Temat 2
Dla wskazanego kandydata w wyborach w USA, przedstaw w sposób czytelny na mapie finansowanie jego kampanii wyborczej, z uwzględnieniem darowizn bezpośrednich i poprzez komitety wyborcze.

Onaszkiewicz Przemysław, Gadawski Łukasz

24 stycznia 2016

1 Architektura rozwiązania

Aplikacja składa się następujących elementów:

1. Baza danych PostgreSQL wraz z rozszerzeniem PostGIS.
2. Instacja serwera *geoserver* umożliwiająca pobieranie danych geograficznych z bazy danych.
3. Zadania zaimplementowane jako tzw. *task*i w systemie budowania wersji gradle:
   - *getData* - umożliwiające pobranie plików CSV zawierających dane numeryczne odpowiednich danych finansowania. Poprzez zmianę skryptu możliwe jest pobranie danych z różnych przedziałów lat.
   - *cleanDb* - wykonuje połączenie z bazą danych oraz wykonanie skryptu tworzącego strukturę bazy danych.
4. Skrypt w języku python przetwarzający pliki CSV z danymi dotyczącymi finansowania i ładującymi odpowiednie dane do bazy danych.
5. Aplikacja internetowa oparta o framework aplikacji internetowych *Express* stworzony pod kątem aplikacji napisanych w Node.js.
   - biblioteka AngualarJS udostępniająca komponenty HTML,
   - biblioteka openlayers umożliwiające prezentację danych pobranych z serwera *geoserver*

2 Opis instalacji

Aplikacja była testowana na systemie *Linux Mint 17.3 Cinnamon 64-bit* w wersji *2.8.6*.

2.1 Przygotowanie bazy danych

Instalacja bazy danych wraz z rozszerzeniem PostGIS oraz sterownika wykorzystywanego przy uruchomieniu skryptu języka python umożliwiającego połączenie z bazą danych (w komendzie zawarte są również wszystkie niezbędne narzędzia potrzebne na kolejnym etapie realizacji projektu):

```
# sudo apt-get update
# sudo apt-get install postgresql postgresql-contrib
      postgis postgresql-9.4-postgis-2.1
      postgresql-9.4-postgis-scripts
      postgresql-9.4-postgis-2.1-scripts
      python3-psycopg2 install qgis python-qgis qgis-plugin-grass
      shp2pgsql
```

Przygotowanie bazy danych pod kątek wykorzystania przez serwer *geoserver* oraz aplikację internetową. Stworzenie użytkownika bazy danych:

```
# sudo -i -u postgres
# createuser --interactive // create "tass-user"
```

Stworzenie bazy danych:

```
# createdb tass
```

Instalacja rozszerzenie PostGIS umożliwiającego wykonywanie operacji na danych geograficznych:

```
# sudo -i -u postgres // login as superuser
# psql -d tass
# CREATE EXTENSION postgis;
# CREATE EXTENSION postgis_topology;
```

Zmiana hasła użytkownika:

```
# psql -d tass
# ALTER user "tass-user" PASSWORD '1234';
```

Stworzenie struktury bazy danych:

```
# gradle cleanDb
```

2.2 Pobranie oraz przygotowanie danych dotyczących finansowania kandydatów

Pobranie danych ze stron FEC (Federal Election Commission):

```
# gradle getData
```

Teraz gdy dane z danego okresu są pobrane należy wykonać skrypt ***extract_data.py***

2.3 Przygotowanie danych geograficznych

Przygotowanie danych geograficznych odbywa się poprze pobranie mapy Stanów Zjednoczonych ze strony ESRI: `http://www.arcgis.com/home/item.html?id=8d2012a2016e484dafaac0451f9aea24`

Dane są dostarczone w formacie GDP, należy je w dalszej kolejności skownertować do postaci SHP (Shapefile) przy użyciu narzędzia QGIS. Należy najpierw wczytać dane w formacie GDP, a następnie tak wczytane dane zapisać w formacie SHP.

