Szkoła Główna Gospodarstwa Wiejskiego
w Warszawie
Wydział Zastosowań Informatyki i Matematyki

Bartosz Matyjasiak
185117

Projekt i implementacja aplikacji mobilnej wyświetlającej aktualne lokalizacje autobusów oraz tramwajów w Warszawie

TODO

Praca dyplomowa inżynierska
na kierunku – Informatyka

Praca wykonana pod kierunkiem
dr. hab. inż. Leszek Chmielewski, prof. SGGW
Instytut Informatyki Technicznej
Katedra Sztucznej Inteligencji

Warszawa, 2020[1]

[1]Dokument skompilowano z klasą `SGGW-thesis` w wersji 1.05. Aktualną wersję klasy można pobrać ze strony `http://stud.lchmiel.pl` → Seminarium dyplomowe.

Oświadczenie promotora pracy

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia tej pracy w postępowaniu o nadanie tytułu zawodowego.

Data ..... Podpis promotora pracy .....

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej, w tym odpowiedzialności karnej za złożenie fałszywego oświadczenia, oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami prawa, w szczególności z ustawą z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. Nr 90 poz. 631 z późn. zm.)

Oświadczam, że przedstawiona praca nie była wcześniej podstawą żadnej procedury związanej z nadaniem dyplomu lub uzyskaniem tytułu zawodowego.

Oświadczam, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną. Przyjmuję do wiadomości, że praca dyplomowa poddana zostanie procedurze antyplagiatowej.

Data ..... Podpis autora pracy .....

Streszczenie

TODO

TODO

Słowa kluczowe – TODO, TODO, TODO

Summary

TODO

TODO

Keywords – TODO, TODO, TODO

Spis treści

1 Wstęp

W dużych miastach komunikacja miejska jest kluczowym aspektem dla mieszkańców. Niestety duże miasta, w tym Warszawa, boryka się z korkami, wypadkami, robotami drogowymi i innymi przez co autobusy czy tramwaje często nie jeżdżą według rozkładu jazdy. Dlatego dobrą informacją dla podróżującego jest lokalizacja GPS autobusu lub tramwaju. Warszawa udostępnia takie dane w projekcie "Otwarte Dane" jednak są one w postaci nieczytelniej dla przeciętnego człowieka. Rozwiązaniem może być aplikacja mobilna, dzięki której użytkownik będzie widział na mapie kiedy dokładnie przyjedzie autobus lub tramwaj.

1.1 Założenia

Aplikacja powinna:

- Pokazywać aktualne pozycje autobusów i tramwajów na mapie
- Pokazywać pozycje przystanków na mapie
- Udostępniać rozkłady jazdy na każdym z przystanków
- Umożliwiać na dodanie linii autobusówej lub tranwajowej do ulubionych
- Umożliwiać na dodanie przystanku do ulubionych
- Wspierać dwa motywy:
  - Jasny
  - Ciemny

1.2 Grafiki koncepcyjne

2 Implementacja

Do implementacji wybrałem React-Native stworzony przez Facebook. Pozwala on na zrobienie aplikacji na telefony z systemem Android oraz iOS przy pomocy jednego kodu źródłowego napisanego w języku JavaScript XML (w skrócie JSX). Skupie się jednak na wersji aplikacji na system Android. Wybrałem także moduł react-native-maps, który jest odpowiedzialny za wyświetlanie mapy Google oraz zarządzanie nią.

2.1 Publiczne API Warszawy

Miasto udostępnia dane w postaci publicznego API. Z pośród wielu punktów końcowych tego API są dostępne:

- Pozycje pojazdów danej linii
- Zbiór linii, które odjeżdzają z danego przystanku
- Rozkład jazdy dla danej linii z danego przystanku
- Zbiór wszystkich przystanków

Co ważne pozycje pojazdów są aktualizowane co 10 sekund i też z taką częstotliwością będą aktualizowane w aplikacji. Wszystkie z wymienionych punktów końcowych API zaimplementowałem w klasie `WarsawAPI`.

2.2 Komponent GlobalContextProvider

W React-Native wszystkie elementy, które wyświetlają się na ekranie są komponentami. Komponenty pomiędzy sobą są połączone relacją rodzic-dziecko. Niesie za sobą to pewne problemy. Jednym z nich jest tzw. prop-drilling. By tego uniknąć użyłem kontekstu dostępnego w React i stworzyłem komponent `GlobalContextProvider` odpowiedzialny za całą logike aplikacji. Komponent ten przechowuje zmienne:

- Zbiór wszystkich przystanków
- Zbiór ulubionych linii
- Zbiór ulubionych przystanków
- Aktualny wyświetlany region mapy

- Pozycje radaru oraz jego promień
- Zaznaczony przystanek lub pojazd

Oraz funkcje do modyfikacji tych zmiennych. Komponent też przechowuje referencje do komponentu mapy oraz udostępnia funkcje od sterowania nią:

- Dopasowanie regionu mapy do grupy przystanków
- Wycentrowanie mapy na lokalizacji GPS użytkownika
- Zaznaczenie pojazdu lub przystanku i wycentrowanie mapy na zaznaczoneniu

Jednak nie umieściłem w nim logiki aktualizowania pozycji pojazdów gdyż każda zmiana stanu tego komponentu powoduje przerysowanie wszystkich komponentów GlobalContext.Consumer, a wraz nim wszystkich jego dzieci. Przez to, że ten komponent jest używany w wielu miejscach to każda aktualizacja pozycji pojazdów, a ta jest co 10 sekund, powodowałaby przerysowanie całej aplikacji. To wiązałoby się z utratą na szybkości działania aplikacji.

