Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Systemy Informacyjno-Decyzyjne

Implementacja usługi sieciowej powiadamiającej o wyjątkowych opóźnieniach komunikacji miejskiej

Lidia Bondarzewska
Numer albumu 269270

promotor
dr inż. Mariusz Kamola

Warszawa 2018

Streszczenie

Tytuł pracy: Implementacja usługi sieciowej powiadamiającej o wyjątkowych opóźnieniach komunikacji miejskiej.

Celem pracy było stworzenie aplikacji sieciowej wspomagającej przemieszczanie się tramwajami po Warszawie poprzez powiadamianie użytkowników o szczególnych opóźnieniach w ich kursowaniu. Opracowany system, w celu pozyskania informacji o położeniu tramwajów w czasie rzeczywistym, wykorzystuje otwarte dane publiczne udostępniane przez Miasto Stołeczne Warszawa. Umożliwia to analizę ich ruchu pod kątem wyszukiwania składów, które poruszają się wolniej niż jest to przewidywane. Pozwala to na sprawdzanie w jak dużym stopniu badany obszar jest zablokowany, do czego jest wykorzystywany zaimplementowany specjalnie na potrzeby tego projektu algorytm. Jest on kluczowym elementem stworzonego systemu. Głównymi założeniami powstałej aplikacji są łatwość obsługi przez różne grupy wiekowe oraz informowanie o szczególnych opóźnieniach w wybranych obszarach Warszawy. Wobec tego kolejnym elementem projektu jest połączenie systemu z jednym z obecnie popularnych serwisów społecznościowych. Zostaje on wykorzystany jako kanał dystrybucji informacji. Publikowane są tam treści informujące o powstałych zatorach w ruchu tramwajowym.

Słowa kluczowe: *serwisy społecznościowe, aplikacja sieciowa, tramwaje, opóźnienia*

Abstract

Title: The implementation of a web service sending notification messages of significant delays in urban public transport.

The main purpose of this thesis was to create a web service facilitating getting around Warsaw by trams. The web application sends notifications of considerable delays in tramway traffic. The developed system uses open data made available by the Capital City of Warsaw in order to obtain real-time information about the location of trams. This allows for a detailed analysis of their movement to be made in terms of finding trams which are moving slower than it was scheduled. Furthermore, with the use of an algorithm which was specifically implemented for purpose of this project, it allows a check to be made on whether and to what extent a given area is congested. The algorithm is a key element of the system. The application itself is easy to use, regardless of the age of a user, and it provides information about significant delays in selected areas of Warsaw. A further important aspect of the project was connecting the system with one of the popular social networking sites which can be used to disseminate information about the occurrences of congestion in tramway traffic.

Keywords: *web service, social networking sites, trams, delays*

załącznik do zarządzenia nr 28/2016 r.
Rektora PW

Politechnika Warszawska

„załącznik nr 3 do zarządzenia nr 24/2016 Rektora PW

.....
miejscowość i data

.....
imię i nazwisko studenta

.....
numer albumu

.....
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta"

Spis treści

1 Wstęp

1.1 Motywacja i cel pracy

Od kilkunastu lat jesteśmy świadkami gwałtownego rozwoju naszego kraju, a w szczególności miast. Wiąże się on z coraz to większymi wymaganiami jakie stawia przed nami codzienność, ale jednocześnie my sami zaczynamy żądać niemal błyskawicznego zaspokajania naszych potrzeb i pragnień. Warto również zauważyć nieustannie pojawiające się coraz bardziej interesujące i ciekawsze możliwości czerpania przyjemności z życia, co dodatkowo zachęca nas do podążania w obranym kierunku. W przedstawionej sytuacji nieunikniony jest wzrost tempa życia. Znane jest wszystkim stare polskie przysłowie "czas to pieniądz" - kto z nas może sobie pozwolić na marnotrawstwo w dzisiejszych czasach? Jesteśmy więc zmuszeni do dostosowania się do otaczających nas reguł.

Oszczędność czasu znajduje się w czołówce listy naszych priorytetów. Ma bowiem ścisły związek z równowagą życia rodzinnego (rozwojem społecznego) oraz komfortem psychicznym co przekłada się m.in. na owocne budowanie kariery zawodowej, samorozwój. Wobec tego widzimy, że nawet najdrobniejsze czynności umożliwiające nam zaoszczędzenie chociażby kilku minut w ciągu dnia są niezmiernie istotne. Ma to skutki nie tylko krótkofalowe, lecz także widoczne w szerszej perspektywie.

Zastanówmy się nad zagadnieniem zarządzania czasem podczas podróżowania, a dokładniej przemieszczania się po mieście czy też między miastami. Nie jest zaskakujące, że czasami wymaga korzystania z transportu publicznego lub zwyczajnie jest to bardziej komfortowe. Również często zdarza się, że wyznaczona trasa jest złożona z odcinków, za które odpowiadają różni przewoźnicy. Do tego momentu wszystko jest w jak najlepszym porządku. Lecz zastanówmy się - co jeśli na naszej trasie wystąpią zdarzenia losowe lub takie, o których nie posiadaliśmy odpowiednio wcześniej informacji? Druga sytuacja nie wynika z zaniedbania z naszej strony - w natłoku informacji łatwo przeoczyć coś, co może okazać się później dla nas istotne. Równie naturalny jest wariant, kiedy po raz pierwszy przyjechaliśmy do pewnego miasta i nie jesteśmy jeszcze w stanie sprawnie się po nim poruszać. Przyczyny powstałego problemu nie są kluczowe w tej sytuacji. Nas jako pasażerów interesuje jej możliwie jak najszybsze i najbardziej komfortowe rozwiązanie. Po kilku latach spędzonych na studiach i pracy w Warszawie mogę powiedzieć, że teraz potrafię poradzić sobie w opisanej sytuacji w tym mieście. Jednakże początki były trudne. Podstawową aplikacją wspomagającą planowanie podróży było, jak i jest do tej pory, `https://jakdojade.pl`. Pozwala ona użytkownikowi w prozaiczny sposób wyznaczyć przejazd uwzględniając podane preferencje [1]. Jednakże ma ona jedną wadę, która w pewnych sytuacjach sprawia, że korzystanie z niej nie daje pożądanego rezultatu. Mianowicie - różnego rodzaju awarie na drodze oraz wywołane przez nie opóźnienia w ruchu komunikacji miejskiej. Jak do tej pory nie powstała też aplikacja, która w spójny i przystępny sposób, a co najważniejsze w czasie rzeczywistym, pozwalałaby na sprawdzenie czy w którymś obszarze miasta takie zdarzenie miało miejsce.

Do wybrania tego właśnie tematu skłoniły mnie w głównej mierze własne doświadczenia zdobywane przez kilka ostatnich lat. Chciałabym swoją pracą sprawić, żeby ludziom w Warszawie żyło się chociaż odrobinę łatwiej i przyjemniej. Uważam, że usprawnienie codziennych dojazdów do szkoły, uczelni albo

pracy znacząco wpłynie na podwyższenie stopnia zadowolenia z komunikacji miejskiej.

Zamierzeniem niniejszej pracy inżynierskiej jest stworzenie rozbudowanego i w pełni działającego prototypu aplikacji sieciowej wspomagającej przemieszczanie się tramwajami po Warszawie poprzez powiadamianie użytkowników o szczególnych opóźnieniach w ich kursowaniu. Opracowany system, w celu pozyskania informacji o położeniu tramwajów w czasie rzeczywistym, wykorzystuje otwarte dane publiczne udostępniane przez Miasto Stołeczne Warszawa [2]. Umożliwia to analizę ich ruchu pod kątem wyszukiwania składów, które poruszają się wolniej niż jest to oczekiwane. Pozwala to na sprawdzanie w jak dużym stopniu badany obszar jest zablokowany. Głównymi założeniami, którymi kierowałam się przy projektowaniu aplikacji, są łatwość obsługi przez różne grupy wiekowe oraz informowanie użytkowników o nietypowych i długotrwałych opóźnieniach w wybranych obszarach Warszawy. Wychodząc naprzeciw oczekiwaniom warszawskich pasażerów, powstała aplikacja jest połączona z jednym z popularniejszych obecnie serwisów społecznościowych. Zostaje on wykorzystany jako kanał dystrybucji informacji - publikowane są na nim treści informujące o powstałych zatorach w ruchu tramwajowym.

1.2 Dane po warszawsku

Skoro tworzymy aplikację wspomagającą przemieszczanie się tramwajami oczywiste jest, że musimy także posiadać dane o ich lokalizacji. Taką możliwość zapewnia nam projekt “Dane po warszawsku”, który powstał kilka lat temu. Jednakże nie jest on jeszcze spopularyzowany w takim stopniu, jak moglibyśmy się tego spodziewać. Może to wynikać z nowatorskiego podejścia jego twórców. Łączy on bowiem nie tylko administrację publiczną i biznes, ale także uczelnie (swój wkład miała również Politechnika Warszawska), organizacje społeczne i pozarządowe. Zaangażowanie tak wielu podmiotów nie jest jeszcze w Polsce zjawiskiem powszechnym, warto więc przyjrzeć się temu projektowi nieco bliżej.

Odpowiednim wstępem wyjaśniającym cel projektu będzie przytoczenie fragmentu wypowiedzi jednego z twórców - “ "Dane po warszawsku" wynikają z wiary w to, że dzięki otwartości i łatwości dostępu do danych opisujących rzeczywistość w mieście, Warszawa będzie bliższa, bardziej zrozumiała i przyjazna swoim mieszkańcom.” [3]. To również forma wsparcia i inspiracji dla osób chcących zrobić coś dla swojej lokalnej społeczności.

Do tej pory zostały wyznaczone dwa etapy - zbudowanie platformy udostępniającej publiczne dane miejskie oraz konkurs na aplikacje wykorzystujące udostępniane przez nią dane, do których odniosę się w następnym podrozdziale. Przeprowadzenie pierwszego etapu wymagało przeanalizowania kilku kluczowych zagadnień, a mianowicie - jakie dane posiada Warszawa, które z nich mogą zostać publicznie udostępnione oraz w jaki sposób.

Zanim zaczniemy używać platformy, warto zwrócić uwagę na warunki korzystania z danych. Okazuje się, że możemy z nich swobodnie korzystać, bowiem są one materiałami urzędowymi i większość surowych danych nie ma charakteru twórczego, zatem nie są chronione przez prawa autorskie. Jedyne na co powinniśmy zwrócić uwagę, to warunki ponownego wykorzystania informacji publicznej. Obowiązuje nas podanie daty wytworzenia oraz źródła danych, tzn. Miasto Stołeczne Warszawa.

1.2 Dane po warszawsku

Informacje o rodzajach dostępnych zbiorów danych wraz z opisami znajdują się na stronie `https://api.um.warszawa.pl/`. Zostały podzielone na trzynaście kategorii obejmujących wiele aspektów naszego codziennego życia w mieście. Niektóre z nich wymagają klucza dostępu do API. W tym celu musimy założyć konto, podając standardowe dla tej czynności dane. W momencie, kiedy uzyskaliśmy unikatowy klucz, możemy przejść do wywoływania przykładowych zapytań podanych w dokumentacji. Dostępne zbiory zawierają m.in. informacje o lokalizacjach żłobków, punktach informacji turystycznej, biur Urzędu Miasta, ambasad, Urzędów Skarbowych, sądów. Możemy też sprawdzić granice dzielnic oraz dane o budynkach i lokalach wraz z szczegółową charakterystyką obiektów. Mamy też dostęp do planów zagospodarowania przestrzennego oraz fotoplanów z kilku poprzednich lat czy też bazy noclegowej obejmującej kilka rodzajów obiektów. Nas oczywiście w tym momencie najbardziej interesują dane o transporcie miejskim, a dokładnie tramwajach.

1.2.1 API tramwajów warszawskich

API pojazdów Tramwajów Warszawskich zawiera dane o lokalizacji geograficznej tramwajów. Zawarte są w nim informacje o wszystkich włączonych do ruchu w danej chwili pojazdach [4]. Niezmiernie istotny jest czas aktualizowania danych. W tym przypadku wynosi on 30 sekund. Wobec tego, mamy dostęp do położenia składów w czasie rzeczywistym. Umożliwi to nam przekazywanie informacji użytkownikom końcowym bez wprowadzającego w błąd opóźnienia.

Poprzez API mamy dostęp do informacji ukazuje tabela 1.1:

Nazwa	Typ danych	Opis
Time	datetime	Znacznik czasu (ostatnia aktualizacja)
Lat	Float	Szerokość geograficzna GPS
Lon	Float	Długość geograficzna GPS
Lines	String	Numery wszystkich realizowanych linii (dla brygad wieloliniowych pojawi się więcej niż jedna linia)
Brigade	String	Numer brygady (prowadzący, wóz i rozkład) (na razie zajmujemy się wariantem tylko z oznaczeniem liczbowym)
LowFloor	Bool	Określenie, czy zadanie realizuje tramwaj niskopodłogowy 1 - tak, 0 - nie

Tabela 1.1. Informacje o Tramwajach Warszawskich udostępniane przez API

W dokumentacji podany jest również identyfikator zasobu *resource_id*. Jest on niezbędny do dostępu do wybranego zbioru danych. W tym przypadku $resource\_id = c7238cfe - 8b1f - 4c38 - bb4a - de386db7e776$. Wykorzystywane zapytanie HTTPS to GET, ponieważ pobieramy zasób wskazany przez podany adres. Lista parametrów wywołania jest dość krótka, potrzebujemy jedynie:

1. *Id* - identyfikator zasobu
2. *apikey* - identyfikator dostępu do API (dostęp do tego zasobu wymaga rejestracji na portalu `api.um.warszawa.pl` i pobrania unikalnego *apikey*)

Należy również wyjaśnić kwestię współrzędnych geograficznych. Są one liczbami typu float zgodnymi ze standardem EPSG 4326 (WGS 84). Na przykładzie centrum Warszawy - 21.012 dla długości oraz 52.230 dla szerokości geograficznej.

Wobec tego, przykładowe zapytanie ma następującą postać: `https://api.um.warszawa.pl/api/action/wsstore_get/?id=c7238cfe-8b1f-4c38-bb4ade386db7e776&apikey=wartosc`

Natomiast zwracany przykładowy JSON prezentuje się następująco:


```
{
  "result": [
    {
      "Lon": 21.0150375,
      "Lines": "18",
      "Time": "2018-07-11T15:59:24",
      "Lat": 52.0262352,
      "LowFlooer": false,
      "Brigade": "14"
    },

    ...

    {
      "Lon": 21.0159721,
      "Lines": "4",
      "Time": "2018-07-11T15:59:29",
      "Lat": 52.1680756,
      "LowFlooer": true,
      "Brigade": "10"
    }
  ]
}
```


Obecnie używana przeze mnie metoda zawierała wcześniej pole *tabor* umożliwiające jednoznaczne zidentyfikowanie tramwaju. Niestety zostało ono usunięte, przez co rozróżnianie tramwajów jest nieco bardziej kłopotliwe, lecz w pełni możliwe. W tej chwili jest ona zastępowana wywołaniem, które zwraca aktualną lokalizację wszystkich pojazdów komunikacji miejskiej, tzn. zarówno tramwajów jak i autobusów. Jednakże w nowej wersji pole *tabor* nie zostało przywrócone. Co więcej, część pól została usunięta, a pozostałe pokrywają się z uzyskiwanymi używanym przeze mnie wywołaniem. Dlatego też nie zdecydowałam się na zmianę na tym etapie zaawansowania prac.

