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

Project and implementation of mobile application displaying present locations of busses and trams in Warsaw

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

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

TODO

Słowa kluczowe – TODO, TODO, TODO

Summary

Project and implementation of mobile application displaying present locations of busses and trams in Warsaw

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 problemami przez co autobusy czy tramwaje często nie jeżdżą dokładnie według rozkładu jazdy. Dlatego dobrą informacją dla podróżującego jest lokalizacja GPS autobusu lub tramwaju. Warszawa udostępnia takie dane w projekcie "Otwarte Dane" [6] 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.

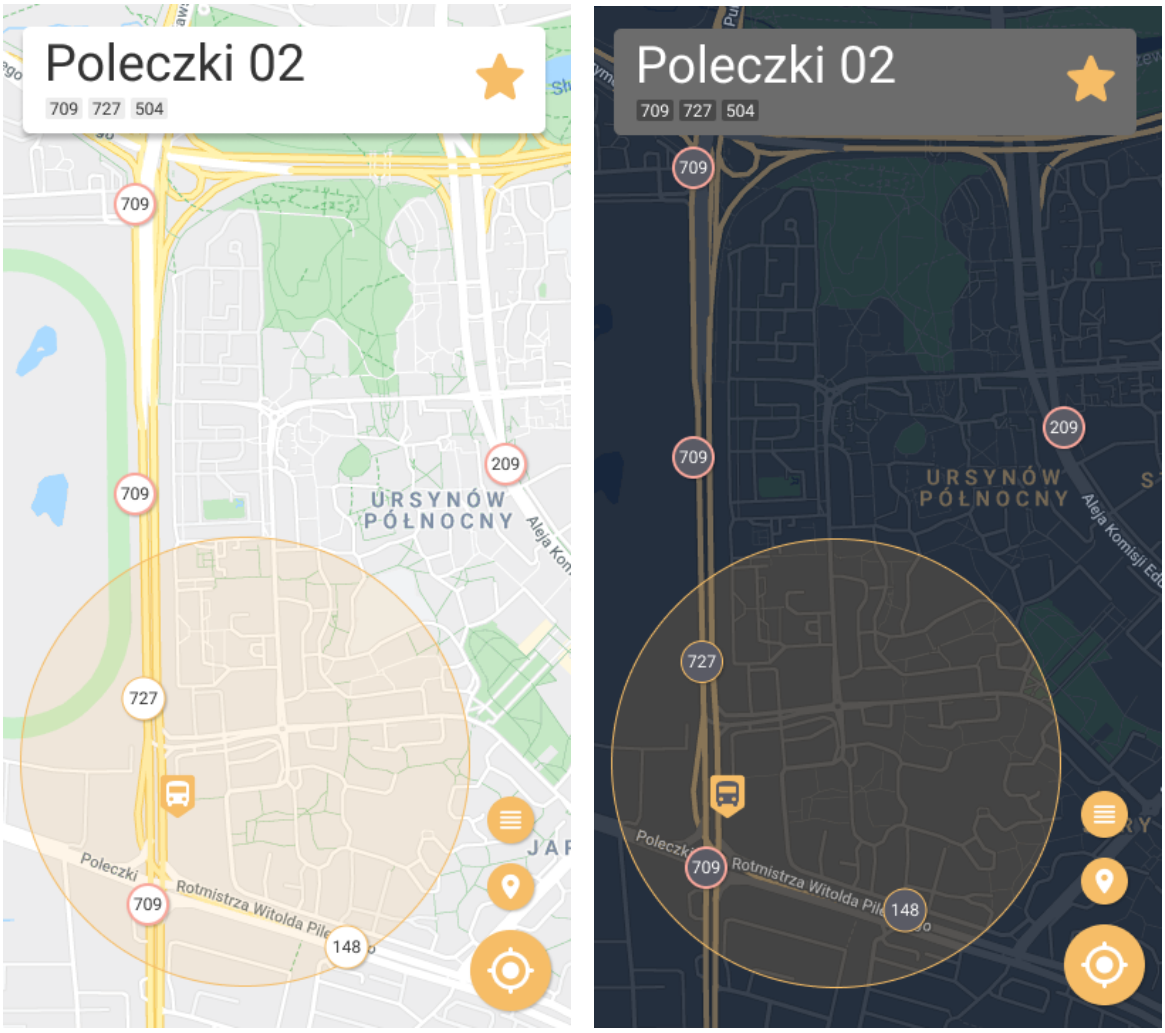
1.1 Założenia

Aplikacja powinna:

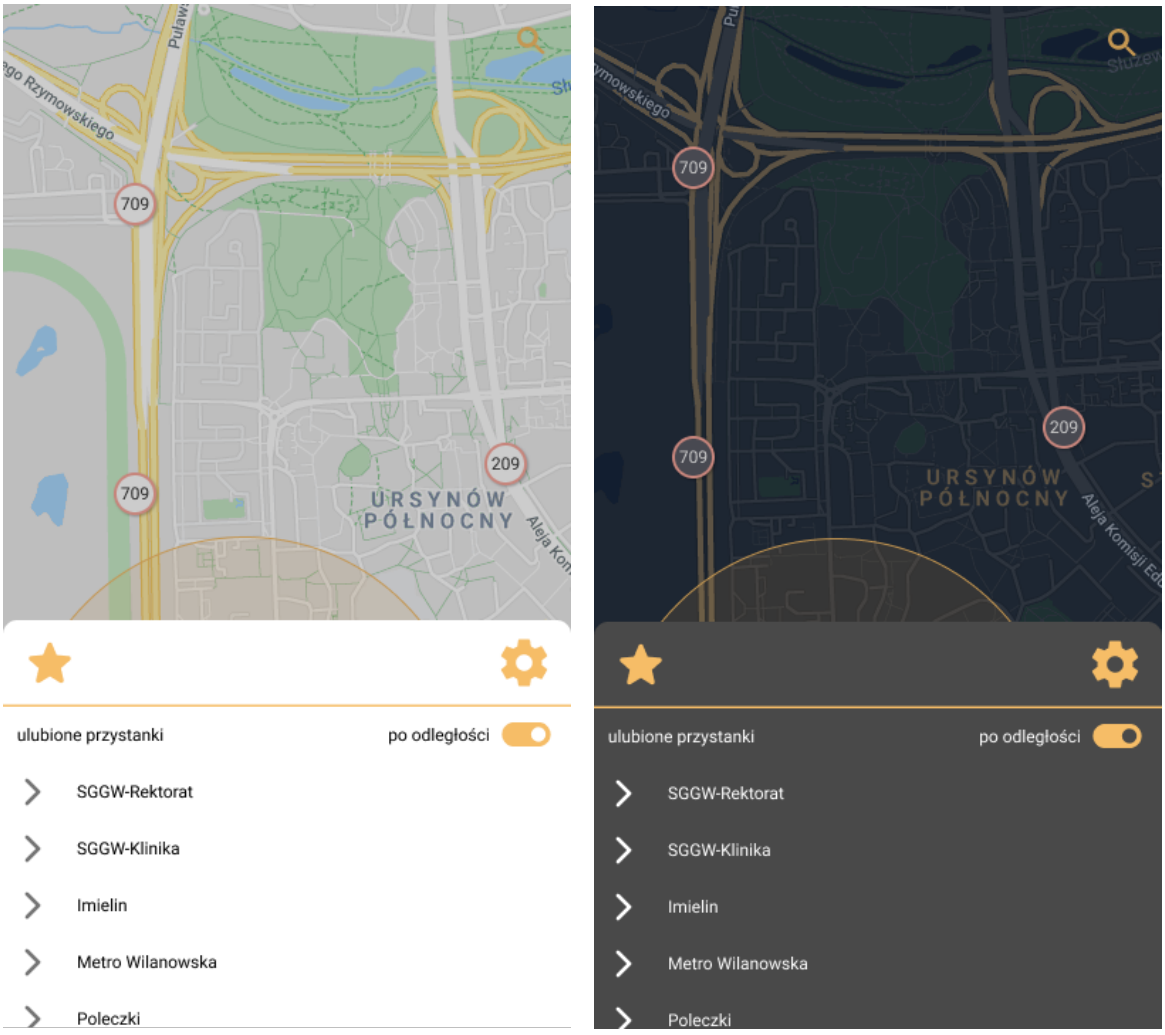
- pokazywać aktualne pozycje autobusów i tramwajów na mapie,
- pokazywać pozycje przystanków na mapie,
- udostępniać rozkłady jazdy na każdym z przystanków,
- umożliwiać na dodanie linii autobusówej lub tranwajowej do ulubionych,
- umożliwiać na dodanie przystanku do ulubionych,
- wspierać dwa motywy:
  - jasny,
  - ciemny.

