Projekt SPDB

Temat: Nawigacja rowerowa

Student: Volodymyr Ostruk, 255356
Prowadzący: dr inż. Robert Bembenik

1. Dokładna treść tematu i cel projektu

Celem projektu jest *stworzenie aplikacji* pozwalającej na wyznaczanie tras przejazdu rowerem miejskim przy jednoczesnej minimalizacji kosztu przejazdu. Aplikacja powinna pobierać informacje dot. początku i końca trasy (w przypadku, gdy użytkownik jest w pobliżu stacji powinna sama sugerować początek trasy w bieżącej lokalizacji użytkownika) i wyznaczać trasę przebiegającą w miarę możliwości istniejącymi ścieżkami rowerowymi. Minimalizacja kosztu przejazdu: jeśli chcemy przejechać długą trasę, to nawigacja powinna prowadzić nas przez inne stacje pozwalające na zwrot i ponowne wypożyczenie roweru przed upływem czasu darmowego wypożyczenia.

2. Założenia wstępne oraz doprecyzowanie projektu (wymagania)

- Użytkownik musi mieć możliwość wyszukania miejsc startowych i docelowych za ich nazwą, podając początkowe litery nazwy i wybierając z listy podobnych miejść.
- Dla wyznaczenia trasy użytkownik powinien podać punkt początkowy i punkt końcowy podróży w postaci dwóch liczb (lab, lng) lub wybierając te punkty z listy punktów dostępnych
- Aplikacja powinna pozwalać na wyznaczanie lokalizacji użytkownika odpowiednim przyciskiem (np. gdy nie wiemy gdzie dokładnie jesteśmy)
- Aplikacja powinna wyszukiwać dowolne drogi w Warszawie, z dowolną ilością “przesiadek” lub jeżeli przesiadka nie jest możliwa z odpowiednim pokazaniem tego.
- Aplikacja powinna pokazywać numer oraz nazwę każdego punktu zmiany roweru na mapie
- Aplikacja powinna uwzględniać czas potrzebny na wymianę roweru na stacji na inny (założenie - 2 minuty)
- Aplikacja powinna przy możliwości budować drogę używając ścieżek rowerowych
- Aplikacja powinna omijać przejścia podziemne i schody (zakładamy, że użytkownik nie chce prowadzić lub nosić roweru schodami)
- Aplikacja buduje drogę od najbliższych stacji wypożyczania rowerów. Droga od wskazanego użytkownikiem miejsca do najbliższej stacji pokazywana jest linią prostą.
- Aplikacja powinna być łatwo dostępną na urządzeniach mobilnych, ponieważ korzystać z niej będziemy na mieście (np. nagła potrzeba przejazdu z jednego miejsca do innego)
- Aplikacja powinna pokazywać odległość oraz czas potrzebny na podołanie użytkownikiem danej odległości używając darmowych rowerów.

3. Wykorzystane technologie oraz sposób realizacji projektu

Projekt był podzielony na dwie części logiczne: cześć frontendu i backendu. Komunikacja pomiędzy odbywa się zgodnie ze stylem architektonicznym REST, czyli frontend wysyła żądania za pomocą protokołu HTTP do backendu i dostaje odpowiedź w formacie JSON.

Przykładowa komunikacja pomiędzy modułami opisana została poniżej:

Wyszukiwanie trasy za pomocą url odbywa się pod adresem : `/route GET`

Parametry dla wyszukiwania drogi:

- `beg_lat` - wymagany
- `beg_lng` - wymagany
- `dest_lat` - wymagany
- `dest_lng` - wymagany
- `beginning` - nazwa lokacji początkowej (zwracana bez zmian w odpowiedzi)
- `destination` - nazwa lokacji końcowej (zwracana bez zmian w odpowiedzi)