Kolejnym krokiem jest zapis danych w formacie SHP do bazy danych PostgreSQL, aby umożliwić wykonywanie zapytań przestrzennych. Odbywa się to użycia narzędzia *shp2pgsql*:

```
# shp2pgsql -I -s 4326 esri-zip-codes.shp
        geo_zip_codes | psql -U tass-user -d tass
```

W tym momencie dane geograficzne zostały załadowane do bazy danych. Aby ułatwić ich prezentację baza danych zostanie podłączona do serwera *geoserver*, tak aby umożliwić pobranie danych przy pomocy usług WMS oraz WFS.

Należy zainstalować geoserver zgodnie z poradnikiem ze strony `http://docs.geoserver.org/stable/en/user/installation/linux.html`

Następnie uruchomić serwer poprzez następujące komendy:

```
# cd /usr/share/geoserver/bin
# ./startup.sh
```

Domyślnie panel administracyjny będzie dostępny pod adresem `localhost:8080/geoserver/web`. Standardowo nazwą użytkownika jest "admin" natomiast hasłem "geoserver".

Kolejnym krokiem jest podłączenie źródła danych z bazy PostgreSQL do serwera *geoserver* zgodnie z poradnikiem ze strony `http://docs.geoserver.org/stable/en/user/gettingstarted/postgis-quickstart/index.html`

Po tych krok możliwe jest pobranie całej mapy w formacie rastrowym do aplikacji przy użyciu usługi internetowej udostępnianej przez *geoserver* - **WMS**.

Następnie zostaną stworzone trzy warstwy (ang. *layers*) podstawie widoków wystawionych w bazie danych umożliwiające pobranie następujących danych w formacie GeoJSON:

- dotacje od osób indywidualnych na kandydata w wyborach prezydenckich,
- dotacje od komitetów wyborczych na kandydata,
- połączenie powyższych dwóch wyników finansowych w celu uzyskania sumarycznej sumy pozyskanych środków przez konkretnego kandydata.

Odpowiednio widoki zostały stworzone na podstawie następujących wyrażeń: Wpłaty indywidualne:

```
SELECT sum(cc.transaction_amt), cc.zip_code, gz.geom
FROM candidates ca
        inner join committees co on ca.cand_id=co.cand_id
```

```
        inner join ind_contribs cc on co.cmte_id=cc.cmte_id
        inner join geo_zip_codes gz on cc.zip_code=gz.zip_code
WHERE ca.cand_id='%cand_id%' and ca.cand_office='P'
GROUP BY cc.zip_code, gz.geom
```

Wpłaty poprzez komitety wyborcze:

```
SELECT sum(cc.transaction_amt), cc.zip_code, gz.geom
FROM candidates ca
        inner join committees co on ca.cand_id=co.cand_id
        inner join ind_contribs cc on co.cmte_id=cc.cmte_id
        inner join geo_zip_codes gz on cc.zip_code=gz.zip_code
WHERE ca.cand_id='%cand_id%' and ca.cand_office='P'
GROUP BY cc.zip_code, gz.geom
```

Łączne wpłaty:

```
SELECT coalesce(sum(ccc.ccsum + icc.icsum),
        sum(ccc.ccsum), sum(icc.icsum)) as suma,
        coalesce(ccc.zip_code, icc.zip_code) as zip_code,
        coalesce(ccc.geom, icc.geom) as geom
FROM
        (SELECT sum(cc.transaction_amt) as ccsum,
         gz.zip_code as zip_code, gz.geom
        FROM candidates ca
                inner join committees co on ca.cand_id=co.cand_id
                inner join comm_contribs cc on co.cmte_id=cc.cmte_id
                inner join geo_zip_codes gz on cc.zip_code=gz.zip_code
        WHERE ca.cand_office='P' and ca.cand_id='%cand_id%'
        GROUP BY gz.zip_code, gz.geom) as ccc
FULL OUTER JOIN
        (SELECT sum(cc.transaction_amt) as icsum,
         gz.zip_code as zip_code, gz.geom
        FROM candidates ca
                inner join committees co on ca.cand_id=co.cand_id
                inner join ind_contribs cc on co.cmte_id=cc.cmte_id
                inner join geo_zip_codes gz on cc.zip_code=gz.zip_code
        WHERE ca.cand_office='P' and ca.cand_id='%cand_id%'
        GROUP BY gz.zip_code, gz.geom) as icc
ON ccc.zip_code=icc.zip_code
GROUP BY coalesce(ccc.zip_code, icc.zip_code),
         coalesce(ccc.geom, icc.geom)
```