1.3 Przegląd istniejących rozwiązań

1.3 Przegląd istniejących rozwiązań

Następnym krokiem po przygotowaniu platformy był konkurs na aplikacje wykorzystujące udostępniane przez nią dane. Bez wątpienia nie mogło w nim zabraknąć aplikacji dla sektora transportu miejskiego. Jedną z nich, a jednocześnie zdobywcą pierwszej nagrody, jest *Warszawski ninja* [5].

Można wyróżnić trzy podstawowe kanały działania. Po pierwsze, informuje innych potencjalnych pasażerów. Prosty interfejs pozwala użytkownikom anonimowo publikować różnego rodzaju zgłoszenia - awarie w komunikacji miejskiej, korki, niespodziewane zdarzenia. Umożliwia też zadawanie pytań dotyczących konkretnych punktów w mieście. Następnie, istnieje możliwość dodania wybranych linii i przystanków do "obserwowanych". W przyszłości aplikacja będzie wysyłała powiadomienia tylko odnośnie elementów z tej listy. Jednak najważniejszym jej elementem jest weryfikacja zgłoszeń przez innych użytkowników. Jest to możliwe dzięki włączonej opcji dodawania komentarzy.

Kolejną praktyczną aplikacją korzystającą z "Danych po warszawsku" jest *Gdzie Ten Tramwaj? (Autobus też!)* [6]. Oferuje ona użytkownikowi widok mapy Warszawy z zaznaczonymi lokalizacjami autobusów oraz tramwajów. Jako że udostępniane dane są odpowiednio często aktualizowane, pozwala to na poprawną ocenę sytuacji. Z mapy można łatwo odczytać położenie pojazdów. Dodatkowym udogodnieniem jest zaznaczenie niedługiego odcinka przebytej drogi poprzez poprowadzenie za znacznikiem czerwonej linii, aby możliwe było określenie kierunku pojazdu. Tutaj również użytkownicy mają możliwość wybrania "ulubionych" linii, jednakże przekłada się to tylko na ograniczenie znaczników na mapie. Aplikacja nie wysyła żadnych powiadomień.

Przedstawione aplikacje należycie spełniają swoje zadanie - informują warszawskich pasażerów o utrudnieniach na wybranych przez nich trasach. Jednakże można doszukać się kilku aspektów, które uwzględnione bądź też zaprojektowane w inny sposób sprawią, że nowo powstająca aplikacja będzie wyróżniała się na tle pozostałych swoimi funkcjonalnościami. Po pierwsze, warto zwrócić uwagę na stopień zależności od użytkowników. Istotna jest również kwestia sposobu wysyłania powiadomień oraz ich treść, a także wybór obszaru obsługiwanego przez aplikację. Kolejnym bardzo istotnym zagadnieniem jest względna anonimowość odbiorców. Korzystanie z aplikacji nie będzie wymagało od użytkownika założenia nowego konta, ponieważ do komunikacji wykorzystany zostanie Twitter. Jest to serwis społecznościowy obdarzony wysokim poziomem zaufania społecznego, który dba o ochronę powierzonych mu danych osobowych. Dzięki temu zyska ona większą popularność - będzie postrzegana jako bezpieczna aplikacja a nie narzędzie do śledzenia lokalizacji lub zbierania innych danych o jej użytkownikach.

2 Realizacja pracy i rozwiązywanie problemów

2.1 System GPS

Niniejsza praca bazuje na systemie GPS [7], dlatego też zdecydowałam się poświęcić mu nieco więcej uwagi. Na wstępie warto zauważyć, że stosowanie określenia "GPS" na system nawigacji jest niepełne, bowiem cała gałąź technologii satelitarnej to GNSS (ang. Global Navigation Satellite System). Jednak największą popularność zyskał właśnie GPS, zapewne dzięki wyjątkowo wczesnej dacie powstania. Czym tak dokładnie jest i z czego się składa system nawigacji satelitarnej określający pozycję na mapie?

Składa się on z trzech segmentów - satelitarnego, naziemnego oraz użytkownika. Departament Obrony USA nadzoruje utrzymywanie i zarządzanie dwóch pierwszych spośród wymienionych segmentów.

W systemie GPS wyróżnia się dwie podstawowe usługi - PPS (z ang. Precise Positioning Service), która jest wykorzystywana przez NATO, armię amerykańską oraz uprawnione starannie wyselekcjonowane podmioty, a także SPS (z ang. - Standard Positioning Service), która to właśnie jest darmowo udostępniana pozostałym użytkownikom. Precyzja wyznaczenia lokalizacji dla usługi SPS w najgorszym przypadku może wynosić do 9 metrów na poziomie ufności 95%. Zależy ona bowiem m.in. od układu saletitów, warunków atmosferycznych i blokady sygnału. Jednakże średnio osiąga wartość około 4 metrów. Dlatego też przyjmujemy, że taka dokładność jest wystarczająca.

Jak jest to wykorzystane w tramwajach? Powstał System Nadzoru Ruchu Tramwajów 2000 [8], nadzorujący wszystkie tabory. Obejmuje on m. in. pojazdy, czyli tramwaje, centrum komunikacyjne, a także centrum dyspozytorskie. W każdym tramwaju w kabinie motorniczego zainstalowany jest systemowy komputer pokładowy, zwany urządzeniem przewoźnym. Niektórymi z jego elementów są również odbiornik GPS oraz zewnętrzne anteny umieszczone na dachu. W celu zwiększenia dokładności informacji o położeniu tramwaju, do urządzenia przewoźnego przesyłany jest sygnał o otwarciu drzwi na przystanku. Umożliwia to znalezienie dla takiego punktu najbliższego przystanku i przesłanie informacji do centrum dyspozytorskiego. Efektem jego działania jest komplet danych o każdym aktywnym tramwaju. Do części z nich mamy dostęp poprzez platformę "Dane po warszawsku".

2.2 Przygotowania do realizacji oraz wstępne założenia

Początek działań nad realizacją pracy inżynierskiej jednoznacznie wyznaczył moment sformułowania założeń, a tym samym sprecyzowanie wymagań stawianych powstającej aplikacji. Dopiero wtedy możliwe było przejście do kolejnego etapu przygotowań, a mianowicie doprecyzowania zakresu pracy. Pozwoliło to na podzielenie projektu na odpowiedniej wielkości etapy i zadania, a następnie realistyczne oszacowanie czasu trwania poszczególnych zadań. Wymienione dwa elementy to kluczowe czynniki wpływające na sukces projektu, należało więc poświęcić im odpowiednią ilość czasu. Dopiero mając listę zadań do wykonania, możliwa była analiza technologii oraz narzędzi odpowiednich do osiągnięcia oczekiwanych rezultatów, a następnie implementacja zaprojektowanego rozwiązania.

Wobec powyższego, celem pracy jest zaimplementowanie usługi sieciowej, która spełnia opisane

2.2 Przygotowania do realizacji oraz wstępne założenia

poniżej założenia.

1. **Niezależność od aktywności użytkowników**

   Jedna z przedstawionych przeze mnie pokrewnych aplikacji, tzn. *Warszawski ninja*, opiera się na pomyśle przekazania inicjatywy użytkownikom. Jednakże moim celem jest stworzenie wyróżniającego się projektu. Wobec tego powstały interfejs nie będzie wymagał interakcji z użytkownikiem, aby poprawnie działać. Jest to kluczowa cecha, bowiem chcemy dostarczyć gotowy produkt, którego konfiguracja oraz użytkowanie nie będzie wymagało dodatkowego nakładu czasu.

2. **Informowanie o wyjątkowych opóźnieniach tramwajów w wybranych obszarach miasta**

   Inne aplikacje pozwalają na "obserwowanie" wybranych linii, w jednej z nich również przystanków. Tutaj chcielibyśmy rozwinąć się w stronę wymienionych przystanków, a dokładniej stworzyć listę obszarów miasta, które mają strategiczne znaczenie w przemieszczaniu się po nim za pomocą tramwajów. Zapewniłoby to użytkownikom możliwość sprawdzenia sytuacji w takim punkcie bez konieczności wyboru linii.

   Jest to znaczne udogodnienie, ponieważ uwzględnia zmiany kursowania poszczególnych linii związane z remontami, okresem świątecznym bądź innymi tego typu sytuacjami. Byłoby to również wygodne rozwiązanie dla osób, które nie znają miasta lub przyjeżdżają sporadycznie, w związku z czym nie potrafią swobodnie się po nim poruszać bez wsparcia tego rodzaju aplikacji.

3. **Połączenie z serwisem społecznościowym**

   Kolejnym istotnym elementem rozwiązania jest integracja aplikacji z jednym z popularnych w Polsce serwisów społecznościowych. Dotychczas powstałe aplikacje nie zawsze są połączone z serwisami społecznościowymi, a jeśli są, to przeważnie w celu reklamy produktu. Dlatego również w tym aspekcie chcemy skierować projekt rozwiązania w przeciwną stronę.

4. **Wykorzystanie popularnych serwisów społecznościowych jako kanałów dystrybucji informacji**

   Obecnie większość posiadaczy smartfonów ma zainstalowanych wiele aplikacji. W związku z tym, nie chcemy obarczać użytkownika koniecznością pobrania kolejnej. Dlatego też proponowane przeze mnie rozwiązanie bazuje na wykorzystaniu Twittera jako kanału dystrybucji informacji dla użytkownika. Dzięki temu ograniczymy jego wysiłek do minimum - jego zadaniem będzie jedynie posiadanie konta na Twitterze.

Dodatkowym atutem aplikacji jest podgląd pozycji tramwajów w czasie rzeczywistym na mapie w oknie przeglądarki. Nie jest to jednak najważniejszy komponent rozwiązania, aczkolwiek jest interesującym obiektem obserwacji oraz nieocenionym wsparciem przy testowaniu funkcjonalności.

Projektując rozwiązanie opierać się będziemy na pewnym kluczowym kryterium. Tramwaj zostaje zaklasyfikowany jako opóźniony, jeżeli w ciągu 5 minut przejechał mniej niż 400 metrów. Zostało ono wyznaczone metodą inżynierską, jednakże weryfikacja jego poprawności będzie niezmiernie interesująca, dlatego zostanie jej poświęcona część rozdziału podsumowującego.

2.3 Wybór technologii

2.3.1 Języki programowania

Podjęcie decyzji o wyborze języków do poszczególnych części aplikacji - głównej części serwerowej oraz warstwy prezentacji odpowiadającej m.in. za obsługę widoku pojazdów na mapie - w moim przypadku nie było trudne. W czasie studiów pisałam projekty w kilku językach, jednak najbardziej przypadł mi do gustu Python. Również w pracy zawodowej miałam okazję pogłębić jego znajomość i właśnie dlatego wybrałam go jako główny język mojej aplikacji. Oto niektóre z jego zalet:

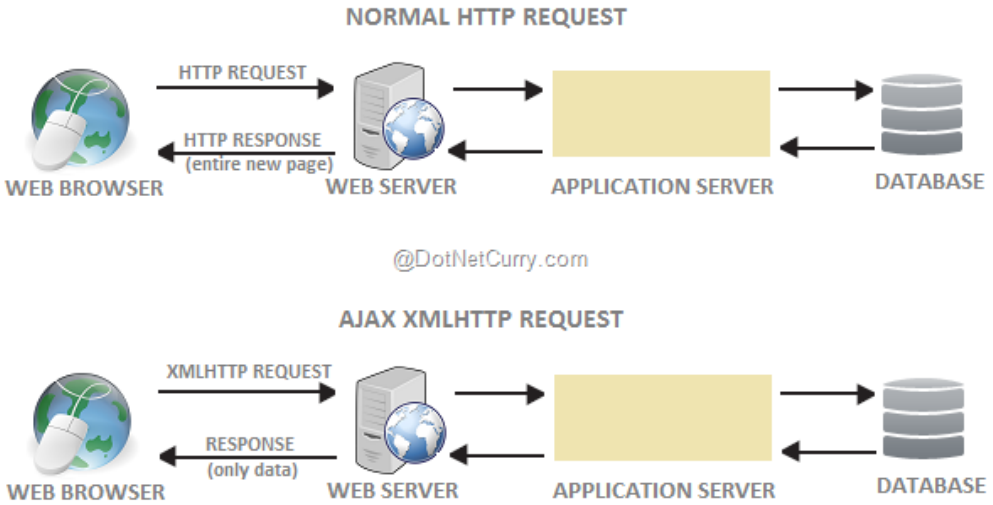
1. **Prosta składnia** - Można nawet użyć określenia intuicyjna. Dzięki temu kod w Pythonie jest czytelny, przejrzysty i estetyczny.
2. **Język interpretowany** - Nie wymaga kompilacji.
3. **Zbiór gotowych bibliotek** - Posiada bardzo dużo bibliotek z wielu dziedzin informatyki i nauki. Wystarczy przeczytanie odpowiedniego fragmentu dokumentacji, by było możliwe dalsze opracowywanie zagadnienia.
4. **Szybkie budowanie prototypów** - Pozwala na zrobienie więcej, wykorzystując mniej kodu. Dzięki temu umożliwia szybką realizację idei, a w konsekwencji również prototypów.
5. **Liczne frameworki** - Istotne zwłaszcza w przypadku aplikacji sieciowych. Pozwalają na skupienie się na pisaniu esencji aplikacji bez konieczności zatrzymywania się na implementowaniu już istniejących gotowych modułów. Wpływa to na znaczną poprawę komfortu pisania.
6. **Django** - Framework przeznaczony do tworzenia aplikacji internetowych. Prawdopodobnie jedna z największych zalet korzystania z Pythona.

   Oferuje m.in. dostęp do takich funkcjonalności jak: struktura bazy danych generowana ze zwykłych klas pythona, zaawansowany ORM (przyspieszenie projektowania struktury i bezpiecznych operacji na danych bez konieczności użycia SQL), automatycznie generowany panel administratora, własny prosty serwer do testowania aplikacji, obsługa kilku baz danych - MySQL, PostgreSQL, SQLite, Oracle [9].

