunków

C. Spis tabel