Po operacji dodania warstw w panelu administracyjnym *geoservera* możliwe jest wykonywanie zapytań do usługi WFS (Web Feature Service) umożliwiającej pobieranie danych geograficznych w postaci GeoJSON. Dane pobrane w takim formacie będzie można w odpowiedni sposób przeanalizować w aplikacji internetowej i stworzyć warstwy wektorowe umożliwiające prezentację danych na podkładzie mapy udostępnionej w ramach usług WMS.

2.4 Przygotowanie aplikacji internetowej

Ostatnim krokiem w celu prezentacji przechowywanych danych jest uruchomienie aplikacji internetowej. W przypadku naszej realizacji wykorzystaliśmy *fram-*

wework Express oparty o Node.JS. Komponenty HTML umożliwiające wybrania kandydata, czy źródła pobranych danych finansowych zostały zaimplementowane przy pomocy biblioteki AngularJS.

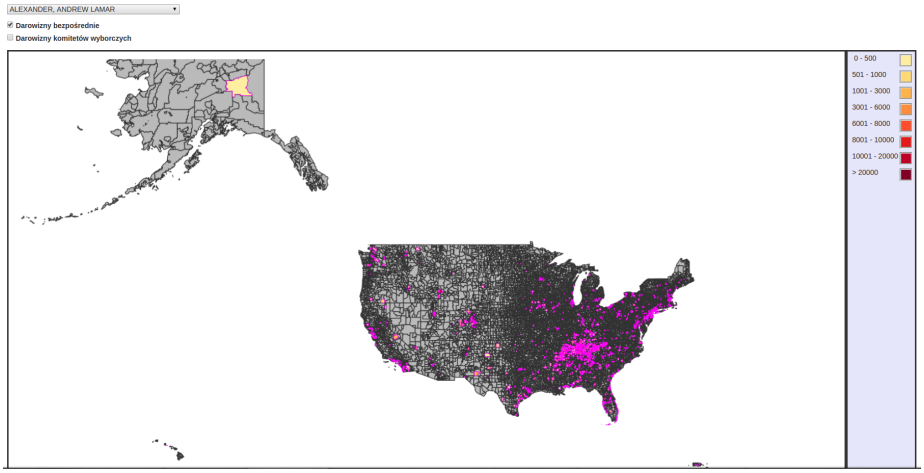
Za prezentację danych geograficznych oraz wykonywanie zapytań do serwera *geoserver* odpowiada biblioteka openlayers (zaimplementowana w języku JavaScript).

Aby uruchomić aplikację konieczne jest wykonanie następującej komendy:

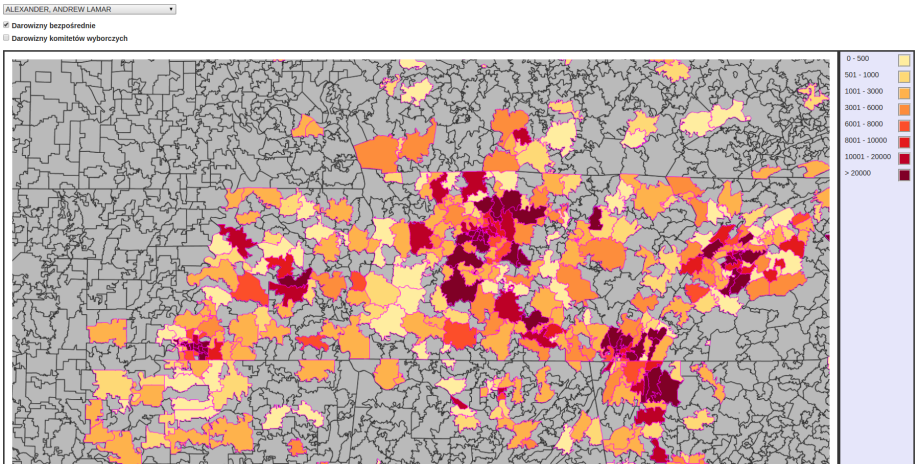
```
# npm install
# npm start
```

Teraz domyślnie aplikacja będzie dostępna pod adresem `http://localhost:3000/`

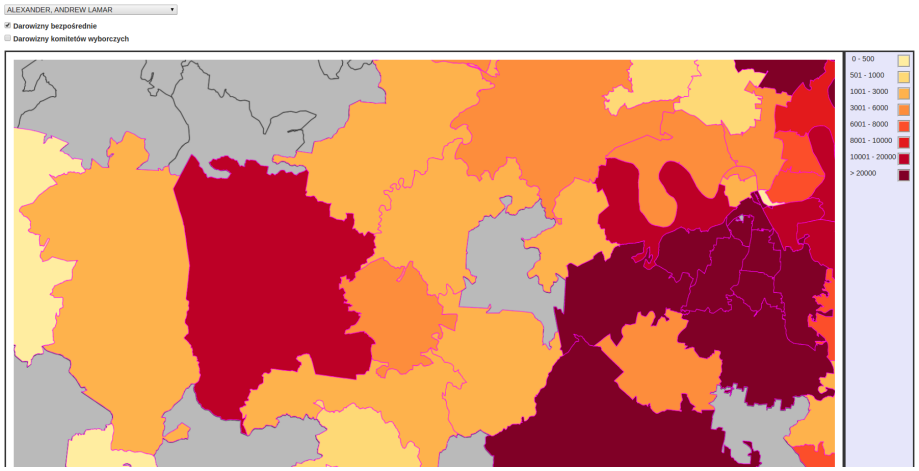
Poniżej przedstawione zostały przykładowe zrzuty ekranu zrealizowanej aplikacji:



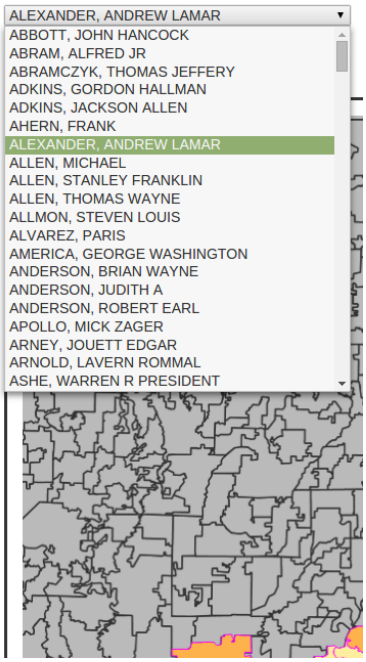
Rysunek 1: Całe Stany Zjednoczone, mała skala mapy.



Rysunek 2: Średnia skala mapy.



Rysunek 3: Duża skala mapy.



Rysunek 4: Wybór kandydata.

Jak widać na podstawie zrzutów niestety nie udało się nam w wyznaczonym czasie zrealizować agregacji danych w przypadku prezentacji danych na mapie o małej skali. Biblioteka openlayers udostępnia funkcjonalność *clusteringu*, tak aby dało się grupować obiekty o odpowiednich wartościach. Inną możliwością mogłoby być zgrupowania konkretnych wartości dla danego stanu.

W stosunku do pierwotnej koncepcji nie zostało zrealizowane wybranie rocznika wyborów ze względu na duże ilości danych przechowywane w bazie danych byłoby niezwykle niewydajne trzymanie wszystkich danych w jednej tabeli, tak jak zrobiliśmy od początku. Chcąc dokonywać zapytań do tak dużych ilości danych należałoby się zastanowić nad lepszą organizacją danych w bazie danych.

