

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<sup>1</sup>

---

<sup>1</sup>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

<b>1</b>	<b>Wstęp</b>	<b>9</b>
1.1	Założenia . . . . .	9
1.2	Grafiki koncepcyjne . . . . .	10
<b>2</b>	<b>Implementacja</b>	<b>12</b>
2.1	Publiczne API Warszawy . . . . .	12
2.2	Komponent GlobalContextProvider . . . . .	12
2.3	Aktualizacja pozycji pojazdów . . . . .	13
2.4	Pinezki przystanków . . . . .	13
2.5	Radar . . . . .	15
2.6	Ukrycie kluczy API w kodzie . . . . .	15
2.7	Rozkłady jazdy . . . . .	16
<b>3</b>	<b>Podsumowanie i wnioski</b>	<b>18</b>
<b>4</b>	<b>Bibliografia</b>	<b>19</b>





# 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 nieczytelnej dla przeciętnego człowieka. Rozwiązaniem może być aplikacja mobilna, dzięki której użytkownik będzie widział na mapie kiedy dokładnie przyjedzie autobus lub tramwaj.

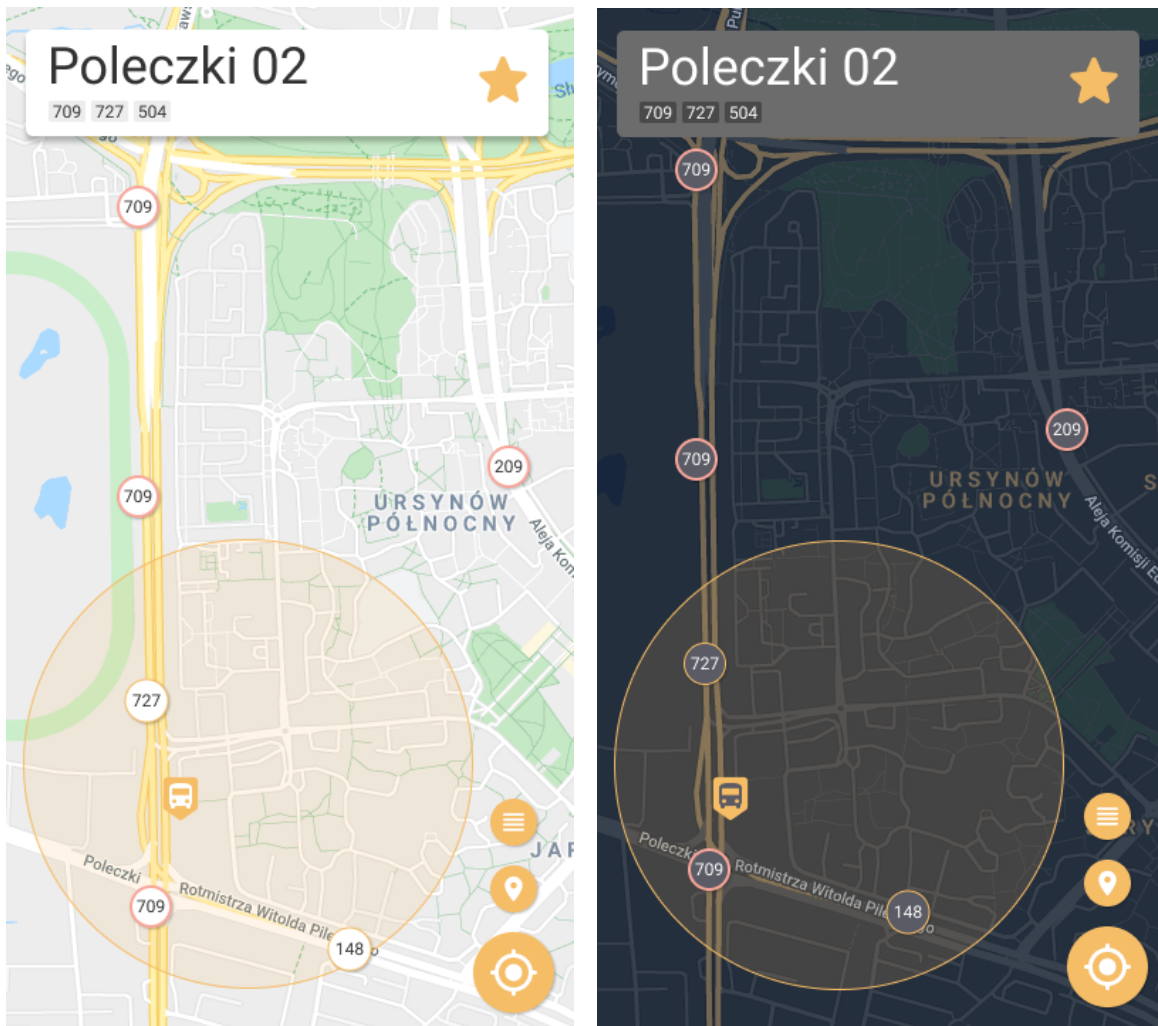
## 1.1 Założenia

Aplikacja powinna:

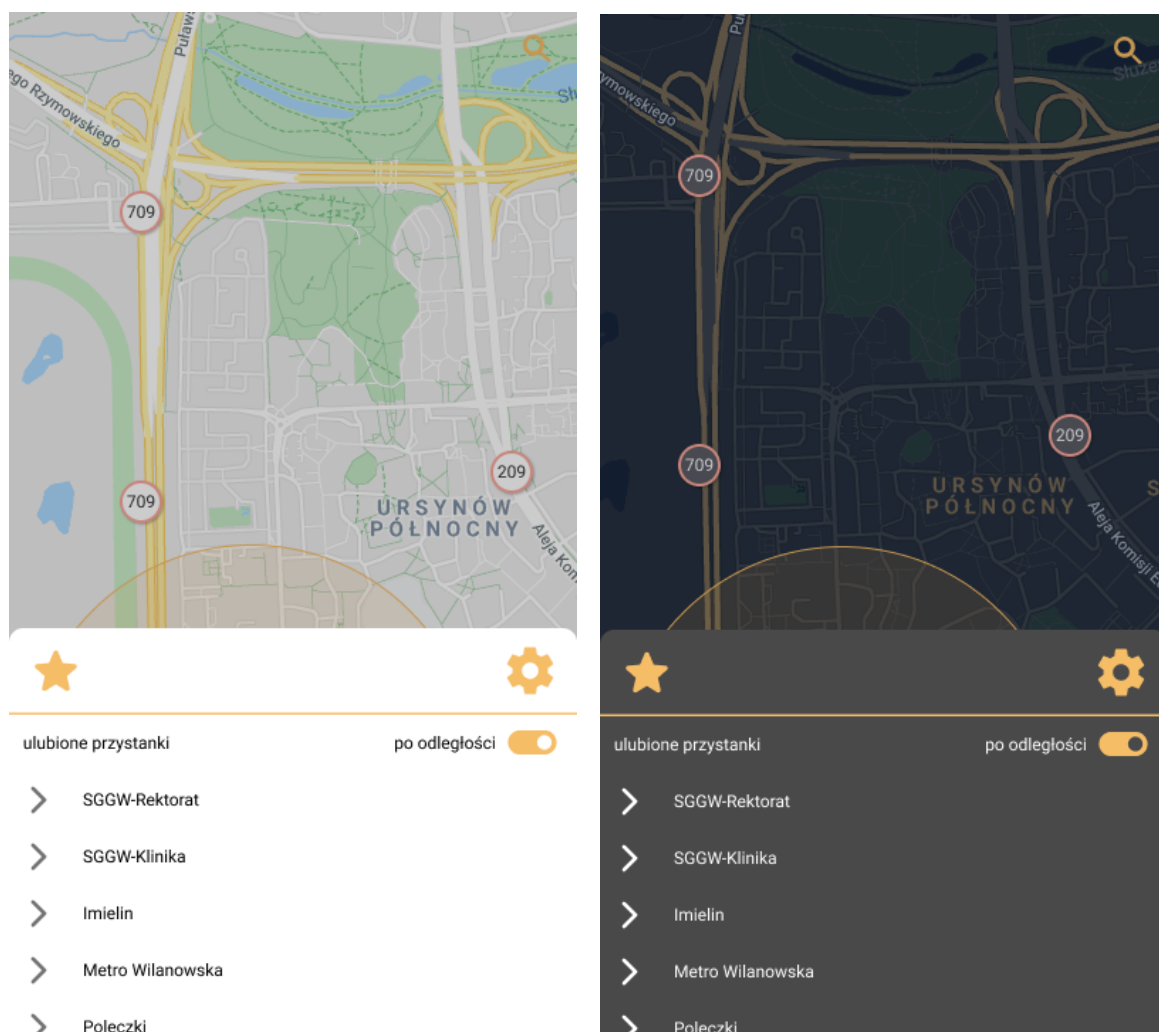
- Pokazywać aktualne pozycje autobusów i tramwajów na mapie
- Pokazywać pozycje przystanków na mapie
- Udostępniać rozkłady jazdy na każdym z przystanków
- Umożliwiać na dodanie linii autobusowej lub tramwajowej do ulubionych
- Umożliwiać na dodanie przystanku do ulubionych
- Wspierać dwa motywy:
  -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

## 1.2 Grafiki koncepcyjne

By przybliżyć wizję projektu wykonałem grafiki koncepcyjne w programie Figma.



**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 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). Skupię 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 odjeżdżają 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 logikę 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 zaznaczeniu

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 uwagę 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. 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 pozycję 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 API, który zwraca ten plik. Aplikacja przy uruchomieniu pobiera ten plik.