2.3 Aktualizacja pozycji pojazdów

By aktualizacja przebiegała sprawnie wraz z wyświetlaniem logike aktualizacji umieściłem w komponencie `GMap`. Jest to komponent, który jako dziecko posiada tylko komponent map. Jest to ważne bo gdy tylko zmieni się stan komponentu `GMap`, a ten będzie się zmieniał co 10 sekund, to wywoła to przerysowanie tylko komponentu map. W tym komponencie zaimplementowałem funkcje, która:

1. Dla każdej ulubionej lub wykrytej przez radar 2.4 linii są pobierane pozycje pojazdów tych linii
2. Jako pojazdy do wyświetlenia są brane pod uwage tylko te pojazdy, które są z linii ulubionej lub w promieniu radaru oraz czas wysłania sygnału GPS nie jest starszy niż 6 minut

Funkcja ta jest uruchamiana co 10 sekund za pomocą funkcji `setTimeout` wbudowanej w język JavaScript.

2.4 Pinezki przystanków

Wiedza o tym gdzie znajduje się przystanek jest bardzo ważna dla użytkownika. Jednak nie można ich wszystkich wyrenderować na mapie gdyż jest ich 6449 w sieci ZTM. Taka ilość

praktycznie spowodowała by, że aplikacja nie nadawałaby się do użytku. Dlatego zoptymalizowałem to w następujący sposób.

Stworzyłem skrypt, który grupuje otrzymane przystanki z API Warszawy po numerze zespołu przystanka oraz wylicza średnią pozycje przystanków grupy. Wynik zapisuje do pliku `.json`. Ze względów optymalizacyjnych plik ten hostuje w serwisie Firebase. Na tym serwisie też stworzyłem punkt końcowy API, który zwraca ten plik. Aplikacja przy starcie pobiera ten plik.

2.5 Radar

Głównym celem radaru jest pokazywanie pinezek pojazdów linii z poza ulubionych. Ogranicza on tez ilość pinezek do narysowania. Autobusów i tramwajów w Warszawie jest zbyt duża ilość by efektywnie pokazać je wszystkie na raz na mapie dodałem do aplikacji funkcje radaru.

Działanie radaru jest następujące:

1. Użytkownik za pomocą przycisku w prawym dolnym rogu ustawia pozycje radaru na środku regionu mapy, który jest aktualnie wyświetlany
2. Dla każdego z grup przystanków jest sprawdzane, czy średnia pozycja grupy jest w promieniu radaru
3. Jeśli tak to wszystkie linie z każdego przynkanka danej grupy są dodawane do zbioru linii radaru. Wszystkie elementy tego zbioru są unikalne.

Podczas implementacji zauważyłem problem. Jeśli w granicach radaru jest n przystanków to tyle samo będzie zapytań do API Warszawy o linie jakie odjeżdzają z danego przystanka. Czas wysłania i odbioru około średnio 40 zapytań był bardzo długi. Dlatego do skryptu i pliku opisanego w 2.4 dodałem pobieranie dla każdego przystanka wszystkich linii oraz zapis ich do pliku wynikowego.

2.6 Ukrycie kluczy API w kodzie

W aplikacji używam dwóch API, Warszawy oraz map Google. Każde z nich wymaga klucza API. Te klucze powinny pozostać prywatne i niewidoczne w kodzie aplikacji. Dlatego by ukryć klucze stworzyłem plik `.env` w lokalizacji domowej projektu, w którym zdefiniowałem dwie zmienne środowiskowe: `WARSAW_API_KEY` oraz `GOOGLE_MAPS_API_KEY` o odpowiednich wartościach.

Następnie w kodzie posługiwałem się zmienną `BuildConfig` z modułu `react-native-config` by otrzymać klucz API Warszawy, a klucz map Google, który wymaga umieszczenia w pliku `AndroidManifest.xml`, przy pomocy modułu `react-native-dotenv`.

Po wykonaniu tych operacji można bezpiecznie użyć systemu kontroli wersji takiego jak "git" i serwisów jak Github do udostępniania kodu. Należy też dopisać lokalizacje pliku `.env` do pliku `.gitignore`. Teraz by było możliwe zbudowanie aplikacji w trybie debug należy wypełnić plik `.env.example` własnymi kluczami i zmenić jego nazwe na `.env`.

3 Użycie

4 Podsumowanie i wnioski

5 Bibliografia

Wyrażam zgodę na udostępnienie mojej pracy w czytelniach Biblioteki SGGW w tym w Archiwum Prac Dyplomowych SGGW.

. . . . . .
(czytelny podpis autora pracy)