Odpowiedź na zapytanie ma następującą postać:

```
{
    "beginning": STRING // nazwa lokacji podana w argumencie
    "destination": STRING, // nazwa lokacji podana w argumencie
    "beg_coord": [FLOAT, FLOAT],
    "dest_coord": [FLOAT, FLOAT],
    "path":
        {
        "points":
            {
     ł  ę   "coordinates": [[FLOAT, FLOAT]] // lista par wspó rz dnych
w z ów
            },
ż           "time": INT, // przewidywany czas trwania podró y (ms)
            "distance": INT, // dystans (m)
            },
    "stations":
        [
ę               "location": [FLOAT, FLOAT], // wspó rz dne stacji
                "name": STRING, // nazwa stacji
                "number": INT // numer stacji
        ]
}
```

Przykładowe zapytanie dla wyszukiwania drogi:

```
/route?beg_lat=1.0&dest_lat=1.0&beg_lng=1.0&dest_lng=1.0
```

Przykładowa odpowiedź na zapytanie:

```
{
    "destination": "asd",
    "dest_coord": [FLOAT, FLOAT],
    "path":
        {
            "points":
                {
                    "coordinates":
                        [
                            [52.235706, 20.996794],
                            [52.248848, 21.005457],
                            [52.230217, 21.012592],
                            [52.231426, 21.021326],
                            [52.23935, 21.016813]
                        ]
                },
            "time": 234000,
            "distance": 30,
        },
    "beginning": "dsa",
    "beg_coord": [FLOAT, FLOAT],
    "stations":
        [
            {
                "location": [52.133163, 21.07482]
                "name": "ul. Wąwozowa- ul.Rosoła"
                "number": 6446
            },
            {
            "location": [52.162342, 21.038431]
            "name": "SGGW I"
            "number": 6341
            }
        ],
}
```

3.1 Backend

Backend, czyli logika aplikacji został zrealizowany z użyciem języka haskell i jego narzędzi wspomagających (cabal, stack, biblioteki hackage).

Dla wyszukiwania najlepszych połączeń korzystamy z pakietu haskell *DataGraph.AStar*, realizującego algorytm A*. Logika dotycząca algorytmu została umieszczona w pliku Freebike/Graph.hs

Funkcja Astar ma następującą specyfikację:

aStar - this function computes an optimal (minimal distance) path through a graph in a best-first fashion, starting from a given starting point.

`:: (Hashable a, Ord a, Ord c, Num c)`	
`=> (a -> HashSet a)`	The graph we are searching through, given as a function from vertices to their neighbours.
`-> (a -> a -> c)`	Distance function between neighbouring vertices of the graph. This will never be applied to vertices that are not neighbours, so may be undefined on pairs that are not neighbours in the graph.
`-> (a -> c)`	Heuristic distance to the (nearest) goal. This should never overestimate the distance, or else the path found may not be minimal.
`-> (a -> Bool)`	The goal, specified as a boolean predicate on vertices.
`-> a`	The vertex to start searching from.
`-> Maybe [a]`	An optimal path, if any path exists. This excludes the starting vertex.

Pełny algorytm został zrealizowany natomiast na takiej zasadzie, że od danego miejsca startowego wybierane są tylko stacje w odległości nie większej niż 20 min (tzw. sąsiedzi - funkcja getAllowedTimeNeighbours przekazywana jako pierwszy argument do funkcji aStar), po czym iteracyjnie jest budowana droga na takiej samej zasadzie (z uwzględnieniem czasu wymiany roweru bikeChangeTime = 2 min). Funkcja generateRoute opakowuje łogike budowania drogi za pomocą aStar..

Lista stacji veturillo może być pobrana z oficjalnej strony w formacie xml (nextbike-live.xml). Cały graf przejść, po którym wyszukujemy drogi został uzyskany parsowaniem pliku json (funkcja parseJSONFromFile, z użyciem biblioteki haskell Aeson), który można uzyskać na portale hackathonu bihapi.(plik precompute.hs)

3.2 Frontend

Frontend, z myślą o mobilności aplikacji został napisany w javaScripcie z użyciem frameworka Apache Cordova, napisanego w Node.js oraz używanego dla łatwego tworzenia frontendu dla różnych platform (www, android, ios, ...)

