

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. Instancja serwera *geoserver* umożliwiająca pobieranie danych geograficznych z bazy danych.
3. Zadania zaimplementowane jako tzw. *taski* 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 AngularJS udostępniająca komponenty HTML,
 - biblioteka openlayers umożliwiająca prezentację danych pobranych z serwera *geoserver*,z

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):

```
1 # sudo apt-get update
2 # sudo apt-get install postgresql postgresql-contrib
3     postgresql-9.4-postgis-2.1
4     postgresql-9.4-postgis-scripts
5     postgresql-9.4-postgis-2.1-scripts
6     python3-psycopg2 install qgis python-qgis qgis-plugin-grass
7     shp2pgsql
```

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

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

Stworzenie bazy danych:

```
1 # createdb tass
```

Instalacja rozszerzenie PostGIS umożliwiającego wykonywanie operacji na danych geograficznych:

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

Zmiana hasła użytkownika:

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

Stworzenie struktury bazy danych:

```
1 # gradle cleanDb
```

2.2 Pobranie oraz przygotowanie danych dotyczących finansowania kandydatów

Pobranie danych ze stron FEC (Federal Election Commission):

```
1 # 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 poprzez 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 skowner-tować do postaci SHP (Shapefile) przy użyciu narzędzia QGIS. Należy najpierw wczytać dane w formacie GDP, a następnie tak wczytane dane zapisać w for-macie SHP.

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

```
1 # shp2pgsql -I -s 4326 esri-zip-codes.shp
2 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:

```
1 # cd /usr/share/geoserver/bin
2 # ./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 krokach 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 su-marycznej sumy pozyskanych środków przez konkretnego kandydata.

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

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

```

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

```

Wpłaty poprzez komitety wyborcze:

```

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

```

Łączne wpłaty:

```

1 SELECT coalesce(sum(ccc.ccsun + icc.icsum),
2               sum(ccc.ccsun), sum(icc.icsum)) as suma,
3               coalesce(ccc.zip_code, icc.zip_code) as zip_code,
4               coalesce(ccc.geom, icc.geom) as geom
5 FROM
6     (SELECT sum(cc.transaction_amt) as ccsum,
7            gz.zip_code as zip_code, gz.geom
8     FROM candidates ca
9            inner join committees co on ca.cand_id=co.cand_id
10           inner join comm_contribs cc on co.cmte_id=cc.cmte_id
11           inner join geo_zip_codes gz on cc.zip_code=gz.zip_code
12     WHERE ca.cand_office='P' and ca.cand_id='%cand_id%'
13     GROUP BY gz.zip_code, gz.geom) as ccc
14 FULL OUTER JOIN
15     (SELECT sum(cc.transaction_amt) as icsum,
16            gz.zip_code as zip_code, gz.geom
17     FROM candidates ca
18            inner join committees co on ca.cand_id=co.cand_id
19           inner join ind_contribs cc on co.cmte_id=cc.cmte_id
20           inner join geo_zip_codes gz on cc.zip_code=gz.zip_code
21     WHERE ca.cand_office='P' and ca.cand_id='%cand_id%'
22     GROUP BY gz.zip_code, gz.geom) as icc
23 ON ccc.zip_code=icc.zip_code
24 GROUP BY coalesce(ccc.zip_code, icc.zip_code),
25           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.

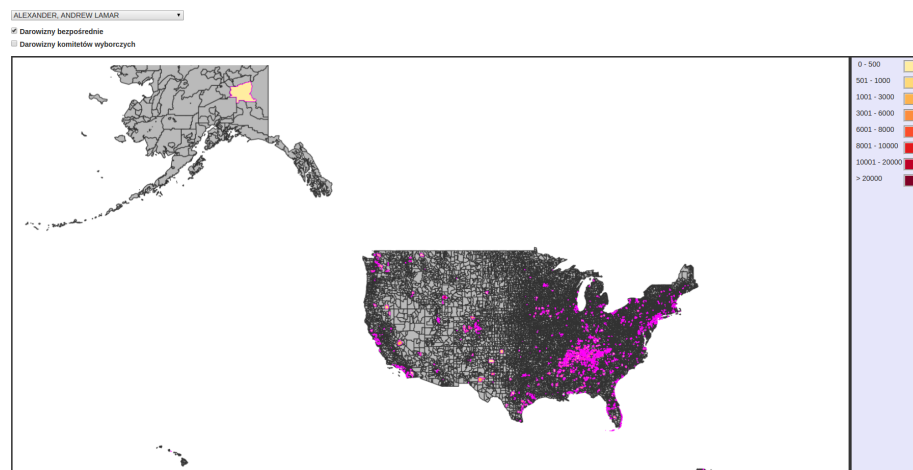
Za prezentację danych geograficznych oraz wykonywanie zapytań do serwera *geoserver* odpowiada biblioteka *openlayers* (zaimplementowana w języku JavaScript).