3 Załączniki

Załączone archiwum składa się z następujących plików:

- folder *config* przechowujący konfigurację projektu,
- folder *data-downloader* zawiera skrypty gradle umożliwiające pobranie danych oraz wykonanie skryptu tworzącego bazę danych (*data-downloader/tass.sql*),
- folder *data-db-extractor* zawierający skryptu w języku python umożliwiające napełnianie bazy danych pobranymi danymi,
- folder *web-app* zawierający aplikację internetową.

Nie załączamy w przesłanym archiwum warstw SHP zawierających mapę Stanów Zjednoczony, ponieważ zajmuje ona 1.9 GB, jeśli zajdzie taka potrzeba

możemy oczywiście dane nagrać na płytę bądź pendrive. Podobna sytuacja wygląda w przypadku danych dotyczących finansowania kampanii, które zajmują 3.9 GB.

Poniżej zamieszczony został skrypt SQL tworzący strukturę bazy danych:

```
DROP RULE IF EXISTS
"CANDIDATES_on_duplicate_ignore" ON CANDIDATES;
DROP RULE IF EXISTS
 "COMMITTEES_on_duplicate_ignore" ON COMMITTEES;
DROP RULE IF EXISTS
 "IND_CONTRIBS_on_duplicate_ignore" ON IND_CONTRIBS;
DROP RULE IF EXISTS
 "COMM_CONTRIBS_on_duplicate_ignore" ON COMM_CONTRIBS;
DROP RULE IF EXISTS
 "COMM_CAND_LINKAGES_on_duplicate_ignore" ON COMM_CAND_LINKAGES;

DROP TABLE IF EXISTS COMM_CAND_LINKAGES;
DROP TABLE IF EXISTS COMM_CONTRIBS;
DROP TABLE IF EXISTS IND_CONTRIBS;
DROP TABLE IF EXISTS COMMITTEES;
DROP TABLE IF EXISTS CANDIDATES;

CREATE TABLE CANDIDATES(
    CAND_ID VARCHAR(9) PRIMARY KEY,
    CAND_NAME VARCHAR(200),
    CAND_ELECTION_YR NUMERIC(4),
    CAND_CITY VARCHAR(30),
    CAND_ST VARCHAR(2),
    CAND_ZIP VARCHAR(9),
    CAND_OFFICE_ST VARCHAR(2),
    CAND_OFFICE VARCHAR(1),
    CAND_OFFICE_DISTRICT VARCHAR(2),
    CAND_PTY_AFFILIATION VARCHAR(3)
);

CREATE TABLE COMMITTEES(
    CMTE_ID VARCHAR(9) PRIMARY KEY,
    CMTE_NM VARCHAR(200),
    CMTE_CITY VARCHAR(30),
    CMTE_ST VARCHAR(2),
    CMTE_ZIP VARCHAR(9),
    CAND_ID VARCHAR(9),
    CMTE_TP VARCHAR(1),
    CMTE_PTY_AFFILIATION VARCHAR(3),
    FOREIGN KEY (CAND_ID) REFERENCES CANDIDATES(CAND_ID)
);

CREATE TABLE COMM_CAND_LINKAGES(
    LINKAGE_ID VARCHAR(9) PRIMARY KEY,
    CAND_ID VARCHAR(9),
    CAND_ELECTION_YR NUMERIC(4),
    FEC_ELECTION_YR NUMERIC(4),
    CMTE_ID VARCHAR(9),
```

```
    CMTE_TP VARCHAR(1),
    CMTE_DSGN VARCHAR(1),
    FOREIGN KEY (CAND_ID) REFERENCES CANDIDATES (CAND_ID),
    FOREIGN KEY (CMTE_ID) REFERENCES COMMITTEES (CMTE_ID)
);