1.2 Grafiki koncepcyjne

By przybliżyć wizje projektu wykonałem grafiki koncepcyjne w programie Figma. Są to rys. 1.1 i rys. 1.2.



Rysunek 1.1. Grafiki koncepcyjne ekranu głównego z wyświetlonym radarem, zaznaczonym przystankiem i przykładowymi autobusami. Od lewej: motyw jasny, motyw ciemny.



Rysunek 1.2. Grafiki koncepcyjne dolnego przybornika z ulubionymi przystankami. Od lewej: motyw jasny, motyw ciemny.

2 Implementacja

Do implementacji wybrałem *framework* React-Native stworzony przez firme Facebook[5]. 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` [4], 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 interfejsu API Warszawy są dostępne:

- pozycje pojazdów danej linii;
- zbiór linii, które odjeżdzają z danego przystanku;
- rozkład jazdy dla danej linii z danego przystanku;
- zbiór wszystkich przystanków.

Pozycje pojazdów są aktualizowane co 10 sekund i też z taką częstotliwością będą aktualizowane w aplikacji. Wszystkie z wymienionych punktów końcowych interfejsu 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 ponowne wyrenderowanie 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 ponowne wyrenderowanie całej aplikacji. To wiązałoby się z utratą na szybkości działania.

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 mapy. Jest to ważne bo gdy tylko zmieni się stan komponentu `GMap`, a ten będzie się zmieniał co 10 sekund, to wywoła to ponowne wyrenderowanie 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;
3. aktualizuje stan komponentu `GMap` pobranymi pojazdami.

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 (stan na

sierpień 2019 [1]). 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 i możliwej przyszłej rozbudowy aplikacji plik ten hostuje w serwisie Firebase. Na tym serwisie też stworzyłem punkt końcowy interfejsu API, który zwraca ten plik. Aplikacja przy uruchomieniu pobiera go.

Dzięki zmiennej `mapRegion` dostępnej w komponencie `GMap` moge ograniczyć pinezki do wyświetlenia. Zmienna `mapRegion` przechowuje aktualnie wyświetlany region mapy i posiada cztery następujące wartości:

- `latitude` – szerokość geograficzna w centrum wyświetlanego regionu mapy,
- `longitude` – długość geograficzna w centrum wyświetlanego regionu mapy,
- `latitudeDelta` – różnica szerokości geografinczych od lewej krawędzi mapy do prawej,
- `longitudeDelta` – różnica długości geografinczych od górnej krawędzi mapy do dolnej.

Używając tych czterech wartości można przefiltrować pinezki i wyświetlić tylko te, które są w obrębie `mapRegion`. Ponieważ, że zmienna `mapRegion` aktualizowana jest tylko kiedy manipulacja mapą (przesuwanie, obracanie itp.) zostanie zakończone to użytkownik nie zobaczy pinezek, które w trakcie ruchu weszły w obręb wyświetlanego regionu mapy. By to rozwiązać do filtracji pinezek używam czterokrotnie większego pola niż pole wyznaczone przez `mapRegion`

By jeszcze bardziej ograniczyć ilość pinezek stworzyłem trzy progi wyświetlania zależne od wartości `latitudeDelta`:

1. $[0; 0.02)$ – wyświetla pinezki pojedyńczych przystanków,
2. $[0.02; 0.035)$ – wyświetla pinezki grup przystanków,
3. $[0.035; \infty)$ – nie wyświetla żadnych pinezek przystanków.

Pinezki przystanków i grup przystanków różnią się tylko wielkością oraz wyglądają na rys. 2.1. Kliknięcie w pinezke grupy przystanków dopasuje wyświetlany region mapy do wszystkich przystanków w grupie, a w pinezke pojedyńczego przystanku zaznaczy go.