Za pomocą takiej architektury możemy wymieniać frontend na dowolny inny (dla aplikacji na urządzeniach mobilnych, tabletach, urządzeniach IoT) korzystając cały czas z tego samego backendu (API).

Frontend działa na zasadzie

Wszystkie główne pliki javaScriptu odpowiadające za łogikę frontendu umieszczone są w podkatalogu www/js.
Pliki

- *autocomplete.js,
- utils.js,
- templates.js

są plikami wspomagającymi (np. uzupełnienie wyboru).
Natomiast pliki

- app.js,
- backend.js,
- routing.js

są plikami głównymi nadzorującymi aplikacje i współpracującymi bezpośrednio z backendem.

Skorzystałem też z bibliotek bootstrap i leaflet wspomagających interfejs użytkownika (np. leaflet ma dobrą obsługę dla map).

Dane frontendu przechowujemy natomiast w pamięci podręcznej javaScriptu (localStorage).

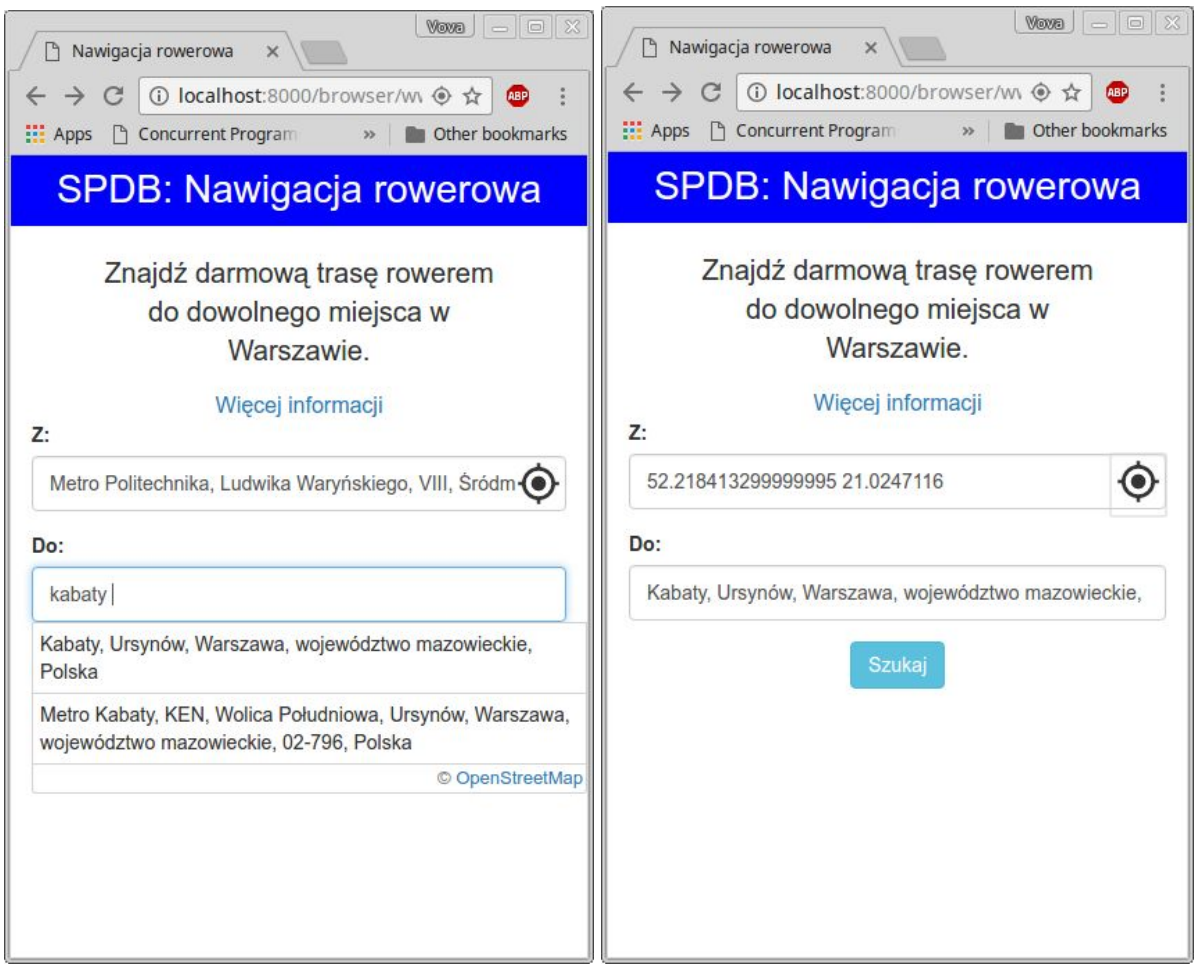
Za pomocą frameworku cordova łatwo możemy rozszerzać nasz frontend do innych platform.
Wystarczy wykonać polecenie:

```
cordova platform add android/ios/ubuntu...
```