CREATE TABLE COMM_CONTRIBS(
    SUB_ID NUMERIC(19) PRIMARY KEY,
    CMTE_ID VARCHAR(9),
    ENTITY_TP VARCHAR(3),
    NAME VARCHAR(200),
    CITY VARCHAR(30),
    STATE VARCHAR(2),
    ZIP_CODE VARCHAR(9),
    CAND_ID VARCHAR(9),
    TRANSACTION_AMT NUMERIC(14,2),
    TRANSACTION_DT DATE,
    FOREIGN KEY (CAND_ID) REFERENCES CANDIDATES (CAND_ID),
    FOREIGN KEY (CMTE_ID) REFERENCES COMMITTEES (CMTE_ID)
);

CREATE TABLE IND_CONTRIBS(
    SUB_ID NUMBER(19) PRIMARY KEY,
    CMTE_ID VARCHAR(9),
    ENTITY_TP VARCHAR(3),
    NAME VARCHAR(200),
    CITY VARCHAR(30),
    STATE VARCHAR(2),
    ZIP_CODE VARCHAR(9),
    TRANSACTION_AMT NUMERIC(14,2),
    TRANSACTION_DT DATE,
    FOREIGN KEY (CMTE_ID) REFERENCES COMMITTEES (CMTE_ID)
);

CREATE INDEX ON COMMITTEES (CMTE_ZIP);
CREATE INDEX ON COMMITTEES (CAND_ID);

CREATE INDEX ON CANDIDATES (CAND_ELECTION_YR);
CREATE INDEX ON CANDIDATES (CAND_ZIP);

CREATE INDEX ON COMM_CAND_LINKAGES (CAND_ELECTION_YR);
CREATE INDEX ON COMM_CAND_LINKAGES (CMTE_ID);

CREATE INDEX ON COMM_CONTRIBS (ZIP_CODE);
CREATE INDEX ON COMM_CONTRIBS (CAND_ID);
CREATE INDEX ON COMM_CONTRIBS (TRANSACTION_DT);
CREATE INDEX ON COMM_CONTRIBS (TRANSACTION_AMT);

CREATE INDEX ON IND_CONTRIBS (ZIP_CODE);
CREATE INDEX ON IND_CONTRIBS (TRANSACTION_DT);
CREATE INDEX ON IND_CONTRIBS (TRANSACTION_AMT);

CREATE RULE "CANDIDATES_on_duplicate_ignore"
AS ON INSERT TO CANDIDATES
```

```
103   WHERE EXISTS(SELECT 1 FROM CANDIDATES
104                  WHERE (CAND_ID)=(NEW.CAND_ID))
105   DO INSTEAD NOTHING;
106
107 CREATE RULE "COMMITTEES_on_duplicate_ignore"
108  AS ON INSERT TO COMMITTEES
109   WHERE EXISTS(SELECT 1 FROM COMMITTEES
110                WHERE (CMTE_ID)=(NEW.CMTE_ID))
111   DO INSTEAD NOTHING;
112
113 CREATE RULE "IND_CONTRIBS_on_duplicate_ignore"
114  AS ON INSERT TO IND_CONTRIBS
115   WHERE EXISTS(SELECT 1 FROM IND_CONTRIBS
116                WHERE (SUB_ID)=(NEW.SUB_ID))
117   DO INSTEAD NOTHING;
118
119 CREATE RULE "COMM_CONTRIBS_on_duplicate_ignore"
120  AS ON INSERT TO COMM_CONTRIBS
121   WHERE EXISTS(SELECT 1 FROM COMM_CONTRIBS
122                WHERE (SUB_ID)=(NEW.SUB_ID))
123   DO INSTEAD NOTHING;
124
125 CREATE RULE "COMM_CAND_LINKAGES_on_duplicate_ignore"
126 AS ON INSERT TO COMM_CAND_LINKAGES
127   WHERE EXISTS(SELECT 1 FROM COMM_CAND_LINKAGES
128                WHERE (LINKAGE_ID)=(NEW.LINKAGE_ID))
129   DO INSTEAD NOTHING;
```