Natomiast do części projektu obejmującej obsługę podglądu na mapie wybrałam JavaScript. Obecnie jest on standardem dla generowania treści dynamicznej i jako jedyny gwarantuje kompatybilność między przeglądarkami. Umożliwia on też odciążenie serwerów oraz ograniczenie ilości zbędnych danych przesyłanych pomiędzy nimi a przeglądarką klienta. Wykorzystuję również AJAX (z ang. Asynchronous JavaScript and XML, Asynchroniczny JavaScript i XML), który jest przeważnie kojarzony właśnie z Javascriptem, chociaż nie jest jedynym obsługiwanym językiem. Jest to technologia tworzenia aplikacji internetowych dla której charakterystyczny w interakcji użytkownika z serwerem jest fakt, że odbywa się ona bez konieczności przeładowania całego dokumentu. Niekoniecznie musimy używać XML, obsługiwane są również inne typy danych, takie jak czysty tekst lub JSON.

2.3 Wybór technologii

Rysunek 2.1 przedstawia schemat obrazujący różnice w działaniu strony z wykorzystaniem różnych rodzajów żądań do serwera.



Rysunek 2.1. Schemat przedstawiający porównanie typowego żądania HTTP oraz żądania z wykorzystaniem AJAX [10]

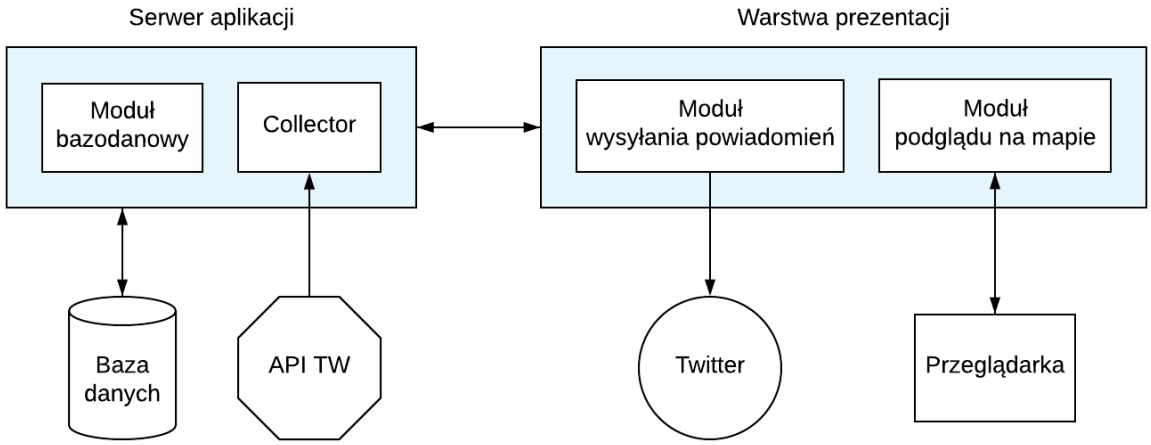
Interesujące są możliwości, które można uzyskać poprzez wykorzystanie AJAX. Wpływa ono na zmniejszenie liczby zapytań do serwera, ponieważ elementy takie jak arkusze stylów czy też skrypty są ładowane tylko jeden raz. Pozwala również obniżyć przeciążenie sieci w sytuacji, gdy chcemy uaktualnić tylko mały fragment, ponieważ umożliwia odświeżenie jedynie danego fragmentu strony. Wobec tego użytkownik ma wrażenie, że oczekiwane zmiany wykonują się niemal natychmiastowo, co podwyższa jego stopień zadowolenia z używanej usługi.

2.3.2 Technologie wspomagające zarządzanie projektem

W sytuacji, kiedy nad projektem pracuje jedna osoba, nie występują konflikty w plikach zmienionych podczas pracy nad różnymi funkcjonalnościami, ponieważ równolegle nikt poza nami nie edytuje kodu. Jednakże dla zwiększenia bezpieczeństwa zdecydowałam się na korzystanie z systemu kontroli wersji Git [11]. Jest to bardzo ważne narzędzie, które śledzi wszystkie zmiany dokonywane w plikach oraz umożliwia ich podgląd bez potrzeby dostępu do sieci. Oferuje również możliwość rozbicia zmodyfikowanego kodu na małe porcje (wiele commitów) Daje również możliwość przywrócenia dowolnej wcześniejszej wersji, jeśli tylko była zatwierdzona. Pozwala na elastyczne operowanie gałęziami - tworzenie lokalnych kopii kodu jest bardzo przydatne podczas testowania nowych funkcji lub równoległej pracy nad kilkoma funkcjonalnościami, jednocześnie nie oddziałując na całą aplikację.

2.4 Architektura rozwiązania

2.4.1 Ogólny zamysł



Rysunek 2.2. Schemat architektury systemu

Idea wyodrębnienia takich właśnie modułów (rysunek 2.2) nasuwa się intuicyjnie. W ramach warstwy prezentacji możemy wyróżnić dwie części - odpowiedzialną za podgląd lokalizacji tramwajów na mapie oraz obsługującą komunikację z Twitterem i wysyłanie powiadomień na odpowiednie kanały. W ramach serwera aplikacyjnego również zostały wydzielone dwa komponenty. *Collector* odpowiada za zbieranie danych, a następnie ich obróbkę do pożądanego formatu. W jego obrębie powstały również własne komendy do Django służące do zautomatyzowania procesu przetwarzania danych, wykorzystując zaimplementowany algorytm wyszukiwania opóźnień, a także procesu testowania aplikacji poprzez symulację API. Drugi mniejszy komponent jest odpowiedzialny za tworzenie modeli bazy danych.

2.4.2 Zbieranie i przechowywanie danych

Moduł Collectora

Jego zadaniem jest pozyskanie i przygotowanie danych, na których opiera się warstwa prezentacji. Przed przystąpieniem do pobierania danych, zajmiemy się przygotowaniem bazy danych, pomimo możliwości korzystania z Django bez niej. Warto jednak skorzystać z oferowanego rozwiązania, ponieważ mamy do dyspozycji wbudowany mapper obiektowo-relacyjny. Dzięki temu możemy opisać strukturę zaprojektowanej bazy danych wyłącznie za pomocą pythonowego kodu - tym m.in. zajmuje się drugi moduł serwera.

Jedno zapytanie do API zwraca nam JSONa z listą informacji o poszczególnych tramwajach. Dla uproszczenia będę używała określenia "paczka", które dobrze oddaje koncepcję rozwiązania. Każdy pojedynczy wpis z takiej paczki zostaje zapisany w tabeli *TramEvent*. Poza polami występującymi

2.4 Architektura rozwiązania

w odpowiedzi wywołania, trzymane są w niej jeszcze dane uzyskane w kolejnych krokach - stan, wektor przesunięcia oraz poprzednie położenie dające niezerowy wektor przemieszczenia. Jednym z najważniejszych dodatkowych pól jest *check_timestamp*, które jest kluczem obcym i określa czas zapisania pomiaru w bazie. Paczki są rozróżniane właśnie po czasie zapisu - do tego służy tabela *CheckTimestamp*. Dzięki temu możemy w prosty sposób manewrować wpisami na podstawie ich "świeżości".

Wykorzystamy bibliotekę Requests, która umożliwia wykonywanie żądań w nieskomplikowany sposób. Jako że będziemy pobierać określony zasób, posłużymy się metodą GET, konstruując URL-a w sposób opisany w jednym z poprzednich rozdziałów: `x=requests.get(_URL)`. Otrzymaliśmy obiekt klasy *Response*, z którego wyłuskamy potrzebne nam informacje. Pamiętamy, że zajmujemy się danymi w formacie JSON, więc aby przeczytać zawartość odpowiedzi serwera posługujemy się wbudowanym dekoderem: `j=x.json()`, po czym mamy dostęp do paczki tramwajów poprzez odwołanie się do pola "result".

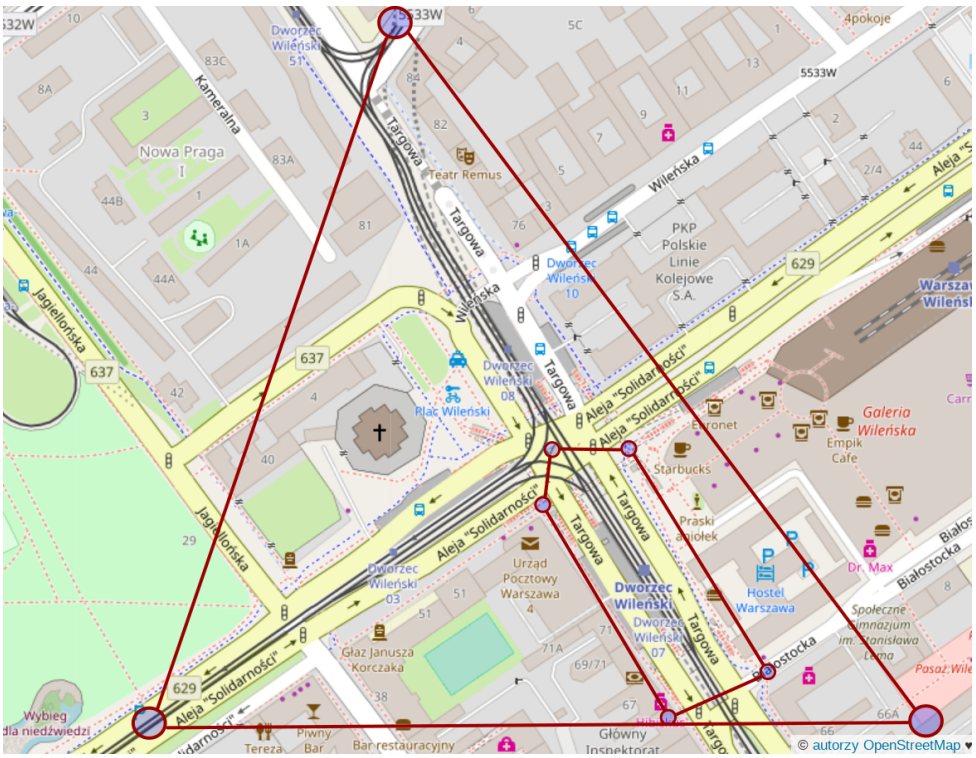
Następnym krokiem jest obróbka do pożądanego formatu. W poszczególnych rekordach często znajdują się białe znaki, które muszą zostać usunięte. Konieczne jest także zapisanie stopki czasowej w formacie umożliwiającym późniejsze wygodne korzystanie.

We wstępnej fazie projektu konieczne było również zdefiniowanie sposobu rozróżniania pojazdów. W obecnej wersji API zostało niestety usunięte pole *tabor*, które pozwalało jednoznacznie zidentyfikować tramwaj. Jednakże dokładna analiza pobieranych danych umożliwiła znalezienie rozwiązania tego problemu. Zaprojektowane rozwiązanie wykorzystuje rozróżnianie tramwajów za pomocą unikalności trójki pól, a mianowicie *(line, brigade, checktimestamp)*. Para *(line, brigade)* okazała się niewystarczająca, ponieważ w obrębie jednej paczki zdarzały się sytuacje, kiedy nawet około 10 rekordów miało takie same wartości dla tych pól. W takim wypadku konieczne było wybranie kolejnego pola, aby zapewnić poprawną identyfikację.

Charakterystyka obszarów

W module *Collectora* zostały również zdefiniowane wybrane obszary, które będą później brane pod uwagę podczas sprawdzania występowania opóźnień tramwajów. Nad wyraz pomocna okazała się biblioteka przestrzenna Shapely. Służy ona do analizy i manipulacji danymi wektorowymi. Pozwala na tworzenie poligonów oraz sprawdzanie ich poprawności, a także różnego rodzaju operacje geometryczne.

Rozpatrywane przez nas obszary są wielokątami, a więc należą do klasy Polygon. Tworząc taki obiekt, podajemy listę współrzędnych. Możliwe jest również tworzenie wielokątów z "dziurami", jednak w tym przypadku nie było to potrzebne. Struktura obiektu opisującego dany obszar jest dwupoziomowa, ponieważ główny obszar jest podzielony na podobszary, które zbiegają się w środku i otaczają pasem fragment drogi prowadzącej do skrzyżowania. Przedstawione na rysunku 2.3 kształty podobszarów zostały wybrane w ten właśnie sposób z uwagi na precyzję określania lokalizacji - sporadycznie zdarza się, że któryś tramwaj zboczy nieznacznie ze swojej drogi. Takie sytuacje zostały uwzględnione poprzez wyznaczenie szerszego pasa otaczającego torowisko, aby nie zgubić żadnego tramwaju. Natomiast obszar nadrzędny to trójkąt lub wielokąt otaczający wszystkie swoje podobszary.



Rysunek 2.3. Przedstawienie obszaru i jego podobszaru na mapie

Obszar-rodzic opisany jest kilkoma podstawowymi polami. W przypadku podobszarów struktura jest analogiczna. Ich wyznaczone powierzchnie odpowiadają częściom skrzyżowania, zatem równocześnie okolicom wyznaczonych w ich obrębie przystanków. Z kolei ich nazwy zawierają główne kierunki w jakie odjeżdżają tramwaje z tychże przystanków. Przykładowy fragment dla obszaru Dworca Wileńskiego:

```
wilenski_area = {
    'name': 'Dw_wilenski',
    'id': 3,
    'area': Polygon([...]),
    'reference_point': {...},
    'subareas': [
      {
        'name': 'Kierunek:Wiatraczna/Kaweczynska_Bazylika',
        'parent_id': 3,
        'id': 31,
        'area': Polygon([...])
      },
      ...
    ]
  }
```

2.4 Architektura rozwiązania

Jako że zadaniem aplikacji jest informowanie o opóźnieniach w wybranych obszarach, każdy podobszar powinien być również zapisany i aktualizowany w bazie danych, a dokładniej w tabeli *StuckAreaState*. Składa się ona z czterech kolumn, z czego jedna to id obszaru przypisane podczas tworzenia obiektu opisującego obszar-rodzica. Pozostałe to:

- **last_observed_state** - stan obszaru wyznaczony w poprzedniej tranzycji
- **real_state** - rzeczywisty stan obszaru
- **last_state_transition** - rzeczywisty stan obszaru

2.4.3 Warstwa prezentacji

Obecne rozwiązanie

Warstwa prezentacji składa się z dwóch części. Najpierw zajmiemy się tą odpowiadającą za podgląd na mapie, ponieważ stanowi ona swego rodzaju bazę części związanej z powiadomieniami na serwisie społecznościowym.

