Projekt na zaliczenie z przedmiotu Systemy informacji przestrzennych GIS

Raport końcowy

Skład osobowy

Maciej Pacześny 187229

Wstęp

Projekt obejmował stworzenie serwisu, który pokazuje na mapie stacje benzynowych różnych sieci, z funkcjonalnością m.in. przeglądania stacji na mapie, wyszukiwania stacji według różnych kryteriów, dodawanie ocen i/lub komentarzy do stacji.

Zaimplementowane przypadki użycia wraz z krótkim opisem

Zrealizowano następujące przypadki użycia:

- przeglądanie stacji benzynowych na mapie

Na mapie umieszczane są odpowiednie kursory, odpowiadające stacjom benzynowym. Kursor jest zróżnicowany, w zależności od stacji, do której należy dana sieć. Kursory umieszczane są po odpowiednim zbliżeniu mapy, oraz są aktualizowane w trakcie przewijania mapy (pod warunkiem odpowiedniego zbliżenia). Po oddaleniu markery są usuwane.

- podgląd cen dla wybranej stacji benzynowej

Po najechaniu kursorem na marker odpowiadający stacji benzynowej pokazuje się wyskakujący “dymek” (pop-up), ze skróconymi informacjami o danej stacji benzynowej, które zawierają nazwę, ceny paliw oraz średnią ocenę użytkowników.

- wyszukiwanie stacji benzynowych

Istnieją dwa sposoby wyszukiwania: prosty i zaawansowany. Prosty sposób pozwala na wyszukanie stacji po nazwie stacji i zwraca jedynie markery na mapie, bez listy wyszukanych stacji. Wyszukiwanie proste obejmuje tylko widoczny obszar + niewielki margines. Jeżeli wyszukiwanie jest aktywne, wtedy oddalanie/ przybliżanie mapy nie wpływa na znaczniki stacji na mapie.

Przy wyszukiwaniu zaawansowanym można wyszukać po nazwie, cenie wybranego paliwa oraz średniej ocenie stacji benzynowej. Wyniki są zwrócone jako markery na mapie oraz jako lista z prawej strony ekranu. Zarówno na liście, jak i na mapie zawartych jest co najwyżej 100 pierwszych wyników. Lista zawiera skrócone informacje o stacji: nazwę stacji, średnią ocenę, odległość od środka mapy w momencie wyszukiwania oraz cenę paliwa (jeżeli typ paliwa był podany jako parametr wyszukiwania). Lista umożliwa również

sortowanie po cenie, odległości oraz średniej ocenie. Na kolejne strony wyników można przechodzić kilkając odpowiednie odnośniki - spowoduje to nie tylko pojawienie się kolejnych wyników na liście, ale również odpowiadających im markerów na mapie. Wyszukiwanie zaawansowane wyszukuje w dużo większym obszarze, niż jest widoczny.

- dodawanie ocen i komentarzy do stacji benzynowych (zalogowany użytkownik)

Zalogowany użytkownik może dodać ocenę i/lub komentarz do wybranej stacji benzynowej. Po kliknięciu na marker stacji pojawia się okno komentarzy, gdzie można przeglądać komentarze i oceny innych użytkowników oraz dodać swój komentarz i/lub ocenę (jedno z dwóch nie może być puste). Istnieje też możliwość usunięcia komentarza dodanego przez siebie.

- zapisywanie danych o swoim samochodzie (model, spalanie) (zalogowany użytkownik)

Zalogowany użytkownik może przejść do swojego konta, gdzie zawarte są informacje o jego koncie (np. nazwa użytkownika i adres email), oraz dane dodanych przez niego samochodów (marka, model, typ paliwa, spalanie). Użytkownik może dodać nowe samochody, lub usunąć istniejące.

Techniczna realizacja projektu

Na projekt składają się: strona www (front-end) oraz back-end do przetwarzania i odpowiedzi na zapytania ze strony www. Front-end został zrealizowany z wykorzystaniem HTML, CSS i JavaScript, bez żadnych dodatkowych bibliotek, z wyjątkiem biblioteki JavaScript i CSS OpenLayers. Back-end został zrealizowany w Pythonie z użyciem frameworka webowego Flask. Jako bazę danych wykorzystano Sqlite, a do komunikacji z bazą użyto frameworka ORM SqlAlchemy.