Rysunek 2.1. Pinezki przystanków lub grup przystanków. Od lewej: motyw jasny, motyw ciemny.

2.5 Radar

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

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 unikalnych linii radaru.

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

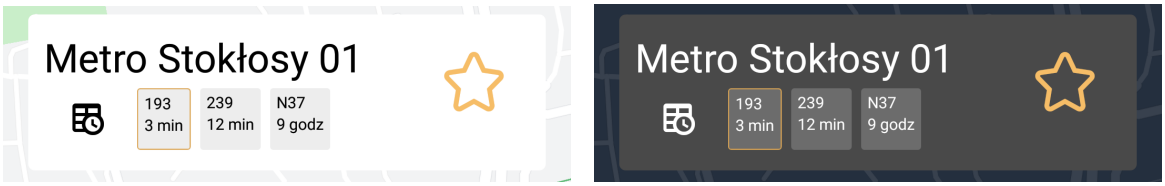
W aplikacji używam dwóch API, Warszawy oraz map Google. Każde z nich wymaga klucza API. Te klucze powinny pozostać prywatne i niewidoczne w kodzie aplikacji. Dlatego by ukryć klucze stworzyłem plik `.env` w lokalizacji domowej projektu, w którym zdefiniowałem dwie zmienne środowiskowe: `WARSAW_API_KEY`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

Następnie w kodzie posługiwałem się zmienną `BuildConfig` z modułu `react-native-config` [2] by otrzymać klucz API Warszawy, a klucz map Google, który wymaga umieszczenia w pliku `AndroidManifest.xml`, przy pomocy modułu `react-native-dotenv` [3].

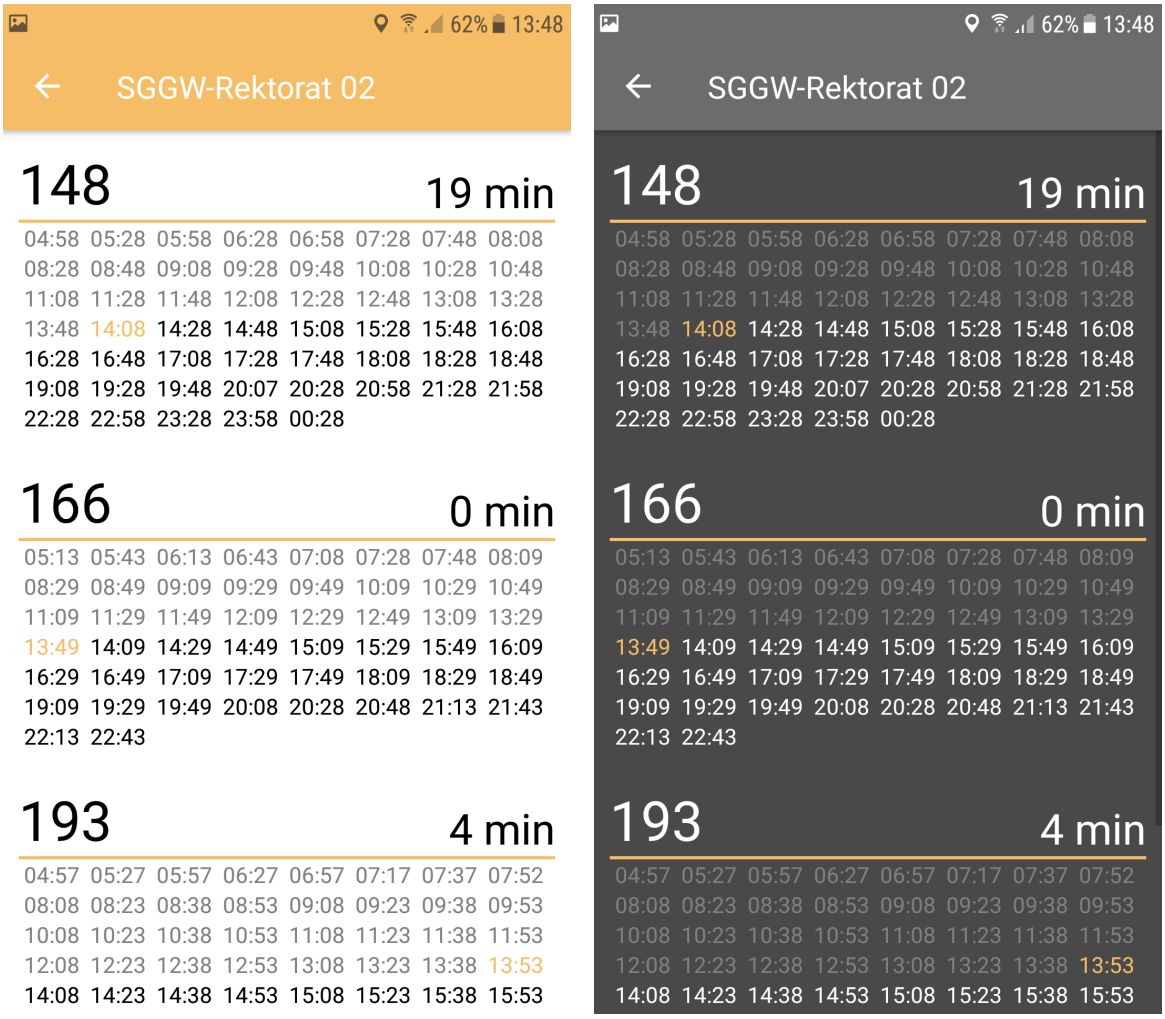
Po wykonaniu tych operacji można bezpiecznie użyć systemu kontroli wersji takiego jak "git" i serwisów jak Github do udostępniania kodu. Należy też dodać lokalizacje pliku `.env` do pliku `.gitignore`. Teraz by było możliwe zbudowanie aplikacji w trybie debug należy wypełnić plik `.env.example` własnymi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