Wykorzystałam OpenStreetMap [12], ponieważ ma ona otwartą licencję i została stworzona jako odpowiedź na zyskujące coraz większą popularność Google Maps. Istnieje kilka bibliotek Open Source umożliwiających wyświetlanie mapy w przeglądarce internetowej. Wybrałam OpenLayers [13], która jest jedną z najbardziej popularnych używanych do takich celów. Z jej pomocą wyświetlenie standardowej mapy jest stosunkowo proste - wystarczy stworzyć obiekt, przypisać go do elementu div oraz podać kilka podstawowych parametrów. Jeżeli nie podamy docelowego formatu współrzędnych, zostaną one przekonwertowane do domyślnego formatu używanego przez mapę. Dodawanie markerów na mapie jest równie proste. Wystarczy dodać nową warstwę, do której przekażemy wektor z punktami (*features*) do zaznaczenia na mapie.

Dla każdego tramwaju z paczki tworzymy odpowiadający mu punkt na mapie, czyli *point_feature*:

```
var point_feature = new ol.Feature({
    geometry: new ol.geom.Point(ol.proj.fromLonLat(coords))
});
```

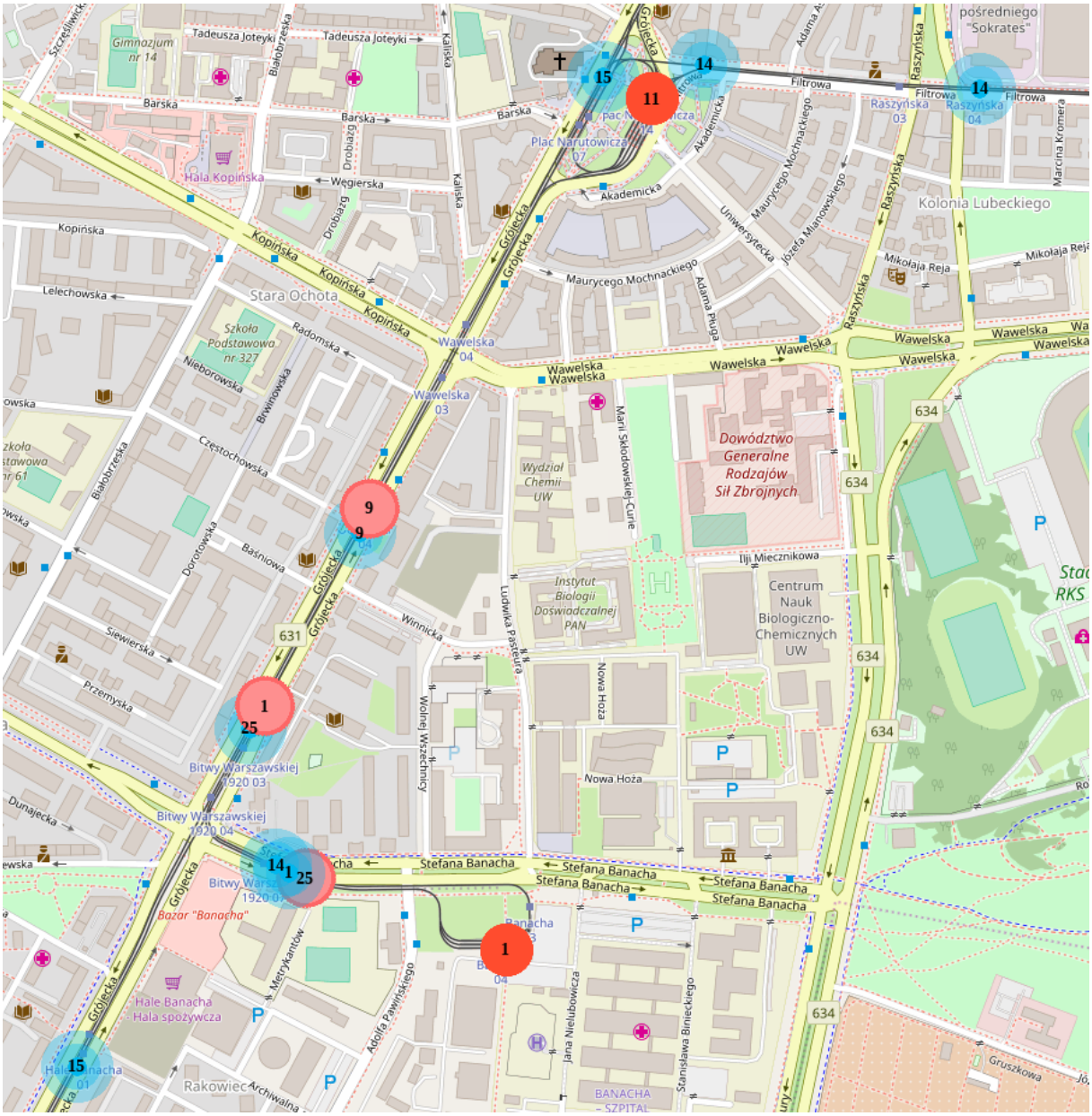
A następnie dodajemy go do wektora źródłowego z ustawionym stylem i wyłuskanym numerem linii:

```
point_feature.setStyle(icon_style);
point_feature.getStyle().getText().setText(tram['line'].toString());
trams_positions_source.addFeature(point_feature);
```

Wtedy możemy je umieścić w wybranej warstwie mapy:

```
// source vector init
trams_positions_source = new ol.source.Vector()
var tram_layer = new ol.layer.Vector({
  source: trams_positions_source
});
```

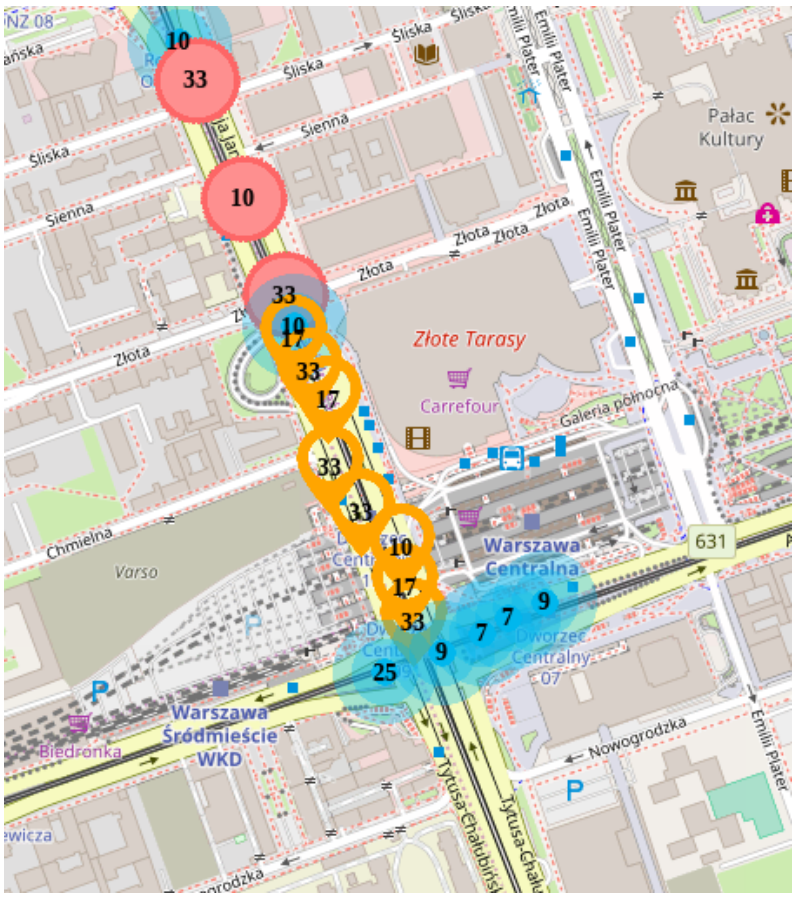
Rysunek 2.4 ukazuje podgląd, który jest efektem wykonania tych operacji.



Rysunek 2.4. Podgląd na mapie w sytuacji wystąpienia jedynie opóźnień pojedynczych tramwajów

Jak widzimy każde kółeczko jest na środku podpisane numerem linii danego tramwaju. Niebieskim kolorem oznaczone są tramwaje, które według przyjętych założeń poruszają się w sposób niebudzący zastrzeżeń. Natomiast czerwone kółeczka z falbanką dookoła przedstawiają tramwaje, które zakwalifikowaliśmy jako opóźnione. W celu odróżnienia opóźnionych pojazdów znajdujących się na pętli lub zajezdniach od tych aktualnie kursujących, używamy czerwono-pomarańczowych znaczników. Mamy jeszcze jeden rodzaj znaczników na mapie, a mianowicie oznaczenie tramwajów opóźnionych w rozważanych przez nas obszarach (rysunek 2.5). Przedstawiane są jako żółte pinezki.

2.4 Architektura rozwiązania



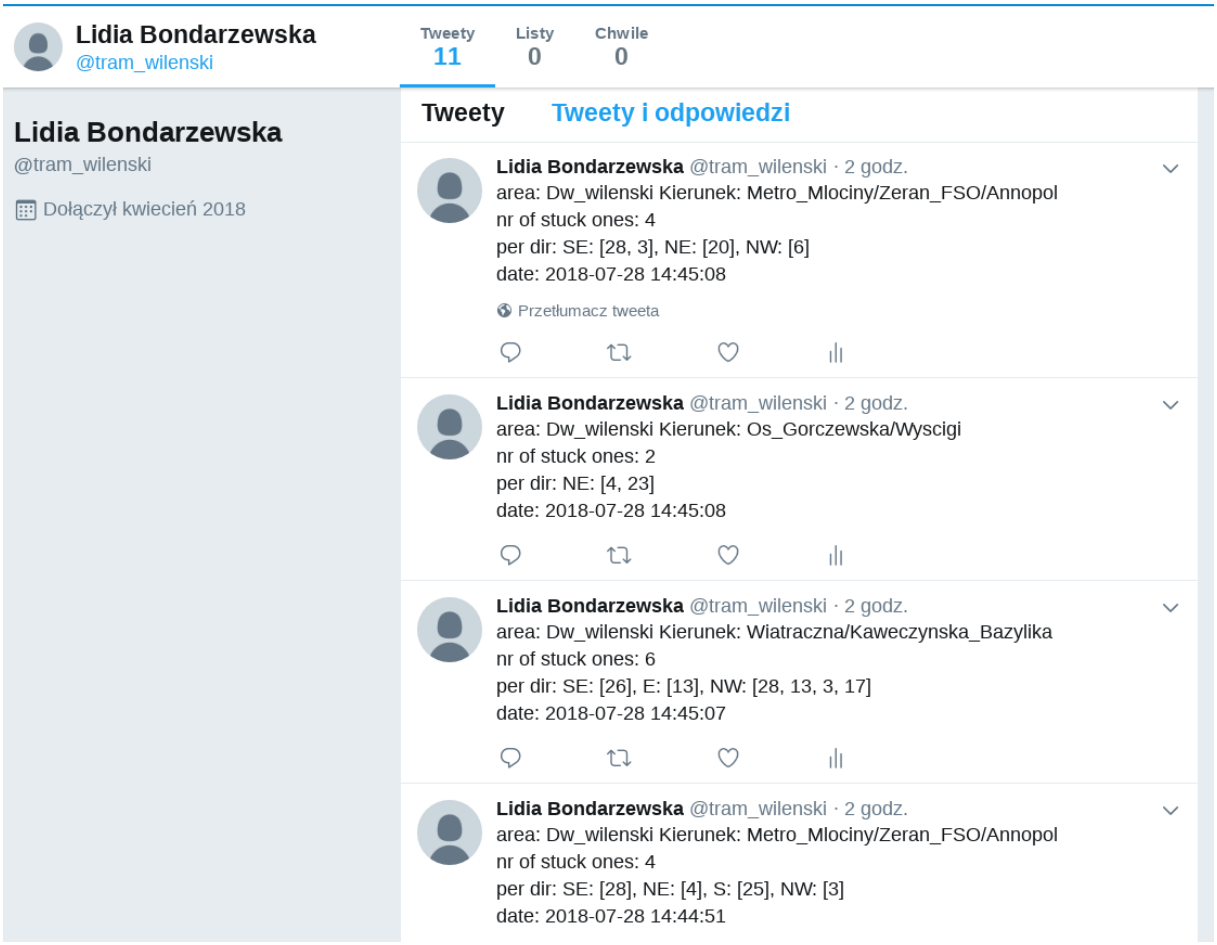
Rysunek 2.5. Podgląd na mapie z wystąpieniem grupowego zatorowania w jednym z badanych obszarów

Druga część odpowiada za publikowanie powiadomień z aplikacji. W tym projekcie zdecydowałam się na integrację z Twitterem, ponieważ planowana forma notyfikacji to wyłącznie tekst. Chcąc, by aplikacja była przyjazna użytkownikowi w jeszcze większym stopniu, zakładamy osobne konta dla każdego obszaru nadrzędnego. Dzięki temu użytkownik będzie miał możliwość dostawania powiadomień jedynie odnośnie interesujących go obszarów miasta poprzez subskrybowanie wybranych kont. Rysunek 2.6 przedstawia konto dedykowane okolicy Dworca Wileńskiego.

Powiadomienia są wysyłane w sytuacjach “zakorkowania się” obszaru. Publikacja postów jest uzależniona od wyznaczonego stanu obszaru, a dokładniej od wartości procentu zakorkowania, który jest obliczany w każdej iteracji algorytmu.

Struktura takiego posta składa się z czterech elementów i przedstawia się następująco:

- **area** - zawiera nazwę obszaru-rodzica oraz podobszaru
- **nr of stuck ones** - sumaryczna liczba tramwajów opóźnionych lub zakorkowanych w obrębie podobszaru
- **per dir** - lista kierunków geograficznych spośród (E, NE, N, NW, W, SW, S, SE), dla których zostały znalezione opóźnione tramwaje w danym podobszarze połączona z listą numerów linii tychże tramwajów
- **date** - data i godzina publikacji powiadomienia



Rysunek 2.6. Konto na Twitterze dedykowane okolicy Dworca Wileńskiego

Niezmiernie pomocna okazała się biblioteka Tweepy [14], która opakowuje API Twittera. Jej wykorzystanie ograniczam do uwierzytelnienia się w utworzonej aplikacji twitterowej za pomocą uzyskanych danych konfiguracyjnych. Podstawowy skrypt do komunikacji z Twitterem:

```
def get_api(cfg):
  auth = tweepy.OAuthHandler(cfg['consumer_key'], cfg['consumer_secret'])
  auth.set_access_token(cfg['access_token'], cfg['access_token_secret'])
  return tweepy.API(auth)

def sent_tweet():
  cfg = {
    "consumer_key"        : "VALUE",
    "consumer_secret"     : "VALUE",
    "access_token"        : "VALUE",
    "access_token_secret" : "VALUE"
    }
```

2.4 Architektura rozwiązania

```
api = get_api(cfg)
tweet = "Hello, world!"
status = api.update_status(status=tweet)
```

Niestety, w przypadku Twittera nie jest możliwe założenie kilku kont dla jednego użytkownika. Wobec tego konieczne było założenie kilku profili, a następnie przed publikacją tweeta wybranie właściwego konta, z którym chcemy się połączyć. Jednakże nie było to szczególne utrudnienie - odpowiednie konto jest wybierane na podstawie przekazanego identyfikatora obszaru.