Back-end zrealizowano według wzorca architektonicznego Model-View-Controller, ale z nieco odmienną terminologią, przyjętą we frameworku Flask - Model-View-Template:

- **model** - model to klasa reprezentująca pewien byt w domenie biznesowej. W tym projekcie domeną są stacje benzynowe, a modelami: użytkownik, stacja benzynowa, paliwo z ceną, komentarz z oceną. Jest to główna jednostka informacji w projekcie. Odpowiada modelowi z MVC.
- **view** - tutaj trafiają żądania użytkownika. Widok odpowiada za przetwarzanie i odpowiedź na żądania użytkownika. W toku przetwarzania pobiera model i przekazuje pobrany model do szablonu (template), żeby wyrenderować odpowiedź - jeżeli odpowiedź powinna zostać zwrócona jako strona HTML. Niekiedy widoki używają również formularzy - są to klasy służące do opisania formularzy wykorzystwanych na stronie do wprowadzenia danych. Odpowiadają za logikę formularza - m.in. za jego walidację. Formularze przekazane do szablonu renderowane są jako formularze HTML. W tym projekcie zaimplementowano również niewielkie REST API, które zwraca odpowiedź w formacie JSON, w takim przypadku szablon nie jest renderowany. Widok jest odpowiednikiem kontrolera z MVC.

- **template** - szablon strony HTML, ta część wzorca jest odpowiedzialna za renderowanie stron WWW z wykorzystaniem szablonów oraz przekazanych przez kontroler danych, i zwrócenie wyrenderowanego szablonu. Jest to odpowiednik widoku z MVC.

Oprócz MVC nie wykorzystano innych wzorców projektowych lub architektonicnych.

Tam, gdzie było to możliwe zastosowano klasyczne rozwiązanie renderowania szablonu po stronie serwera i przesyłania gotowej strony do klienta. Takie rozwiązanie jednak ma jedną dużą wadę - cała strona ulega odświeżeniu. Z tego powodu zastosowano to tylko tam, gdzie można było na to pozwolić - na podstronie rejestracji, logowania czy na podstronie konta użytklownika. Na głównej stronie, która zawiera mapę i markery, odświeżenie strony spowoduje odświeżenie całej mapy i usunięcie wszystkich markerów. Dlatego na głównej stronie koniecznym było odświeżenie wybranych elementów strony, ale bez odświeżania całej strony. Zrobiono to w ten sposób, że zapytania pod niektóre adresy zwracają jedynie część HTMLa, która ma zostać umieszczona w wybranym divie. Dla takich elementów kod po stronie Javascript wysyła odpowiednie żądanie do serwera, a zwróconą odpowiedź umieszcza w odpowiednim divie - w ten sposób div zostanie odświeżony bez odświeżenia całej strony. Takie rozwiązanie zastosowano dla wyskakujących dymków (pop-up), okna komentarzy, okna wyszukiwania zaawansowanego i okna wyników wyszukiwania zaawansowanego.

Największym problemem okazało się dodawanie komentarzy tak, żeby strona nie uległa odświeżeniu, a jednocześnie użytkownik zobaczył komentarz dodany przez siebie. Rozwiązano to tak, że formularz wysyła komentarz na serwer, a odpowiedź (komentarze, łącznie z nowym komentarzem) umieszcza w znaczniku iframe, który jest ukryty. Funkcja po stronie Javascript pobiera kod z iframe i wstawia do diva, w którym są komentarze.