2.7 Rozkłady jazdy

By stworzyć możliwość sprawdzenia rozkładu jazdy dla danego przystanka postanowiłem dodać przycisk w postaci ikony w kontenerze u góry ekranu po zaznaczeniu przystanka (patrz rys. 2.2). Po naciśnięciu aplikacja zmienia widok na liste odjazdów dla każdej z linii (patrz rys. 2.3). Na rysunkach można zauważyć, że dodałem informacje na temat ile czasu zostało do następnego odjazdu autobusu lub tramwaju. Jest to czas z zaokrągleniem do minut w dół obliczony na podstawie rozkładu jazdy.



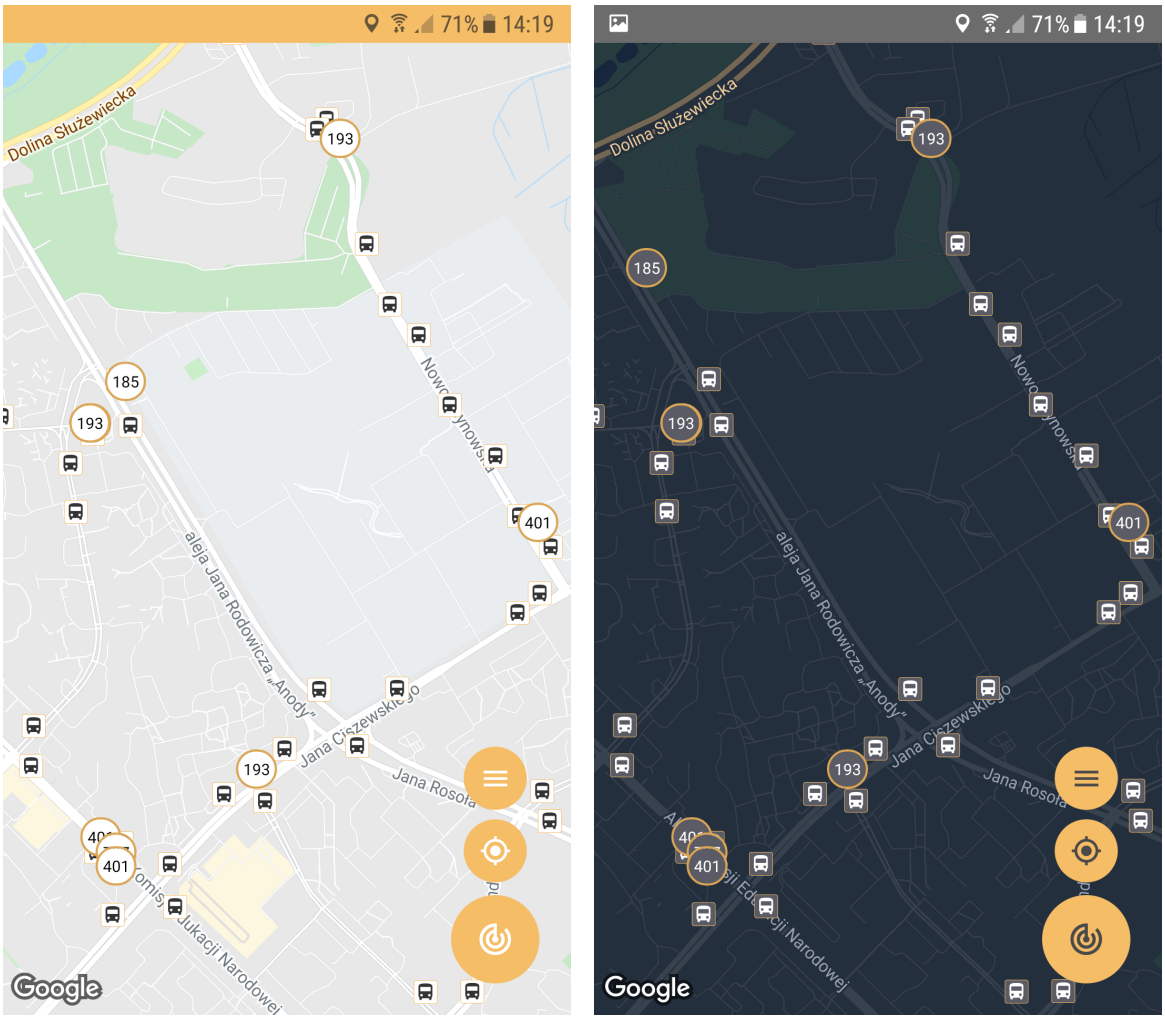
Rysunek 2.2. Górny kontener prezentujący informacje o zaznaczonym przystanku. Od lewej: motyw jasny, motyw ciemny.



Rysunek 2.3. Zrzut ekranu rozkładu jazdy. Od lewej: motyw jasny, motyw ciemny.

3 Podsumowanie i wnioski

Stworzono aplikacje mobilną spełniającą założenia przedstawione w 1.1. Końcowo ekran główny aplikacji wygląda jak na rys. 3.1. Funkcje aplikacji pozwalają na szybkie korzystanie z aplikacji przez użytkownika po dodaniu linii i przystanków do ulubionych. Aplikacja posiada też spore możliwości rozwoju takie jak przewidywanie czasów przyjazdu, kupowanie