Praca dyplomowa zrealizowana na studiach podyplomowych„GIS -System Informacji Geograficznej" (edycja 2019/20)Pracownia GIS, Instytut Oceanografii, Uniwersytet Gdański

Autonomiczny system przetwarzania danych nieprzestrzennych do postaci wektorowej oraz rastrowej, połączony z geoserwerem udostępniającym usługę „wms" oraz „wfs".

WSTĘP

Celem projektu było stworzenie autonomicznego systemu zdolnego przetwarzać wejściowe dane nieprzestrzenne w formę użyteczną do analiz GIS – wektorową oraz rastrową. Głównym zadaniem, było ograniczenie ingerencji administratora w obróbkę danych. Ograniczenie nakładu manualnej pracy umożliwia wykorzystanie takiego systemu do wstępnej lub całkowitej analizy danych dostarczanych z zewnętrznych źródeł (bezpośrednio z urządzeń pomiarowych, baz danych , za pośrednictwem wywołań API). Aplikacja taka może być wykorzystany przez jednostki oraz osoby bez znajomości temtyki gis.

Kolejnym celem było uruchomienie ww. komponentów w środowisku serwerowym. Uzyskując tym samym dostępność do systemu globalnie za pomocą sieci internet. Jednocześnie udostępnienie zostały usługi w postaci „WMS" oraz „WFS".

Jako hipotetyczny model przyjęto miasto Słupsk w woj. pomorskim. Jako wstępne dane wejściowe nieprzestrzenne wykorzystano dane meldunkowe ludności na terenie miasta spreparowane na potrzeby projektu.

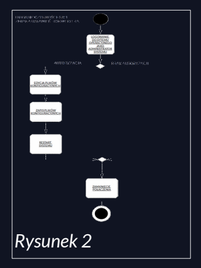
Dla projektu została zbudowana strona internetowa zawierającą pełną dokumentację techniczną projektu. Strona dostępna jest pod adresem „http://akiwior.pl/" . Strona została zbudowana na podstawie bezpłatnego schematu Mini Profile Template (*„https://templatemo.com / tm-530-mini-profile"*).


Rysunek 1: http://akiwior.pl

Jednocześnie domena „akiwior.pl" została przypisana do uruchomionego serwera.

Do zarządzania projektem użyto oprogramowania Git. Cały kod źródłowy systemu został opublikowany w zdalnym repozytorium na *„https://github.com/kivi-or/gis-dyplom.git"*.

METODYKA


Rysunek 2

W trakcie projektowania systemu stworzono model systemu oraz opisano go za pomocą formalnego języka modelowania **UML.** Opracowano przypadki użycia systemu pokazujące co system powinien robić oraz diagramy czynności określający w jaki sposób oprogramowanie osiągnąć ma te cele. Diagramy UML zostały wykorzystany jako plan projektu. Opracowano widoki przypadków użycia systemu względem wymagań A1, A2, A3, A4. Jeden z diagramów czynności został zaprezentowany na Rysunku nr 2. Pełna dokumentacja UML została umieszczona pod adresem „http://akiwior.pl/uml/uml.html" oraz w repozytorium „https://github.com/kivi-or/gis-dyplom.git".

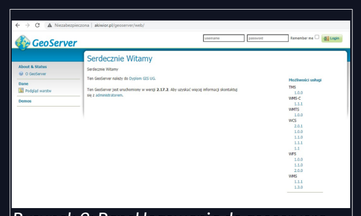
W wyniku analizy stawianych przed systemem wymagań określono niezbędną infrastrukturę sprzętową oraz konieczne oprogramowanie. Zdecydowano, się na wydzierżawienie zdalnego serwera do czasu obrony pracy dyplomowej. Do obsługi systemu postanowiono użyć jedynie oprogramowania typu opensource.

WARSTWA SPRZĘTOWA

Wymaganiem niezbędnym było autonomiczność systemu oraz jego dostępność za pomocą sieci internet. W celu spełnienia powyższego wymagania wykupiono usługę *VPS Kvm (Virtual Private Server - Kvm virtualization)* będącą dostępem do zarządzanego serwera [1 CPU, 1GB RAM, 25.0 GB HDD/SDD, STAŁE IP185.204.216.109].

WARSTWA OPROGRAMOWANIA

Na maszynie uruchomiono serwerową wersję systemu operacyjnego linux-ubuntu-20.04-x86_64. Jako serwer www (http) oraz serwer proxy uruchomiono oprogramowanie ngninx w wersji 1.18.0, ponadto zainstalowano openJDK, oraz iptables.