Aby uruchomić aplikację konieczne jest wykonanie następującej komendy:

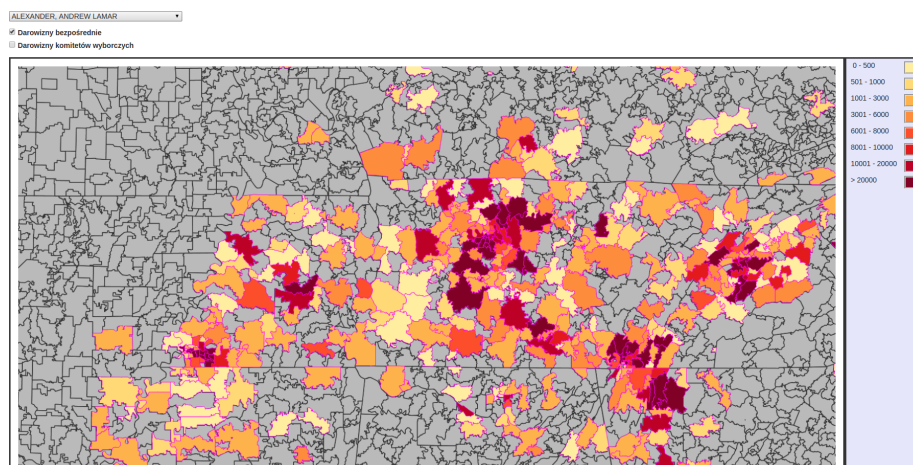
```
1 # npm start
```

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

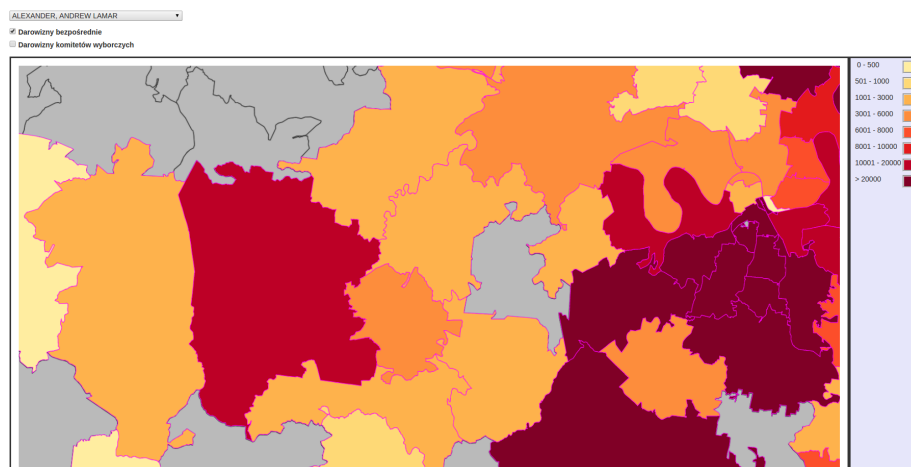
Poniżej przedstawione zostały przykładowe zrzuty ekranu zrealizowanej aplikacji:



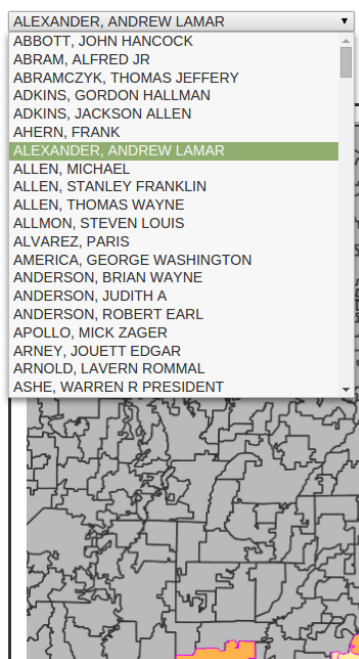
Rysunek 1: Całe Stany Zjednoczone, mała skala mapy.



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 zgrupowanie 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 niewygodne 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 Zjednoczonych, ponieważ zajmuje ona 1.9 GB, jeśli znajdzie taka potrzeba możemy oczywiście dane nagrać na płytę bądź pendrive.