biletów lub wyświetlanie tras przejazdu danych linii autobusowych lub tramwajowych. Dzięki zastosowanej technologii React-Native aplikacja ma możliwość rozbudowy na systemy iOS.



Rysunek 3.1. Zrzut ekranu głównego aplikacji. Od lewej: motyw jasny, motyw ciemny.

4 Bibliografia

[1] Informator statystyczny. Technical report, Zarząd Transportu Miejskiego w Warszawie, https://www.ztm.waw.pl/statystyki/ | sierpień 2019, Sierpień 2019. (dostęp: 21.01.2020).

[2] Wiele autorów. *Moduł react-native-config*. https://github.com/luggit/react-native-config. (dostęp: 21.01.2020).

[3] Wiele autorów. *Moduł react-native-dotenv*. https://github.com/zetachang/react-native-dotenv. (dostęp: 21.01.2020).

[4] Wiele autorów. *Moduł react-native-maps*. https://github.com/react-native-community/react-nati (dostęp: 21.01.2020).

[5] Facebook, https://facebook.github.io/react-native/docs. *Dokumentacja React-Native*. (dostęp: 21.01.2020).

[6] m.st. Warszawa, https://api.um.warszawa.pl/. *Otwarte dane*. (dostęp: 21.01.2020).

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

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