Wizja rozwiązania na Facebooku

Interesujące jest zagadnienie próby przeniesienia publikowania postów na Facebooka. Motywacja jest zrozumiała - w Polsce Facebook cieszy się wyraźnie większą popularnością niż Twitter. Daje się to zauważyć szczególnie wśród środowiska akademickiego, które stanowi znaczną część potencjalnych użytkowników systemu, dlatego warto zadbać o ich komfort korzystania z aplikacji.

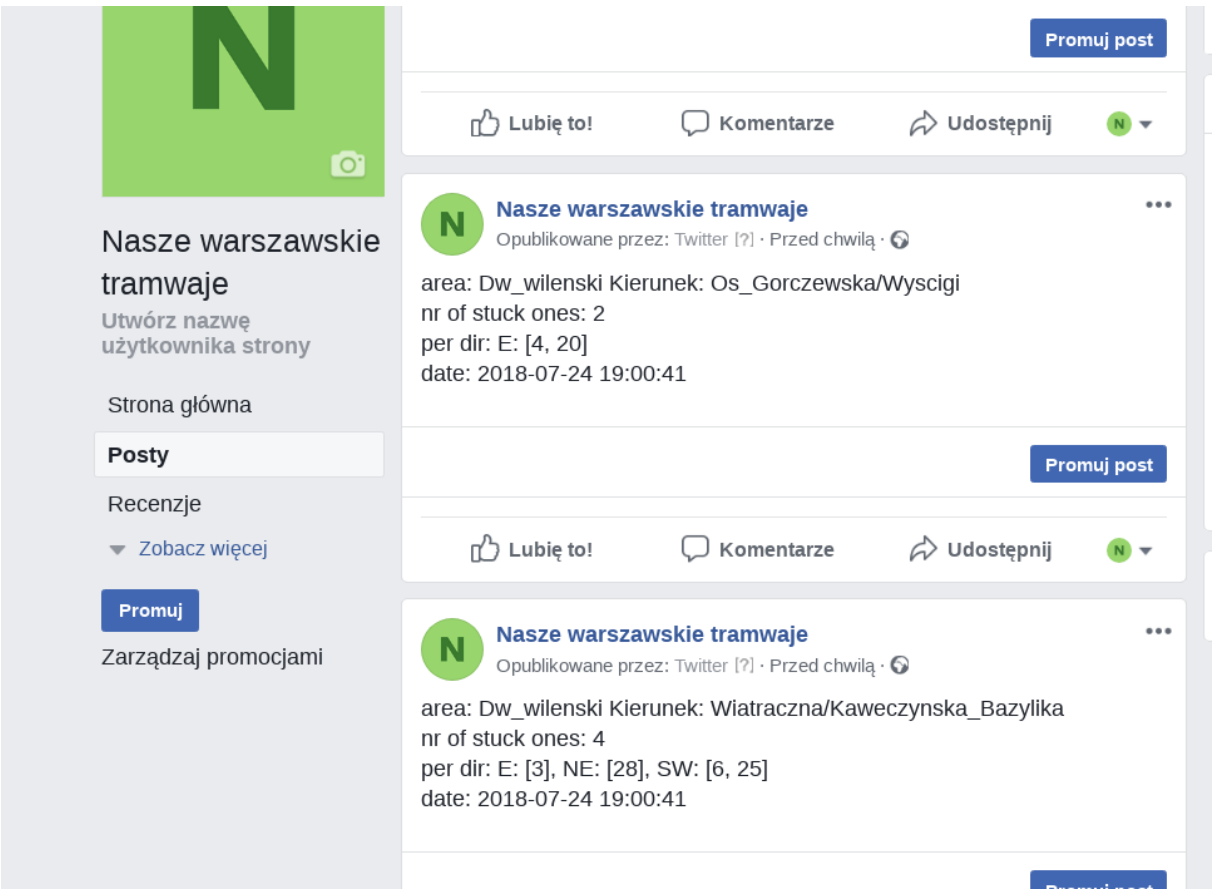
Najlepszym rozwiązaniem byłoby połączenie tych dwóch portali tak, aby nie stracić użytkowników przyzwyczajonych do Twittera bądź posiadających konto jedynie na tym portalu. Okazuje się, że Twitter i w tym aspekcie jest nam przychylny, ponieważ oferuje poszukiwaną przez nas funkcjonalność [15].

Przed rozpoczęciem konfiguracji należy się upewnić, czy mamy stworzoną stronę na Facebooku lub posiadamy uprawnienia administratora dla tej strony. Jest to istotne, bowiem w innych przypadkach nie będziemy mogli publikować postów na tejże stronie.

Pierwszym krokiem jest połączenie naszego konta na Twitterze z profilem na Facebooku. W tym celu przechodzimy do zakładki Ustawienia → Aplikacje, a następnie wybieramy opcję Połącz z Facebookiem. Na tym etapie wybieramy też ustawienia prywatności związane z widocznością naszych postów dla pozostałych użytkowników. Następnie na tym samym panelu wybieramy stronę, z którą chcemy połączyć dane konto i wyrażamy zgodę na publikowanie na niej.

Taki sposób jest może mniej wygodny, ale daje oczekiwane rezultaty. Jest to jedyna działająca droga jaką udało mi się znaleźć wobec wprowadzonych w bieżącym roku zmian w polityce Facebooka odnośnie publikowania postów z aplikacji jako zalogowanych użytkowników [16].

Wobec tego plan rozwiązania przedstawia się następująco. Dla każdego konta na Twitterze związanego z obszarem tworzymy bliźniaczą stronę na Facebooku oraz łączymy je ze sobą w wyżej opisany sposób. Wybór strony, na którą chcemy wysłać powiadomienie jest dokonywany na poziomie tworzenia tweeta. W pliku konfiguracyjnym mamy listę danych potrzebnych do logowania i autoryzacji dla każdego konta na Twitterze. Na podstawie przekazanego identyfikatora obszaru wybierane jest odpowiednie konto i jego dane konfiguracyjne, po czym tworzona jest treść tweeta z podanych parametrów, a następnie jest publikowany. Wygląd takiej strony testowej ukazuje rysunek 2.7.



Rysunek 2.7. Wygląd strony testowej z powiadomieniami na Facebooku

Zatem udało nam się połączyć, ale należy jeszcze sprawdzić ile takich stron możemy założyć z jednego profilu. W tym przypadku nie ma wyznaczonego limitu, jednakże są pewne ograniczenia - strona musi dotyczyć czegoś konkretnego oraz powinniśmy mieć podstawy do utworzenia strony o wybranej treści.

Pozostaje jeszcze kwestia dziennego limitu postów. Jest on zdecydowanie mniejszy niż na Twitterze, lecz dokładna wartość nie jest upubliczniona, ponieważ ułatwiłoby to unikanie wykrycia spamu. Rekomendowana jest publikacja do 5 postów dziennie, co w naszym przypadku jest osiągalne, ponieważ rozpatrywane przez nas korki tramwajowe nie występują w tak wysokim natężeniu, aby liczba oraz częstotliwość publikacji na stronie stała się problemem. Treść powiadomień również ma znikome szanse zakwalifikowana jako spam, bowiem niesie ze sobą przekaz informacji a nie nakładania do udostępniania postów lub wchodzenia na inne strony.

2.5 Implementacja

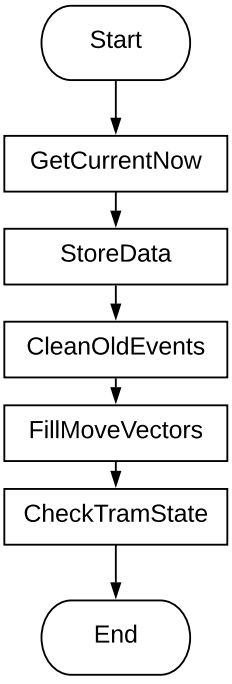
2.5 Implementacja

2.5.1 Własne komendy do Django

Jako że zapytanie do API jest wykonywane co pół minuty, program zawiera nieskończoną pętlę, która w tym interwale wywołuje odpowiednie funkcje. Są to zatem powtarzalne czynności, które można łatwo zautomatyzować. Django posiada mechanizm pozwalający użytkownikowi na dodawanie nowych komend uruchamianych przez *manage.py* [17]. Dlatego korzystanie z tego frameworku jest takie wygodne.

W tenże sposób dodałam komendy *collect_trams* oraz *simulate_trams*, która wywołuje pierwszą na danych zebranych w folderze i symuluje działanie w czasie rzeczywistym. Komenda *collect_trams* to niemal serce aplikacji. Możemy ją wywołać z dwoma parametrami:

- **prepare_db** - przygotowuje część bazy danych odpowiedzialną za zdefiniowane obszary. Jeżeli ich elementy zostały zmienione lub dodano/usunięto obszary zapisujemy to w bazie przed rozpoczęciem pracy
- **run_infinitely** - wywołuje funkcję *HandleOneShot* obsługującą jeden przebieg pętli w określonym interwale, tzn. 30 sekund. Przekazuje do niej otrzymane z zapytania dane. Jej strukturę przedstawia rysunek 2.8:



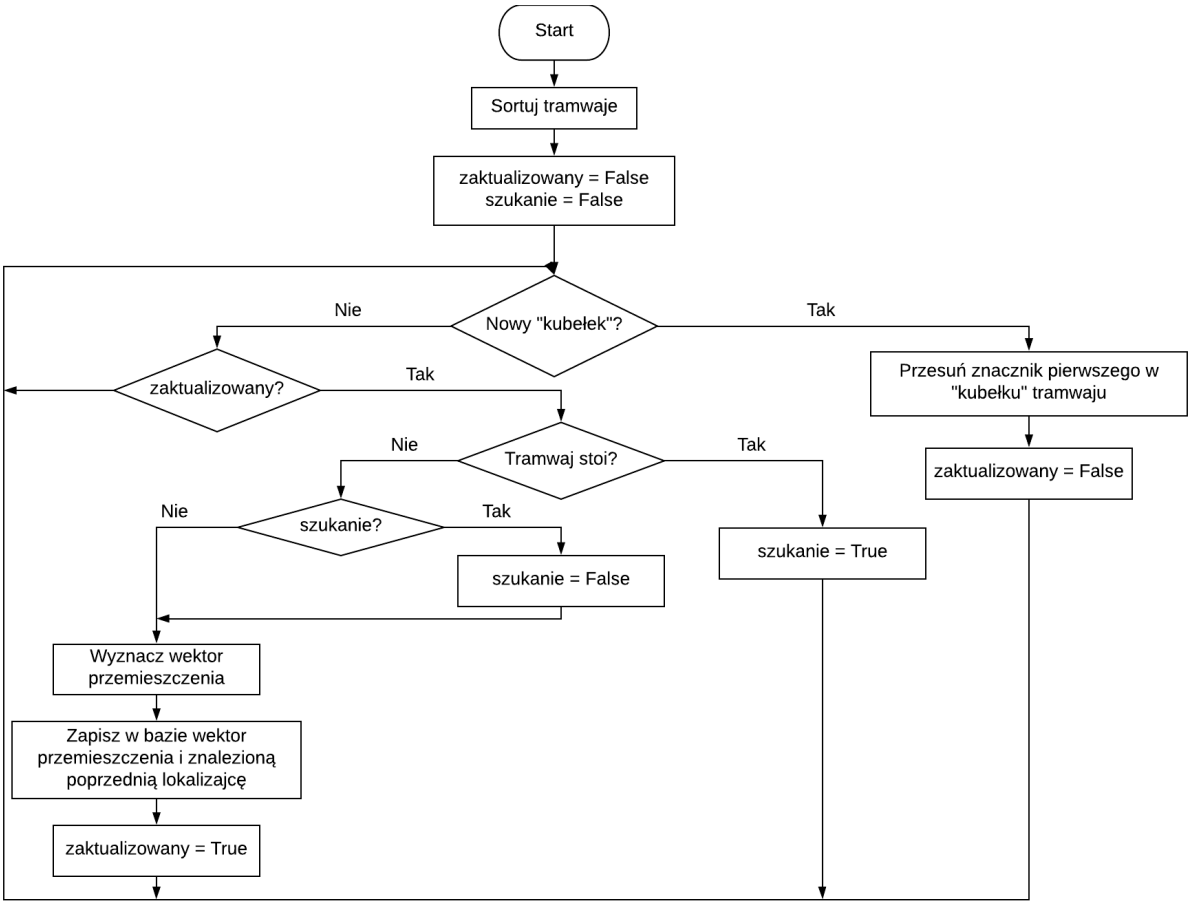
Rysunek 2.8. Schemat blokowy funkcji *HandleOneShot*

Jako parametr przekazujemy jej paczkę tramwajów pobraną przez moduł collectora, a także aktualny czas. Po kolei zapisuje dane tramwajów do bazy, usuwa odpowiednio stare wpisy, oblicza wektory przemieszczenia, wyszukuje tramwaje opóźnione, następnie wyznacza ich stan, by na końcu wyznaczyć stany w jakich znajdują się obszary. W niej więc odbywa się cały proces przetwarzania informacji.

Komenda *simulate_trams* wywołuje *collect_trams* na danych, które są już zapisane w plikach. Opiera się na symulacji generatora czasu, aby usprawnić proces testowania funkcjonalności. Zostanie ona bardziej szczegółowo omówiona w rozdziale poświęconemu testowaniu aplikacji.

2.5.2 Wyznaczanie wektora przemieszczenia oraz kierunku pojazdów

Chcąc wyznaczyć kierunek geograficzny w jakim porusza się tramwaj, musimy najpierw znaleźć jego wektor przemieszczenia. Do tego wykorzystujemy funkcję *FillMoveVectors*, której struktora została przedstawiona na rysunku 2.9. Na początku wybieramy z bazy zbiór obiektów, które są odpowiednio stare i sortujemy je. To bardzo ważna część algorytmu - porządkujemy je po numerze linii, numerze brygady i malejąco po czasie.



Rysunek 2.9. Schemat blokowy funkcji *FillMoveVectors*

W pętli przechodzącej po zbiorze tramwajów wyznaczamy wektor przemieszczenia dla dwóch pierwszych elementów z “kubełka”, gdzie przez “kubełek” rozumiemy podzbiór, w którym dla każdych dwóch sąsiednich elementów para (numer linii, numer brygady) jest taka sama. W przypadku, kiedy tramwaj stoi szukamy w “kubełku” pierwszego elementu, który ma współrzędne różne od obecnie znalezionego. Dzięki temu unikniemy wyznaczenia błędnego kierunku.