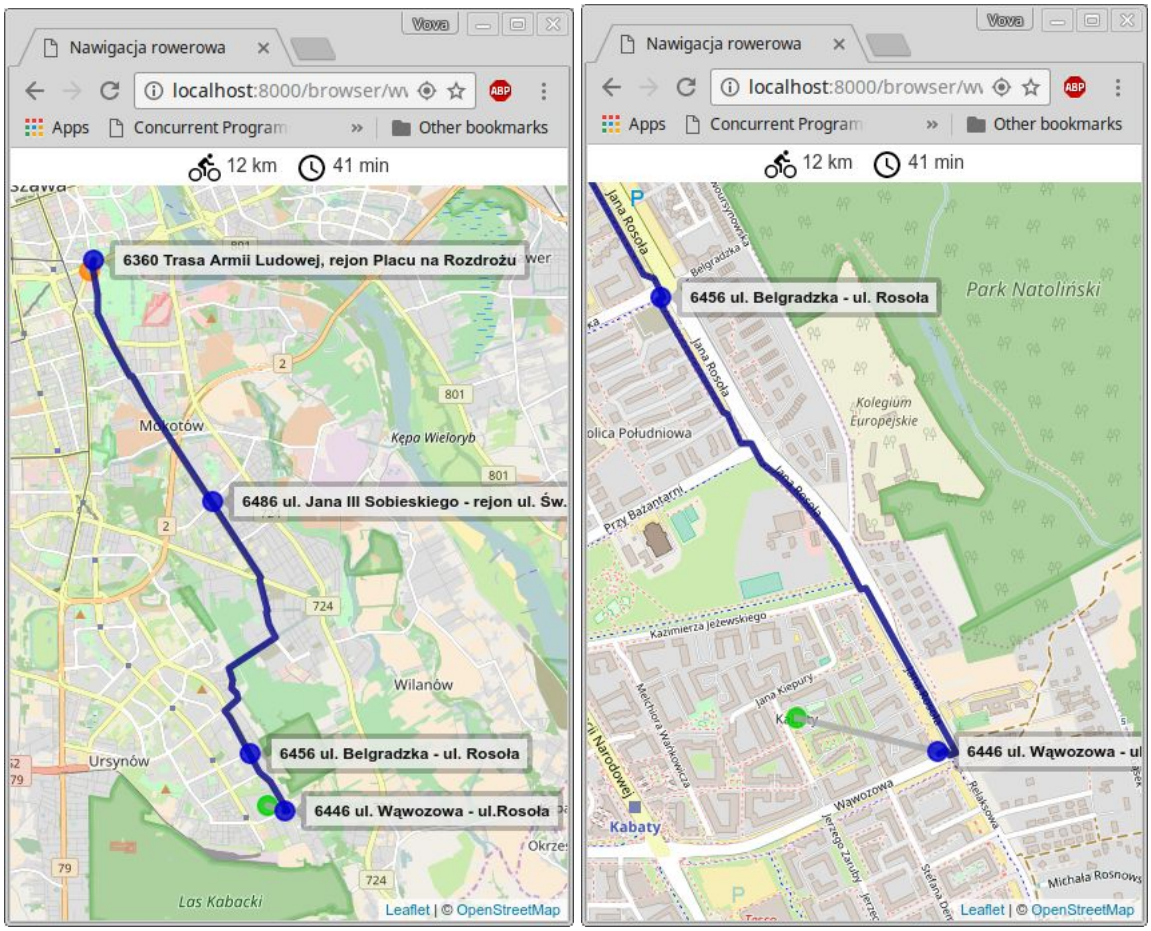
4. Opis działania

Po uruchomieniu aplikacji dostępne będą dwa typy usług: API pod adresami serwowanymi z backendu oraz interfejs użytkownika pod adresem serwowanym frameworkiem cordova (port za każdym uruchomieniem może być inny).

Przechodząc pod adres interfejsu wskazanego cordovą widzimy widok domowy aplikacji, gdzie użytkownik może podać początek i koniec trasy. Początek może być