Dzięki zmiennej `mapRegion` dostępnej w komponencie `GMap` mogę 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 geograficznych od lewej krawędzi mapy do prawej
- `longitudeDelta` - różnica długości geograficznych 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 zakończona 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 pojedynczych przystanków
2. `[0.02;0.035)` – wyświetla pinezki grup przystanków
3. `[0.035;∞)` – nie wyświetla żadnych pinezek przystanków

Pinezki przystanków i grup przystanków różnią się tylko wielkością. Kliknięcie w pinezke grupy przystanków dopasuje wyświetlany region mapy do wszystkich przystanków w grupie.



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

## 2.5 Radar

Głównym celem radaru jest pokazywanie pinezek pojazdów linii z poza ulubionych. Ogranicza on też 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 funkcję radaru.

Działanie radaru jest następujące:

1. Użytkownik za pomocą przycisku w prawym dolnym rogu ustawia pozycję 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 odjeżdżają 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 dodać lokalizację pliku `.env` do pliku `.gitignore`. Teraz by było możliwe zbudowanie aplikacji w trybie debug należy wypełnić plik `.env.example` własnymi kluczami i zmienić jego nazwę na `.env`.

## 2.7 Rozkłady jazdy

By stworzyć możliwość zobaczenia rozkładu jazdy dla danego przystanku postanowiłem dodać przycisk w postaci ikony w kontenerze u góry ekranu po zaznaczeniu przystanku jak na rysunku 2.2. Po naciśnięciu aplikacja zmienia widok na liste odjazdów dla każdej z linii tak jak na grafice 2.3. Na grafikach można zauważyć, że dodałem informacje na temat ile czasu zostało do następnego odjazdu autobusu lub tramwaju. Jest to czas z zaokrągleniem minut w dół.



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



← SGGW-Rektorat 02

**148** 19 min

04:58 05:28 05:58 06:28 06:58 07:28 07:48 08:08  
 08:28 08:48 09:08 09:28 09:48 10:08 10:28 10:48  
 11:08 11:28 11:48 12:08 12:28 12:48 13:08 13:28  
 13:48 14:08 14:28 14:48 15:08 15:28 15:48 16:08  
 16:28 16:48 17:08 17:28 17:48 18:08 18:28 18:48  
 19:08 19:28 19:48 20:07 20:28 20:58 21:28 21:58  
 22:28 22:58 23:28 23:58 00:28

**166** 0 min

05:13 05:43 06:13 06:43 07:08 07:28 07:48 08:09  
 08:29 08:49 09:09 09:29 09:49 10:09 10:29 10:49  
 11:09 11:29 11:49 12:09 12:29 12:49 13:09 13:29  
 13:49 14:09 14:29 14:49 15:09 15:29 15:49 16:09  
 16:29 16:49 17:09 17:29 17:49 18:09 18:29 18:49  
 19:09 19:29 19:49 20:08 20:28 20:48 21:13 21:43  
 22:13 22:43

**193** 4 min

04:57 05:27 05:57 06:27 06:57 07:17 07:37 07:52  
 08:08 08:23 08:38 08:53 09:08 09:23 09:38 09:53  
 10:08 10:23 10:38 10:53 11:08 11:23 11:38 11:53  
 12:08 12:23 12:38 12:53 13:08 13:23 13:38 13:53  
 14:08 14:23 14:38 14:53 15:08 15:23 15:38 15:53

← SGGW-Rektorat 02

**148** 19 min

04:58 05:28 05:58 06:28 06:58 07:28 07:48 08:08  
 08:28 08:48 09:08 09:28 09:48 10:08 10:28 10:48  
 11:08 11:28 11:48 12:08 12:28 12:48 13:08 13:28  
 13:48 14:08 14:28 14:48 15:08 15:28 15:48 16:08  
 16:28 16:48 17:08 17:28 17:48 18:08 18:28 18:48  
 19:08 19:28 19:48 20:07 20:28 20:58 21:28 21:58  
 22:28 22:58 23:28 23:58 00:28

**166** 0 min

05:13 05:43 06:13 06:43 07:08 07:28 07:48 08:09  
 08:29 08:49 09:09 09:29 09:49 10:09 10:29 10:49  
 11:09 11:29 11:49 12:09 12:29 12:49 13:09 13:29  
 13:49 14:09 14:29 14:49 15:09 15:29 15:49 16:09  
 16:29 16:49 17:09 17:29 17:49 18:09 18:29 18:49  
 19:09 19:29 19:49 20:08 20:28 20:48 21:13 21:43  
 22:13 22:43

**193** 4 min

04:57 05:27 05:57 06:27 06:57 07:17 07:37 07:52  
 08:08 08:23 08:38 08:53 09:08 09:23 09:38 09:53  
 10:08 10:23 10:38 10:53 11:08 11:23 11:38 11:53  
 12:08 12:23 12:38 12:53 13:08 13:23 13:38 13:53  
 14:08 14:23 14:38 14:53 15:08 15:23 15:38 15:53

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

### **3 Podsumowanie i wnioski**

Stworzono aplikację mobilną spełniającą założenia przedstawione w 1.1. 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.

## **4 Bibliografia**



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

.....  
(czytelny podpis autora pracy)