Rysunek 3: Panel logowania do geoservera

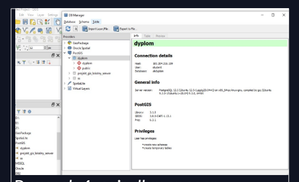
Uruchomiono geoserwer w wersji 2.16. („http://akiwior.pl/geoserver/web/"). Serwer skonfigurowano tak by udostępniał zasoby w sieci przy pomocy usług:

- **WMS:**

"http://akiwior.pl/geoserver/ows?service= wms&version=1.1.1&request= GetCapabilities"

- **WFS:**

"http://akiwior.pl/geoserver/ows?service= wfs&version=1.0.0&request= GetCapabilities"


Rysunek 4: qgis dbmanager

Uruchomiono bazę danych postgresql z dodatkiem postgis. Umożliwiono podłączenie się zdalne (sieć internet) do ww. bazy przez port 5432. Za pomocą klientów takich jak QGIS możliwe są operacje odczytu/zapisu danych bezpośrednio w bazie(rysunek 7) . Jednocześnie do bazy postgresql dostęp posiada geoserwer, który może serwować mapy utworzone na podstawie danych przechowywanych w tabelach.

APLIKACJA

Aplikacja została oparta o trzy niezależne elementy: skrypt PHP – obsługujący www, skrypt Python wykonujący analizę danych , oraz program GDAL rasterize uruchamiany przez powłokę bash.

System generuje plik rastrowy zawierający mapę rastrową zawierającą rozkład analizowanej wielkości w mieście Słupsk dostarczonej w formie pliku csv w formacie nazwa ulicy, nr_adr, dana liczbowa. Taki typ pliku źródłowego wybrano ze względu na uproszczenie obsługi systemu. Pliki csv mogą być edytowane/tworzone w każdym edytorze tekstu-arkuszu kalkulacyjnym oraz są powszechnie generowane przez bazy danych.

LICZBA MIESZKAŃCÓW-
Dane wejścia:

W wersji poglądowej użyto danych o meldunkach w mieście Słupsk. Plik zawiera dane w schemacie ulica, TERYT, nr_adr, Płeć, Ilość meldunków rozdzielone średnikiem. Dane te odbiegają strukturą od założonych w wymaganiach pod kątem struktury gdyż wygenerowane zostały przez bazę danch. Aplikację wzbogacono o funkcję dedykowaną (Listing 1) dla tych danych przekształcającą je w słownik zawierający sumę meldunków w danym adresie.

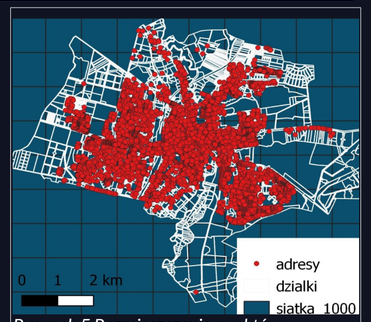
```
def meldunki_do_slownik(plik, seperator):
    f = open(plik , 'r', encoding="utf-8", newline='')
    reader = csv.reader(f, delimiter=seperator, quotechar='|')
    slownik_sum = {}
    slownik = {}
    adres = ''
    czlowiek = 0
    for row in reader:
        if row[2].replace('"','')+'--'+row[3].replace('"','') ==
adres:

            slownik[adres][row[4].replace('"','')]=int(row[5])
        else:
            adres = row[2].replace('"','')
+'--'+row[3].replace('"','')
            slownik[adres]={}
            slownik[adres][row[4].replace('"','')] = int(row[5])

    for key, item in slownik.items():
        slownik[key]['suma_M-K'] = item.get("Ż", 0) +
item.get("M", 0)
    for adres in slownik:
        slownik_sum[adres] = slownik[adres]['suma_M-K']
    return slownik_sum
```
Listing 1: Funkcja dadykowana danym meldunkowym

PUNKTY ADRESOWE:

Punkty adresowe dostępne są w formie przestrzennej (ewidencję punktów adresowych prowadzi Prezydent Miasta Słupska). Punkty adresowe – *dalej adresy* zostały pobrane z usługi WFS udostępnianej przez słupski magistrat oraz zapisane lokalnie w postaci warstwy punktowej gpkg. Następnie adresy zostały przekształcone do pliku GeoJSON za pomocą QGIS. Warstwa punktowa zawiera 7114 obiektów (stan na 24.01.2021r – rysunek 5). Plik GeoJSON ma strukturę rozbudowanego słownika (listing 2). Plik taki odczytany może być za pomocą modułu json standardowej instalacji Pythona.



Rysunek 5 Rozmieszczenie punktów adresowych

```
{
"type": "FeatureCollection",
"name": "pkt_adresowe",
"crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:EPSG::2177" } },
"features": [
{ "type": "Feature", "properties": { "mslink": 3520.0, "miejscowosc": "Słupsk", "kod": "76-200",
"poczta": "Słupsk", "przedr": "ul.", "przedrostek": "ulica", "ulica": "Podgórna", "ulica_pelna":
"Podgórna", "nr_adr": "28", "nr_adr2": null, "dzialka_id": null, "budynek_id": null, "kat": 38.0,
"just": "uc" }, "geometry": { "type": "Point", "coordinates": [ 6437430.29, 6038210.78 ] } },
--cięcie--
```
Listing 2: Wycinek geoJSON punktu adresowego

Do połączenia danych wejściowych, które podanych są przez użytkownika jako tekst, wybrano klucz oparty o kody TERYT. Kody wyznaczone są przez Główny Urząd Statystyczny. Kod TERYT jest ciągiem cyfr przypisany każdej ulicy w gminie. Dane te w formie pliku csv można pobrać ze strony GUS.