Użyte rozwiązania geoprzestrzenne

Poniżej opisano użyte rozwiązania geoprzestrzenne oraz funkcje biblioteki OpenLayers, wykorzystane od ich implementacji:

- wyznaczanie punktu na mapie ze współrzędnych geograficznych - do tego celu wykorzystano funkcję fromLonLat, która jako argumenty przyjmuje długość i szerokość geograficzną.
- umieszczanie markerów na mapie - ze współrzednych geograficznych stacji benzynowej obliczono punkt na mapie, w którym należy umieścić marker. Następnie w tym miejscu umieszczano cechę (Feature), która zawiera punkt (Point); cecha zawiera ikonę odpowiednią dla stacji danej sieci
- wyznaczanie środka mapy - w celu wyznaczenia środka mapy wywołano funkcję getCenter dla aktualnego widoku (View) mapy, a następnie uzyskaną wartość zamieniono na współrzędne geograficzne za pomocą funkcji toLonLat.
- wyznaczanie promienia mapy - w celu wyznaczenia, które stacje benzynowe znajdują się w okręgu o promieniu N metrów od środka mapy, potrzebne jest wyznaczenie promienia. Żeby to zrobić, posłużono się funkcją calculateExtent, która oblicza zasięg (Extent) dla aktualnego widoku mapy. W OpenLayers klasa Extent reprezentuje

prostokątny obszar mapy, określony przez jego współrzędne lewego górnego i prawego dolnego rogu. Jest to zazwyczaj używane do określenia widocznego obszaru mapy lub obszaru granicznego dla cechy lub warstwy. Mając zasięg, można wybrać dwa punkty na mapie do obliczenia odległości pomiędzy nimi (np. punkt w lewym, dolnym rogu i punkt w prawym, dolnym rogu), żeby wyznaczyć szerokość aktualnego widoku mapy. Szerokość obliczana jest za pomocą funkcji getDistance, następnie dzielona na dwa i zaokrąglana do najbliższej liczby całkowitej w dół. Tak obliczona wartość jest przyjmowana jako promień.

Oprócz biblioteki OpenLayers, implementacje rozwiązań geoprzestrzennych znalazły swoje zastosowanie również po stronie serwera, gdzie zaszła potrzeba obliczenia odległości pomiędzy dwoma punktami współrzędnych przy ustalaniu, które stacje benzynowe znajdują się w okręgu o promieniu N metrów od podanego punktu. Okazało się to problemem ze względu na zastosowaną bazę danych - baza Sqlite nie zawiera funkcji geoprzestrzennych. Ostatecznie obliczanie odległości zrealizowano w czystym SQL, z użyciem wzoru Harvesine.

Realizacja wymagań

W projekcie postawiono i zrealizowano następujące wymagania:

- strona www wyświetla się poprawnie na dowolnej, nowoczesnej przeglądarce internetowej na komputerze osobistym (dostosowanie wyglądu do obsługi urządzeń przenośnych nie jest przewidziane)
- strona www zgodna ze standardami HTML/XHTML, CSS konsorcjum W3
- back-end stworzony w oparciu o wzorzec MVC
- hasła użytkowników przechowywane w bazie danych w bezpieczny sposób (hash + salt)
- baza danych zabezpieczona przed atakami SQL Injection
- zróżnicowanie widoku graficznego markera na mapie w zależności od sieci, do której należy dana stacja benzynowa

Jedno postawione wymaganie nie zostało zrealizowane:

- bezpieczne uwierzytelnianie użytkownika (oparte na tokenie)

Zamiast uwierzytelniania opartego na tokenie zaimplementowano uwierzytelnianie oparte na sesji: po udanym zalogowaniu klient otrzymuje informacje o ID swojej sesji, wygenerowane przez back-end, i zapisuje je do pliku cookie. W następnych żądaniach do strony klient posługuje się plikiem cookie w celu uwierzytelnienia, a back-end używa tych informacji do uzyskania informacji o kliencie. Co do uwierzytelniania opartego na tokienie, to jest ono szczególnie użyteczne w przypadku REST API, które nie obsługuje plików cookie, a informacje uwierzytelniające muszą być przesyłane w każdym żądaniu, o ile dany endpoint API tego wymaga. W tym projekcie jest tylko jeden endpoint REST API, który służy do pobierania informacji o stacjach benzynowych, i nie wymaga uwierzytlenienia. Z tego powodu nie zaimplementowano uwierzytelniania opartego na tokenie.