2.5 Implementacja

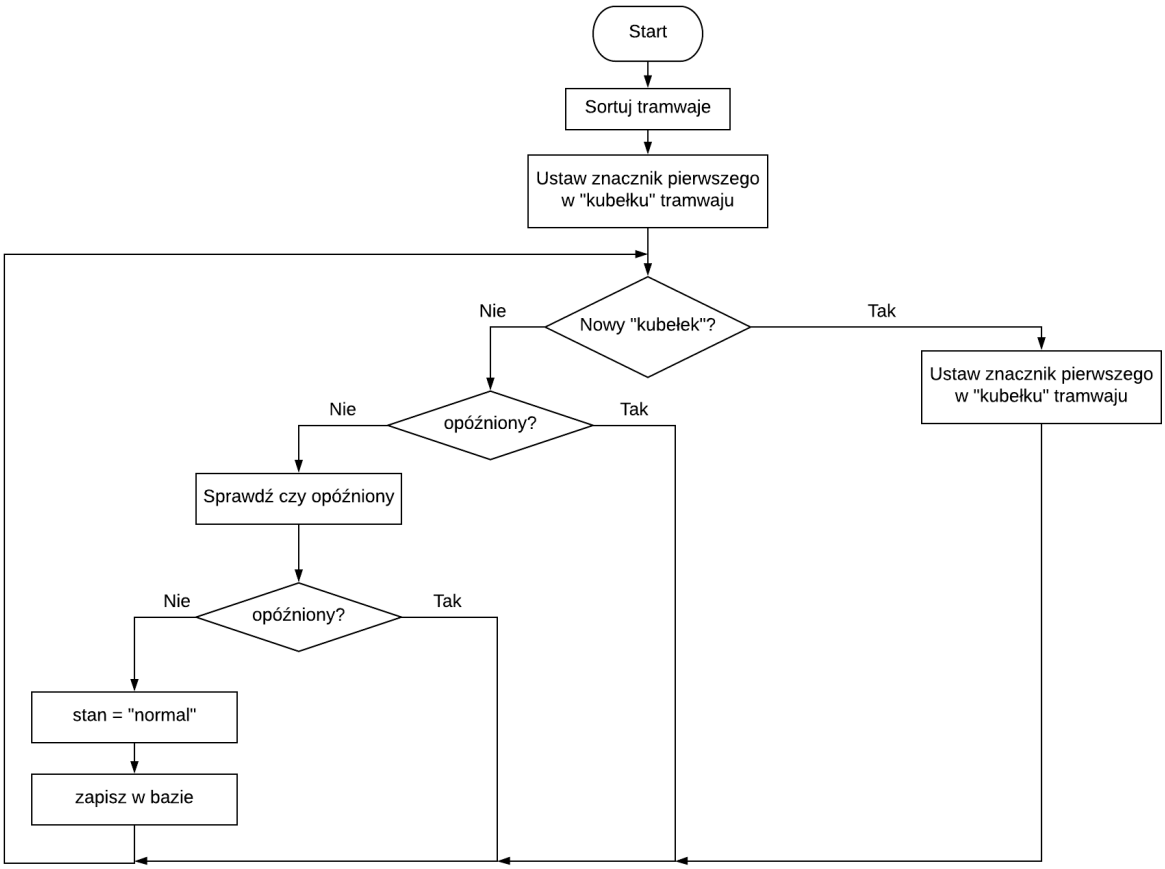
Za ten fragment algorytmu odpowiada funkcja *VectorToCompass*. Wykorzystuje ona wektor przemieszczenia obliczony za pomocą funkcji *ComputeMoveVector*, do której przekazujemy w parametrach współrzędne geograficzne dwóch wyznaczonych wcześniej punktów.

```
def VectorToCompass(self, move_lon, move_lat):
    angle = math.atan2(move_lat, move_lon)
    octant = int(8 * angle / (2 * math.pi) + 8.5) % 8
    return ["E", "NE", "N", "NW", "W", "SW", "S", "SE"][octant]
```

Najprostszym rozwiązaniem na wyznaczenie kierunku jest znalezienie kąta danego wektora z wykorzystaniem funkcji cyklometrycznej arcus tangens, która zwraca kąt w radianach. Następnie konwertujemy go do ósemek koła, dzieląc przez 2π i mnożąc przez 8. Otrzymany wynik zaokrąglamy do najbliższej liczby całkowitej. Na końcu wyznaczamy resztę z dzielenia przez 8, aby znaleźć odpowiednią ćwiartkę. W celu zapewnienia zwrócenia dodatniego wyniku dodajemy do wyniku przed operacją modulo 8, czyli jeden pełny obrót. Dodatkowo, aby zabezpieczyć się przed błędem zaokrąglania dodajemy 0.5 (w niektórych językach domyślna konwersja typu float do integer zaokrągla liczby ujemne do zera zamiast w dół). Dlatego w sumie dodajemy 8.5. Finalnie wybieramy właściwy kierunek z listy.

2.5.3 Znajdowanie opóźnionych pojazdów

Rozpoczynamy od wybrania z bazy tramwajów, które mogą być potencjalnie opóźnione, przy czym mają być one nie starsze niż sprzed pięciu minut. Sortujemy je jak w funkcji *FillMoveVectors*, a mianowicie po numerze linii, numerze brygady i malejąco po czasie. W tym momencie należy zauważyć, że każdy tramwaj jest zapisywany do bazy domyślnie jako opóźniony, to znaczy ze stanem “stuck”. Tutaj również podobnie poruszamy się po zdefiniowanych wyżej “kubełkach”. Algorytm został przedstawiony na rysunku 2.10.



Rysunek 2.10. Schemat blokowy funkcji *FindStuckTrams*

Do sprawdzenia, czy tramwaj jest opóźniony, wykorzystujemy funkcję *IsStuck*, której przekazujemy dwie pozycje tramwaju. Obliczanie odległości między danymi dwoma punktami na kuli wyznaczamy metodą Haversine'a, która oblicza odległość geodezyjną i zwraca wynik w metrach. Jeżeli obliczona odległość jest mniejsza niż przyjęte kryterium (400 metrów), uznajemy go za zakorkowany.

2.5.4 Wyznaczania stanu tramwaju i obszaru

Na tym etapie do dalszej obróbki wybieramy tramwaje z najświeższego pomiaru. Będziemy wykorzystywać listy z obszarami, w których opóźnienia nie są problemem - zajezdnie, pętle, a także te w których ich oczekujemy (*STUCK_AREAS_TO_CHECK*), a zawieranie się tramwaju w jednym z tych obszarów będziemy oznaczać odpowiednimi flagami. Jako że wyznaczamy "procent zakorkowania" dla każdego podobszaru, musimy też w jakiś sposób zliczać tramwaje. Idealnie nadaje się do tego *defaultdict* - umożliwia on ustawienie domyślnej wartości, dzięki czemu łatwiejsza jest późniejsza inkrementacja konkretnego licznika, także w przypadku, gdy chcemy to zrobić dla klucza, który jeszcze nie istnieje w słowniku. Oczywiście są one czyszczone po każdym przebiegu głównej pętli.

Pierwsza część algorytmu prezentuje się następująco. Dla każdego tramwaju z wybranego zbioru zaczynamy od konwersji jego wektora przemieszczenia na kierunek geograficzny. Następnie

2.5 Implementacja

sprawdzamy czy zawiera się w którymś ze zdefiniowanych obszarów i ustawiamy odpowiednie flagi, które wykorzystamy później do wyznaczenia bardziej precyzyjnego stanu pojazdu niezbędnego do oznaczenia go na mapie. Jeżeli badany tramwaj znajduje się w którymś z obszarów z listy *STUCK_AREAS_TO_CHECK*, sprawdzamy w którym dokładnie z jego podobszarów się zawiera. W tym momencie inkrementujemy też odpowiednie liczniki dla tego podobszaru - w zależności od wyznaczonego w poprzednim kroku stanie pojazdu, czyli “normal” lub “stuck”, a także licznik wszystkich tramwajów zawartych danym podobszarze. Następnie przechodzimy do wyznaczenia stanu pojazdu, uwzględniając jego ewentualne zawieranie się w którymś z przedstawionych rodzajów obszarów. Po tym kroku aktualizujemy w bazie wyznaczony stan - możemy otrzymać “stuck”, “stuck_in_jam”, “in_loop”. Jeżeli nic się nie zmieniło zostajemy przy “normal”. Jak pamiętamy z poprzedniego rozdziału, te stany mają odzwierciedlenie w kolorach znaczników na mapie.

Jednak najważniejszym elementem tego algorytmu jest wyznaczenie stanu obszaru, które umożliwia wysyłanie powiadomień na Twittera. Wyróżniamy następujące stany, w jakich może znajdować się obszar - “normal”, “stuck” oraz “potentially_stuck” i “potentially_normal”, które są stanami pośrednimi. Wykorzystujemy je do opóźnienia tranzycji, aby mieć pewność, że zator lub swobodny przejazd rzeczywiście dla tego obszaru istnieje, a także zabezpieczyć się przed czasami wybrakowanymi odpowiedziami z API. Należało również określić wartość progową “procentu zakorkowania” służącą do kwalifikowania obszaru jako potencjalnie zablokowanego oraz wolnego od zatorów - wynoszą one odpowiednio 40% oraz 5%. Wobec tego wyróżniamy następujące tranzycje:

1. **normal → potentially_stuck**

   Przy jednorazowym przekroczeniu warunku zakorkowania, czyli w sytuacji, gdy "procent zakorkowania" wynosi co najmniej 40%. Jest to proste przejście - nie wymaga sprawdzania czasu jaki upłynął od ostatniej tranzycji. W tej chwili zaczyna się proces uznawania obszaru za zatorowany poprzez przypisanie mu stanu pośredniego.

2. **potentially_stuck → stuck**

   Dodatkowym warunkiem jest sprawdzenie odstępu czasowego od ostatniej zmiany stanu danego obszaru. Przyjmujemy, że powinien on wynosić co najmniej 5 minut. Dopiero wtedy w sytuacji, gdy przez cały ten czas "procent zakorkowania" nie spadł poniżej wartości progowej, przejście może zostać wykonane.

3. **potentially_stuck → normal**

   Przy jednorazowym spadku wartości "procentu zakorkowania" poniżej 40%. Proces uznawania obszar za zatorowany zostaje zatrzymany, ponieważ zator nie utrzymał się wystarczająco długo.

4. **stuck → potentially_normal**

   Przy jednorazowym spadku wartości "procentu zakorkowania" poniżej 40%. W tej chwili zaczyna się proces uznawania obszaru za wolny od zatorów poprzez przypisanie mu stanu pośredniego.

5. **potentially_normal → normal**

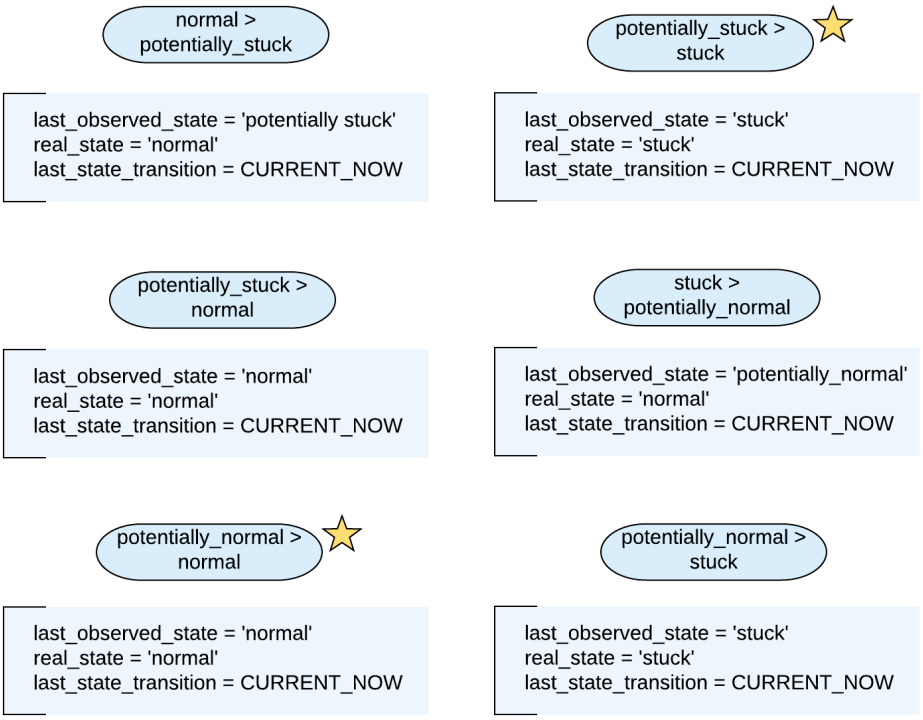
Tutaj również potrzebny jest dodatkowy warunek - sprawdzenie odstępu czasowego od ostatniej zmiany stanu danego obszaru. Analogicznie jak w przejściu "potentially_stuck → stuck" sprawdzamy, czy w ciągu ostatnich 5 minut wartość "procentu zakorkowania" utrzymała się na poziomie co najwyżej 5%.

6. **potentially_normal → stuck**

   To przejście oznacza, że obszar był przed chwilą w pewnym stopniu zatorowany i trwa proces uznawania go za wolny od zatorów. Dlatego wystarczy jednorazowe przekroczenie warunku zakorkowania, aby został uznany za obszar zatorowany bez konieczności wprowadzania go w stan pośredni.

Istotne jest spostrzeżenie, że zaprojektowanie tranzycji w taki właśnie sposób uniemożliwia wykonanie się więcej niż jednej dla danego obszaru podczas jednego obiegu pętli.

Przypomnijmy, że w bazie każdy obszar jest opisany za pomocą pól *id*, *last_observed_state*, *real_state* oraz *last_state_transition*. Schemat przedstawiony na rysunku 2.11 obrazuje ich aktualizację dla każdej tranzycji. Gwiazdką zostały oznaczone przejścia, które wymagają sprawdzenia czasu jaki upłynął od ostatniej zmiany stanu dla danego obszaru.



Rysunek 2.11. Schemat aktualizacji atrybutów obszaru podczas tranzycji

3 Testowanie

Nieodłącznym elementem pracy nad budowaniem aplikacji była symulacja API. Konieczne było nie tylko w miarę możliwości zebranie danych o sytuacjach wyjątkowych w różnych obszarach miasta, lecz także o niezaburzonym kursowaniu tramwajów w różnych porach. W tym celu powstała komenda *simulate_trams* oraz prosty skrypt bash przygotowujący bazę i wywołujący odpowiednie polecenia, aby zautomatyzować i przyspieszyć proces testowania funkcjonalności.

```
#!/bin/bash

set -e

source venv/bin/activate
rm -f db.sqlite3
./manage.py migrate
./manage.py collect_trams --prepare_db
./manage.py simulate_trams --steps=number
```

Do komendy *simulate_trams* został dodany opcjonalny argument –steps. Służy on do zatrzymania symulacji po wybranej liczbie kroków, czyli liczbie przetworzonych plików z danymi z wybranego katalogu. Dzięki temu możemy analizować powtarzalne zachowanie na tym samym zbiorze danych.