Poniżej zamieszczony został skrypt SQL tworzący strukturę bazy danych:

```
1 DROP RULE IF EXISTS
2 "CANDIDATES_on_duplicate_ignore" ON CANDIDATES;
3 DROP RULE IF EXISTS
4 "COMMITTEES_on_duplicate_ignore" ON COMMITTEES;
5 DROP RULE IF EXISTS
6 "IND_CONTRIBS_on_duplicate_ignore" ON IND_CONTRIBS;
7 DROP RULE IF EXISTS
8 "COMM_CONTRIBS_on_duplicate_ignore" ON COMM_CONTRIBS;
9 DROP RULE IF EXISTS
10 "COMM_CAND_LINKAGES_on_duplicate_ignore" ON COMM_CAND_LINKAGES;
11
12 DROP TABLE IF EXISTS COMM_CAND_LINKAGES;
13 DROP TABLE IF EXISTS COMM_CONTRIBS;
14 DROP TABLE IF EXISTS IND_CONTRIBS;
15 DROP TABLE IF EXISTS COMMITTEES;
16 DROP TABLE IF EXISTS CANDIDATES;
17
18 CREATE TABLE CANDIDATES(
19     CAND_ID VARCHAR(9) PRIMARY KEY,
20     CAND_NAME VARCHAR(200),
21     CAND_ELECTION_YR NUMERIC(4),
22     CAND_CITY VARCHAR(30),
23     CAND_ST VARCHAR(2),
24     CAND_ZIP VARCHAR(9),
25     CAND_OFFICE_ST VARCHAR(2),
26     CAND_OFFICE VARCHAR(1),
27     CAND_OFFICE_DISTRICT VARCHAR(2),
28     CAND_PTY_AFFILIATION VARCHAR(3)
29 );
30
31 CREATE TABLE COMMITTEES(
```

```

32 CMTE_ID VARCHAR(9) PRIMARY KEY,
33 CMTE_NM VARCHAR(200),
34 CMTE_CITY VARCHAR(30),
35 CMTE_ST VARCHAR(2),
36 CMTE_ZIP VARCHAR(9),
37 CAND_ID VARCHAR(9),
38 CMTE_TP VARCHAR(1),
39 CMTE_PTY_AFFILIATION VARCHAR(3),
40 FOREIGN KEY (CAND_ID) REFERENCES CANDIDATES(CAND_ID)
41 );
42
43 CREATE TABLE COMM_CAND_LINKAGES(
44 LINKAGE_ID VARCHAR(9) PRIMARY KEY,
45 CAND_ID VARCHAR(9),
46 CAND_ELECTION_YR NUMERIC(4),
47 FEC_ELECTION_YR NUMERIC(4),
48 CMTE_ID VARCHAR(9),
49 CMTE_TP VARCHAR(1),
50 CMTE_DSGN VARCHAR(1),
51 FOREIGN KEY (CAND_ID) REFERENCES CANDIDATES (CAND_ID),
52 FOREIGN KEY (CMTE_ID) REFERENCES COMMITTEES (CMTE_ID)
53 );
54
55 CREATE TABLE COMM_CONTRIBS(
56 SUB_ID NUMERIC(19) PRIMARY KEY,
57 CMTE_ID VARCHAR(9),
58 ENTITY_TP VARCHAR(3),
59 NAME VARCHAR(200),
60 CITY VARCHAR(30),
61 STATE VARCHAR(2),
62 ZIP_CODE VARCHAR(9),
63 CAND_ID VARCHAR(9),
64 TRANSACTION_AMT NUMERIC(14,2),
65 TRANSACTION_DT DATE,
66 FOREIGN KEY (CAND_ID) REFERENCES CANDIDATES (CAND_ID),
67 FOREIGN KEY (CMTE_ID) REFERENCES COMMITTEES (CMTE_ID)
68 );
69
70 CREATE TABLE IND_CONTRIBS(
71 SUB_ID NUMBER(19) PRIMARY KEY,
72 CMTE_ID VARCHAR(9),
73 ENTITY_TP VARCHAR(3),
74 NAME VARCHAR(200),
75 CITY VARCHAR(30),
76 STATE VARCHAR(2),
77 ZIP_CODE VARCHAR(9),
78 TRANSACTION_AMT NUMERIC(14,2),
79 TRANSACTION_DT DATE,
80 FOREIGN KEY (CMTE_ID) REFERENCES COMMITTEES (CMTE_ID)
81 );
82
83 CREATE INDEX ON COMMITTEES (CMTE_ZIP);
84 CREATE INDEX ON COMMITTEES (CAND_ID);
85

```



```

86 CREATE INDEX ON CANDIDATES (CAND_ELECTION_YR);
87 CREATE INDEX ON CANDIDATES (CAND_ZIP);
88
89 CREATE INDEX ON COMM_CAND_LINKAGES (CAND_ELECTION_YR);
90 CREATE INDEX ON COMM_CAND_LINKAGES (CMTE_ID);
91
92 CREATE INDEX ON COMM_CONTRIBS (ZIP_CODE);
93 CREATE INDEX ON COMM_CONTRIBS (CAND_ID);
94 CREATE INDEX ON COMM_CONTRIBS (TRANSACTION_DT);
95 CREATE INDEX ON COMM_CONTRIBS (TRANSACTION_AMT);
96
97 CREATE INDEX ON IND_CONTRIBS (ZIP_CODE);
98 CREATE INDEX ON IND_CONTRIBS (TRANSACTION_DT);
99 CREATE INDEX ON IND_CONTRIBS (TRANSACTION_AMT);
100
101 CREATE RULE "CANDIDATES_on_duplicate_ignore"
102 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;

```