Do obsługi systemu służy 11 funkci Python umieszczonych w module „kody_teryt.py". Funkcje wywoływane są przez system za pomocą skryptu Python. Skrypt Python na serwerze uruchamiany jest przez skrypt „upload.php" pobierający oraz manpulujący plik.iem (Listing 3).

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]
["name"]);
$uploadOk = 1;
$imageFileType =
strtolower(pathinfo($target_file,PATHINFO_EXTENSION));

// Check if file already exists
if (file_exists($target_file)) {
  echo "Sorry, file already exists.";
  $uploadOk = 0;
}
--ciach--
```
Listing 3: Wycinek skryptu upload.php

Aplikacja do wykonania analizy wykorzystuje zestaw słowników Python, w których kluczami są ciągi znaków „TERYT—nr_adr". Wejściowy adres w pliku csv jest zamieniany na ww. kod w oparciu o przygotowany słownik. Analogicznie przygotowywane są punkty adresowe. Słownik kodów TERYT z GUS wzbogacany jest o słownik nazw potocznych by analiza mogła być wykonana w sposób kompleksowy.

Zestaw słowników używanych do analizy jest zdefiniowany przez administratora. Administrator dostarcza do aplikacji plik zawierający warstwę punktów adresowych; plik zawierający siatkę grid, oraz plik zawierający słowniki z kodami TERYT. Poszczególne funkcje wykonują operacje na danych. Administrator poprzez modyfikację pliku run.py może dowolnie modyfikować zakresem i sposobem wykonywania aniliz.

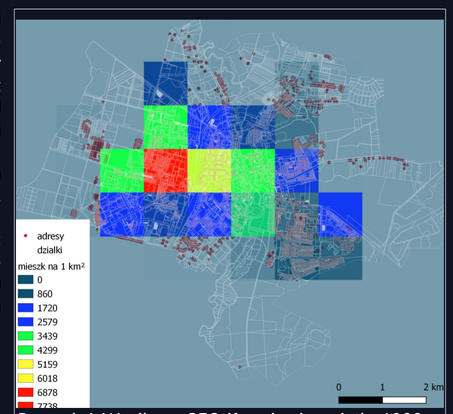
```
from funkcje_kody import *
import subprocess

# importowanie slownika z pliku csv 1 kolumna ulica, kolumna 0 to kod
teryt
teryt = teryt_load('csv/ulice_kod_full.csv', 1, 0, ';')
#importowanie z jsona punktow adresowych-geojson
adresy = load_json_dict('input_file_dir/pkt_adresowe.geojson')
#importowanie siatki 1000x1000-geojson
grid = load_json_dict('siatka_1000/siatka_slupsk_1000.geojson')
#wgranie pliku z danymi z pliku csv
#wgranie danych z meldunkow
dane = meldunki_do_slownik('csv/lud_utf8.csv';')
# dane = ulice_dane_load_csv("csv/ulica_numer_dana.csv",0, 1, 2, ';', teryt)
# dodanie teryt do adresow
adres_teryt = add_teryt_to_adresy_geojson(adresy, teryt)
#dodanie danych do adresow
adres_teryt_data = add_data_to_ulice(adres_teryt, dane,0)
#dodanie id siatki do kazdego punktu adresowego
adres_teryt_data_grid = dod_do_adres_grid_id(adres_teryt_data, grid)
#dodanie danych data do siatki
grid_data = add_data_to_gird(grid, adres_teryt_data_grid)
#zapisanie geojson nowy_slownik_tekstowy.geojson
convert_to_json(grid_data, 'nowy_slownik_testowy.geojson')
convert_to_json(adres_teryt_data, 'punty_adresowe_meldunki.geojson')
list_files = subprocess.run(["gdal_rasterize", "-ts", "1000", "1000", "-a",
"suma_danych", "-a_nodata", "999", "nowy_slownik_testowy.geojson",
"raster/ludnosc.tif"])
```
Listing 4: Skrypt run.py uruchamiany po stronie serwera

W efekcie otrzymywane są dwa pliki geoJSON. Jeden zawierający warstwę punktową adresów z przypożądkowanymi wielkościami oraz drugi zawierający siatkę grid z sumowanymi wartościami wejściowymi.

W badanym przykładzie siatka miała rozdzielczość 1000 m na 1000 m. Uzyskano ją w programie QGIS.

Ostatecznym etapem analizy jest przekształcenie danych geoJSON do postaci geotif za pomocą programu GDAL rasterize oraz udostępnieniu otrzymanego pliku tiff geoserverowi.



Rysunek 6: Wynikowy GEOtif zawierający siatkę 1000m na 1000m z liczbą mieszkańców