Podgląd na mapie w czasie rzeczywistym umożliwił zweryfikowanie poprawności działania modułu odpowiedzialnego za wysyłanie powiadomień. Posługując się dodatkowo panelem administratora Django, możliwe było sprawdzenie czy przejścia między stanami obszarów zachodzą w oczekiwany sposób, a tym samym czy posty są publikowane w odpowiednim momencie. Ułatwiło to również weryfikację zgodności treści posta z warunkami panującymi w okolicach przejazdu w danym obszarze w danej chwili. Możliwe było bowiem zatrzymanie podglądu w dowolnym momencie i sprawdzenie, czy wymienione w powiadomieniu opóźnione tramwaje rzeczywiście takie są i czy poruszają się określonym przez nas kierunku.

W celu bardziej obiektywnej oceny łatwości w korzystaniu z aplikacji, została ona udostępniona grupie studentów złożonej z sześciu osób. Reprezentowali oni zarówno kierunki informatyczne jak i przyrodniczo-humanistyczne. Szczególne zainteresowanie wzbudził podgląd na mapie z rozróżnieniem stanów tramwajów. Uznali również, że wybranie Twittera jako kanału dystrybucji informacji jest dla nich wygodnym rozwiązaniem, ponieważ nie wymusza na nich pobierania kolejnej aplikacji. Jest również poręczne, gdyż sprawdzenie nie zabiera dodatkowo czasu - po prostu dostaną jedno powiadomienie więcej, które mogą zignorować, jeżeli w danej chwili nie są zainteresowani.

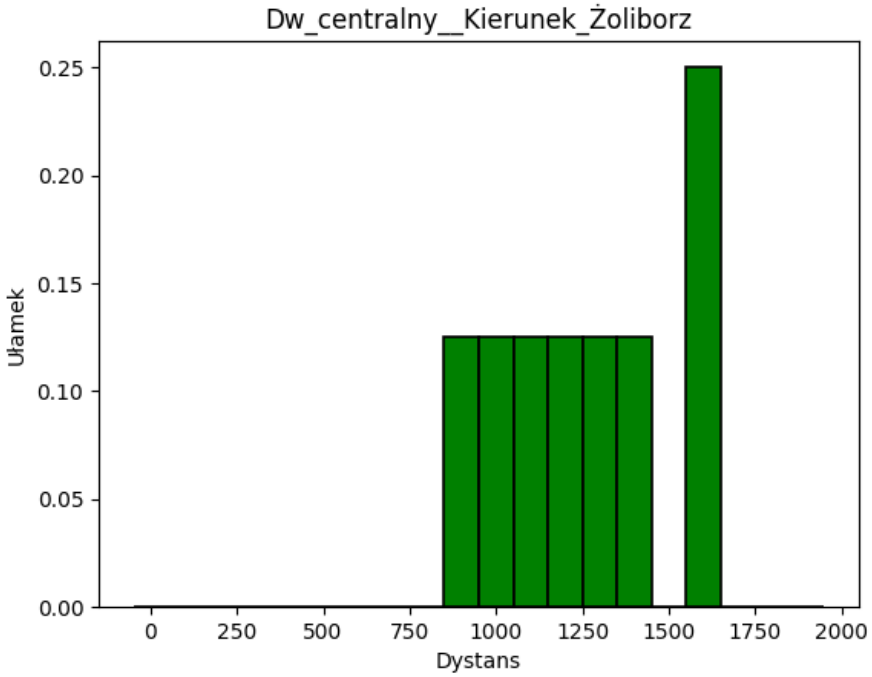
4 Podsumowanie

4.1 Prezentacja wyników

W ramach niniejszej pracy inżynierskiej powstała aplikacja sieciowa, która wspomaga przemieszczanie się tramwajami po Warszawie. Użytkownik ma dostęp do dwóch funkcjonalności - podglądu tramwajów na mapie w przeglądarce w czasie rzeczywistym ze znacznikami określającymi stan pojazdu oraz kanału dystrybucji informacji na Twitterze poprzez dedykowane konta, których wygląd został szczegółowo przedstawiony w poprzednich rozdziałach.

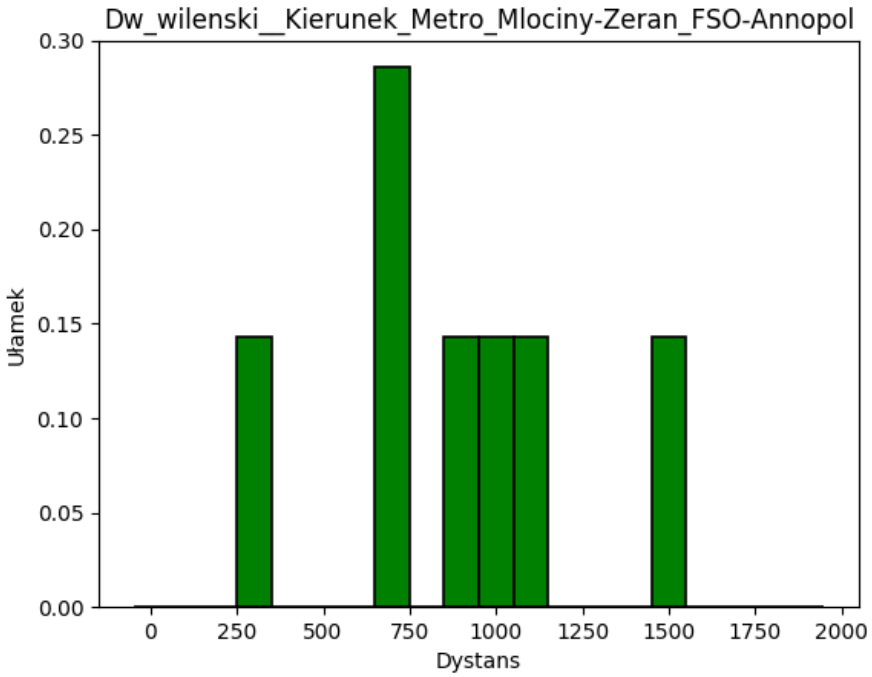
Jednak cały mechanizm opiera się pewnym przyjętym kryterium - tramwaj zostaje zaklasyfikowany jako opóźniony, jeżeli w ciągu 5 minut przejechał mniej niż 400 metrów. Sprawdźmy zatem, czy dobrane kryterium spełnia swoją rolę. Do analizy wykorzystamy kilka obszarów z różnych części miasta, które często są uznawane za jedne z newralgicznych punktów komunikacji tramwajowej w Warszawie.

Oto moja analiza przypadków, gdy w rozważanych obszarach nie ma korka (rysunki 4.1 - 4.4).

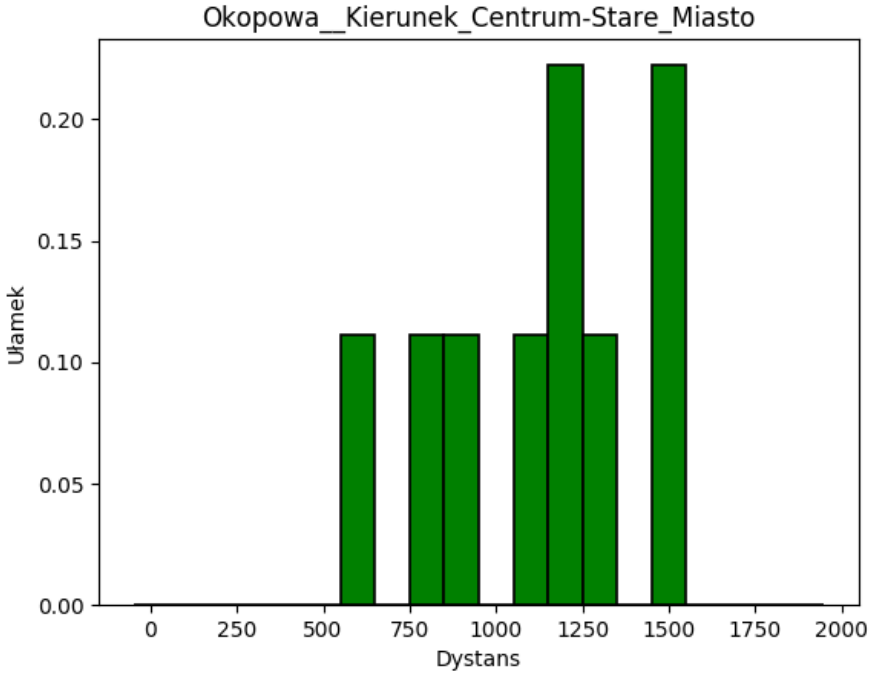


Rysunek 4.1. Histogram odległości pokonanych przez tramwaje w 15 minutowych oknach, gdy nie ma zatorów w podobszarze Dworca Centralnego

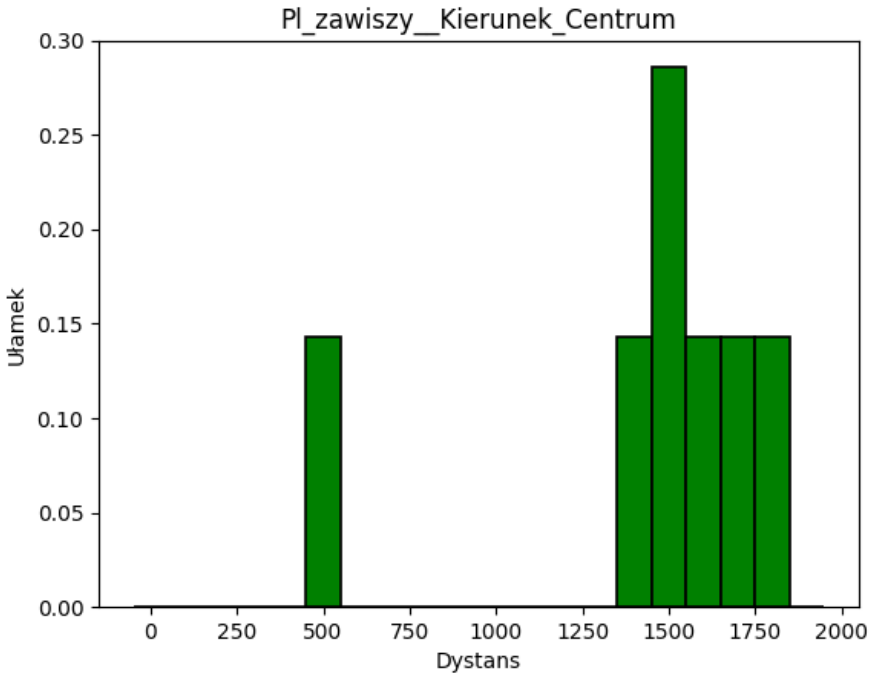
4.1 Prezentacja wyników



Rysunek 4.2. Histogram odległości pokonanych przez tramwaje w 15 minutowych oknach, gdy nie ma zatorów w podobszarze Dworca Wileńskiego



Rysunek 4.3. Histogram odległości pokonanych przez tramwaje w 15 minutowych oknach, gdy nie ma zatorów w podobszarze Okopowej

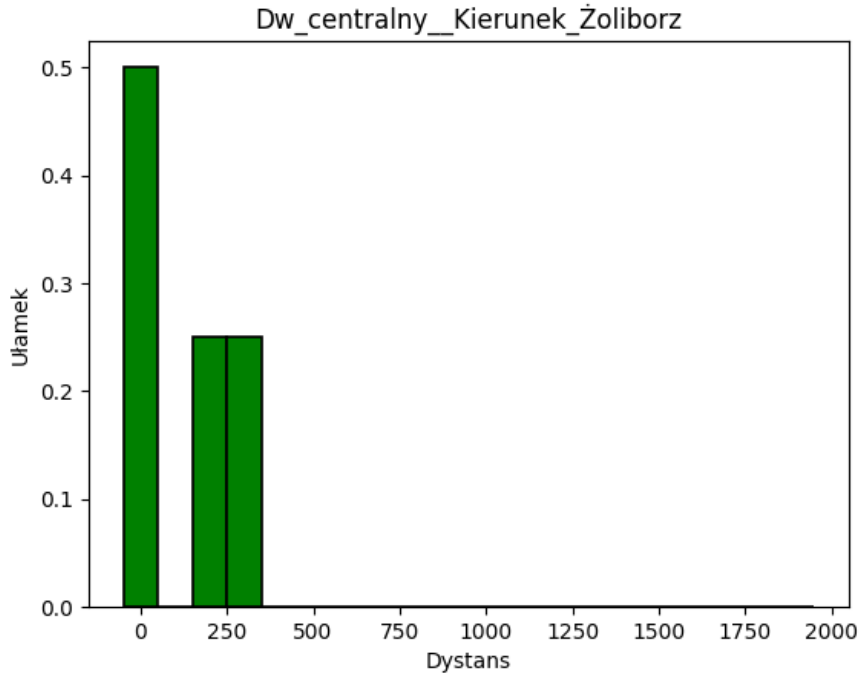


Rysunek 4.4. Histogram odległości pokonanych przez tramwaje w 15 minutowych oknach, gdy nie ma zatorów w podobszarze Pl. Zawiszy

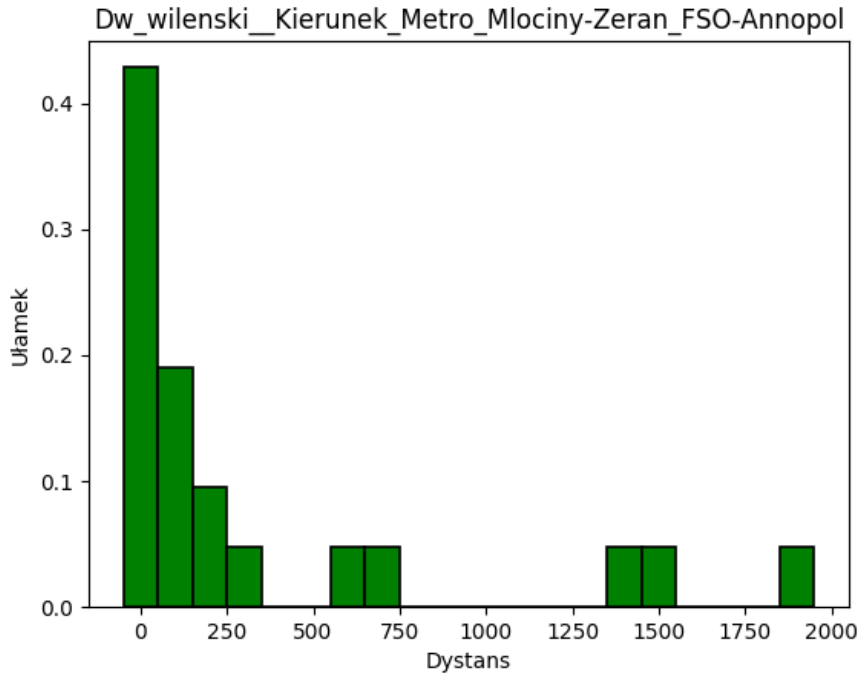
Zauważmy, że dla każdego z nich liczba tramwajów, które pokonały mniej niż 400 metrów jest mniejsza niż 5%, które jak pamiętamy jest jednym z warunków ostatecznego przejścia do stanu “normal”, czyli wolnego od zatorów.