również wskazany z użyciem przycisku wyznaczenia lokalizacji. Jeżeli użytkownik wprowadza nazwy miejscowości (bo jest też możliwe wprowadzenie koordynat geo), wyświetlana jest lista podpowiedzi nazw, pobierana z serwisu OpenStreetMap.



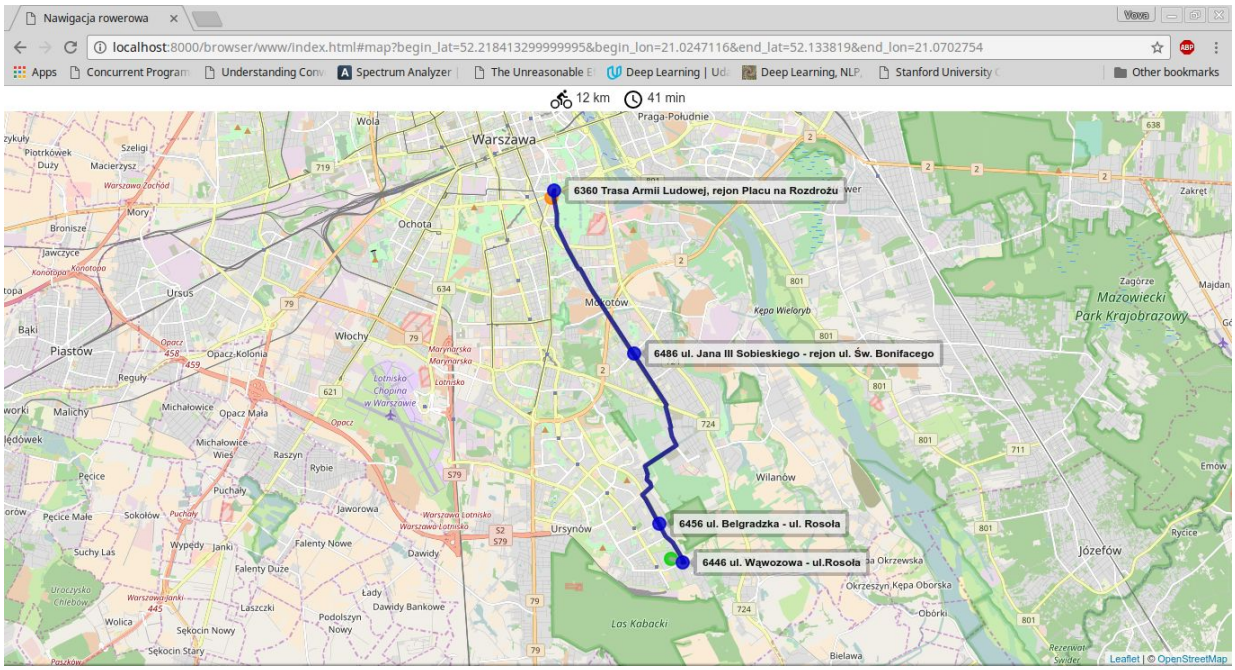
Dalej użytkownik może nacisnąć przycisk szukaj po czym na kolejnym widoku jest wyświetlana mapa z wyznaczoną drogą wraz z miejscami wymiany rowerów:



Jak widać, na górze też jest pokazywana odległość i czas potrzebny na jej podołanie. Korzystamy z interfejsu openStreetMaps, ponieważ jest dokładniejszy i wszystkie ścieżki rowerowe naniesione są linią przerywaną. Oprócz tego w razie potrzeby sami mamy możliwość edytowania i dodawania elementów na mapie (potrzebna jest rejestracja).

Punkty wymiany rowerów oraz drogi rowerowe pokazywane są kolorem niebieskim, punkty startu i końca natomiast kolorem zielonym i czerwonym. Linie proste od początku i końca do punktów wymiany rowerów pokazane są kolorem szarym.

Poniżej widać jak aplikacja oraz przekazywane parametry url wyglądają na szerokim monitorze:



5. Proces instalacji

Dla poprawnego dzialania aplikacji backendu potrzebne będą do wykonania następujące czynności (przetestowana platforma - Linux Mint):

- Pliki projektu (frontend i backend)
- Zainstalowany haskell (https://www.haskell.org/downloads) lub haskell platform (https://www.haskell.org/platform/)
- Zainstalowany cabal (https://www.haskell.org/cabal/download.html)
- Zainstalowany stack (https://docs.haskellstack.org/en/stable/README/)

5.1 Backend

Jeżeli powyższe zależności są spełnione my możemy przechodzić do etapu budowania aplikacji:

- stack solver
- stack setup

Jeżeli nie ma żadnych błędów kontynuujemy:

- stack build
- stack exec _usługa_

5.2 Frontend

Uruchomić frontend jest o wiele łatwiej, wystarczy:

- Zainstalować npm
- Zainstalować cordova
- Uruchomić cordova build
- Uruchomić cordova serve browser
- Przejść na wskazany link

6. Wnioski

Podsumowując, główny cel projektu wyznaczony prowadzącym został zrealizowany. Nauczyłem się budować aplikacje z obsługiwaniem map i dróg na nich, obsługiwać api OpenStreatMap, także nauczyłem nowego frameworku do frontendu cordova. Największymi wyzwaniami w projekcie było uzyskanie danych w odpowiednim formacie (budowanie grafu) i połączenie modułów dla poprawnego działania.
Mam nadzieję, że aplikacja będzie pomagała studentom mieć więcej pieniędzy, oszczędzając na przejazdach rowerami.

7. Kolejne kroki

- Sprawdzenie o dostępności rowerów w dany moment i poinformowanie o tym użytkownika
- Nałożenie dodatkowych reguł na wyszukiwanie drogi (np. użycie tylko ścieżek rowerowych, minimalna ilość wymian rowerem, droga przez park i t.d.)
- Podpięcie przestrzennej bazy danych do aplikacji (np. przy skalowaniu)
- Wygenerowanie i dopasowanie interfejsu użytkownika do innych platform mobilnych (ios, android). Obecnie aplikacja dostępna jest jako serwis www (zoptymalizowany pod urządzenia mobilne)

8. Źródła

https://www.haskell.org/documentation
https://hackage.haskell.org/packages/
https://cordova.apache.org/docs/en/latest/
http://www.bihapi.pl/
http://www.bihapi.pl/media/uploads/dokumentacja-api-bihapi/
http://rozie.blox.pl/2013/06/Nextbike-monitorowany.html
https://github.com/Informatic/nextbike2json