Zamieszczone poniżej histogramy (rysunki 4.5 - 4.8) przedstawiają analizę przypadków, gdy w rozważanych obszarach występują zatory.

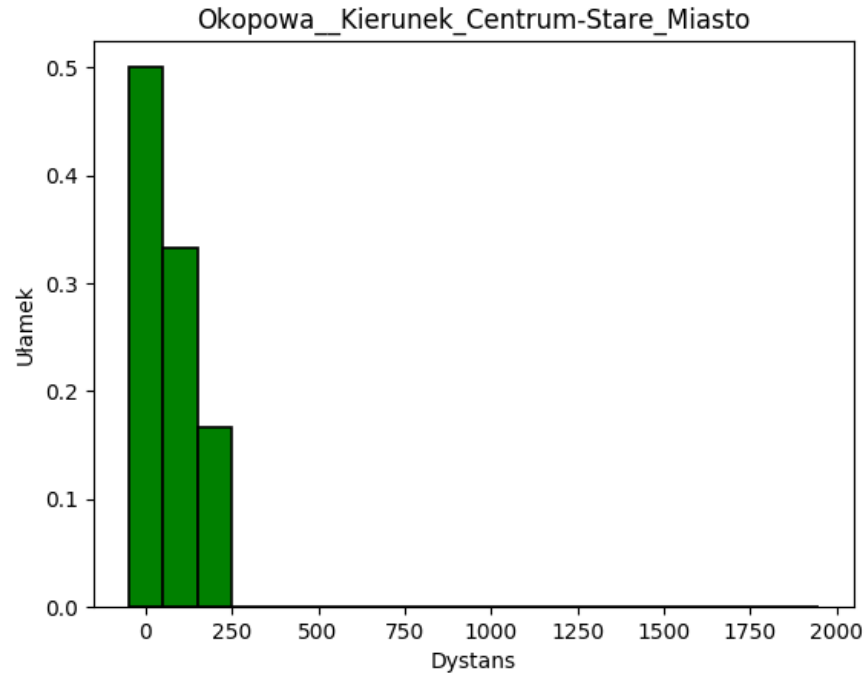
4.1 Prezentacja wyników



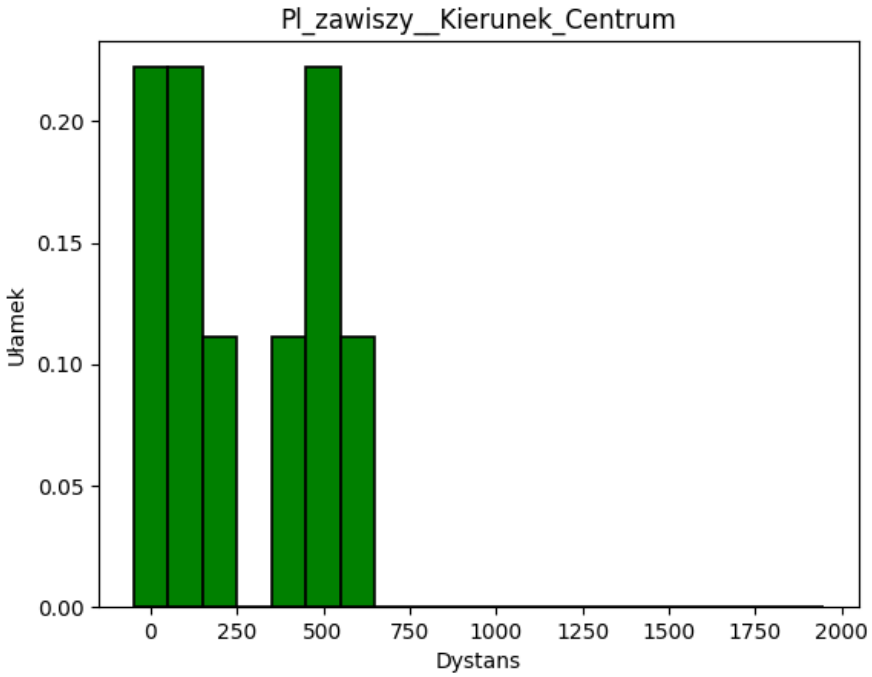
Rysunek 4.5. Histogram odległości pokonanych przez tramwaje w 5 minutowych oknach, gdy występują zatory w podobszarze Dworca Centralnego



Rysunek 4.6. Histogram odległości pokonanych przez tramwaje w 5 minutowych oknach, gdy występują zatory w podobszarze Dworca Wileńskiego



Rysunek 4.7. Histogram odległości pokonanych przez tramwaje w 5 minutowych oknach, gdy występują zatory w podobszarze Okopowej

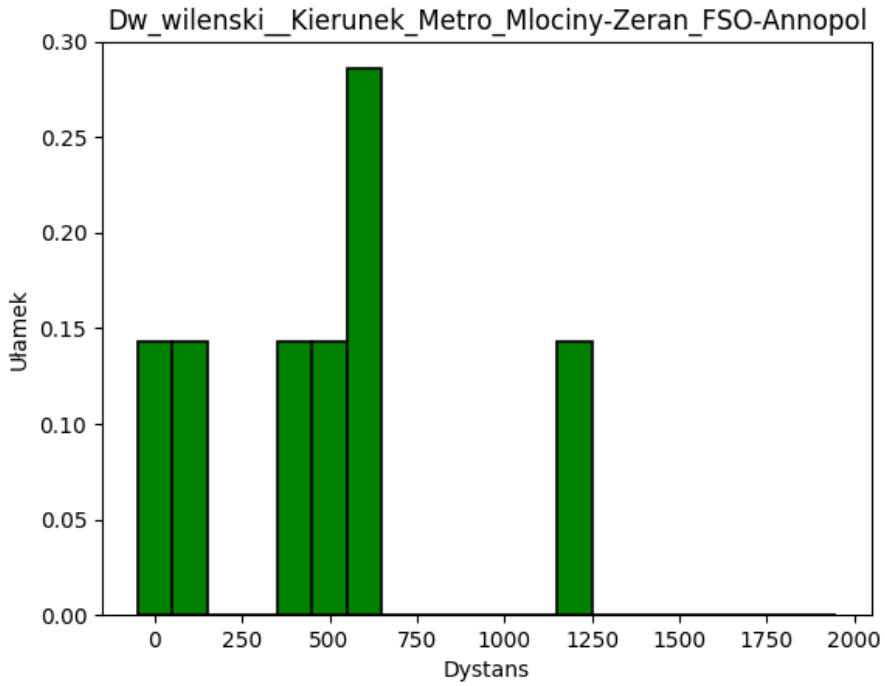


Rysunek 4.8. Histogram odległości pokonanych przez tramwaje w 5 minutowych oknach, gdy występują zatory w podobszarze Pl. Zawiszy

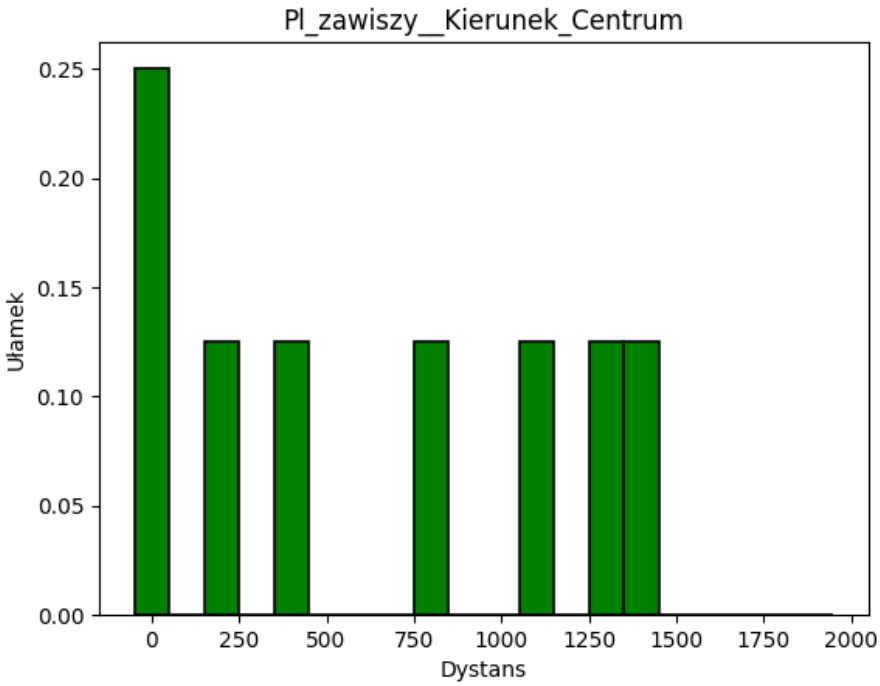
4.1 Prezentacja wyników

W tym przypadku prezentują się one zupełnie inaczej. Liczba tramwajów, które pokonały mniej niż 400 metrów powinna wynosić co najmniej 40%, a tutaj mamy przykład sytuacji, kiedy stanowi zdecydowaną większość. Na pierwszy rzut oka może się wydawać, że tylko 3 przedziały w histogramie to mało. Jednak zauważmy, że podobego wyniku oczekiwaliśmy, ponieważ tramwaje wtedy stoją lub pokonują dość krótkie odcinki.

Spójrzmy jeszcze na kilka histogramów (rysunki 4.9 - 4.10) dla momentów, kiedy obszar znajduje się w stanie przejściowym “potentially_stuck”, czyli dopiero rozpoczął się proces uznawania go za obszar w pewnym stopniu zatorowany.



Rysunek 4.9. Histogram odległości pokonanych przez tramwaje w 15 minutowych oknach, gdy zaczyna powstawać zator w podobszarze Okopowej



Rysunek 4.10. Histogram odległości pokonanych przez tramwaje w 15 minutowych oknach, gdy zaczyna powstawać zator w podobszarze Pl. Zawiszy

Widzimy, że próg 40% został przekroczony, a więc pierwsza tranzycja wykonała się zgodnie z przyjętymi założeniami. Tak jak oczekiwaliśmy, wartości pokonanego dystansu należą przedziałów leżących po obu stronach przyjętej granicy oraz nie tylko w jej okolicach - są bardziej rozproszone niż w poprzednio analizowanych przypadkach.

Jak widać z mojej analizy jasno wynika, że dobrane kryterium jest bardzo dobrym wyznacznikiem. Dzięki temu aplikacja działa zgodnie z przyjętymi założeniami.

Przeprowadziłam również testy wydajnościowe, które zostały uruchomione na laptopie Sony VAIO Pro 13. Jego podstawowe parametry to: procesor i5 o taktowaniu 1.6 GHz, pamięć RAM 4 GB, dysk twardy SSD 128 GB. Sprawdziłam średni czas odpowiedzi serwera na zapytanie do API tramwajów warszawskich - od momentu wysłania zapytania do otrzymania odpowiedzi. Otrzymałam następujące wyniki dla 100 wywołań API: średnia arytmetyczna 0.195s, odchylenie standardowe 0.145s, minimum 0.152s, maksimum 1.117s. Przeprowadzenie analogicznych testów dla API Twittera jest niezasadne ze względu na sporadyczność jego wykorzystania.

4.2 Wnioski

Udało mi się zrealizować wszystkie założenia i cele stawiane w ramach pracy. Stworzyłam aplikację sieciową powiadamiającą o wyjątkowych opóźnieniach tramwajów w Warszawie. Poprawność jej działania jest w zupełności niezależna od aktywności użytkowników, a dedykowane obszarom konta utworzone na Twitterze służą jako kanały dystrybucji informacji o sytuacjach wyjątkowych.

Podczas realizacji pracy napotkałam więcej problemów niż oczekiwałam na początku działań nad

realizacją pracy. Jednakże proces ich rozwiązywania był bardzo kształcący i korzystnie wpłynął na jakość rozwiązania.

Podstawą aplikacji jest analiza ruchu tramwajów w czasie rzeczywistym, dlatego też jest ona uzależniona od źródła danych, czyli platformy "Dane po warszawsku". W celu zapewnienia ciągłości w publikowaniu powiadomień, konieczna jest stała obserwacja zmian wprowadzanych w udostępnianym przez nią API i możliwie szybka reakcja, aby dostosować odpowiednie moduły do nowych warunków. Z tego względu konserwacja i utrzymanie aplikacji to kolejny etap projektu, który nie powinien być zaniedbywany w trosce o komfort użytkowników. Należy również pamiętać o uwzględnieniu aspektów pracy związanych z wykorzystaniem portali społecznościowych oraz komunikacją z nimi. Biorąc pod uwagę zmiany jakim coraz częściej ulegają, najbezpieczniejszym rozwiązaniem jest poleganie na co najmniej dwóch takich portalach. Zmniejsza to szanse na konieczność zawieszenia usługi na czas adaptacji.

Wnioskiem płynącym z przeprowadzonej przeze mnie analizy przedstawionych histogramów jest to, że udało mi się poprawnie dobrać kryterium kwalifikujące tramwaje jako opóźnione. Zostało ono wyznaczone metodą inżynierską, która bazuje na subiektywnej ocenie pokonanych dystansów w dobranym oknie czasowym opartej na szczegółowej analizie ruchu tramwajów w różnych sytuacjach. Dlatego też powiadomienia wysyłane przez aplikację są zgodne z warunkami panującymi na przejazdach.

4.3 Perspektywy rozwoju

Aplikacja powstała w wyniku pracy inżynierskiej ma bardzo duży potencjał. Wydaje mi się, że najbardziej interesujące i twórcze byłoby rozwinięcie jej do poziomu `https://jakdojade.pl`, które umożliwia wyznaczenie optymalnej trasy z wykorzystaniem różnych środków transportu publicznego. Jednakże nie uwzględnia ono warunków panujących na drodze, dlatego na tym polu moglibyśmy się wykazać. Docelowo chcielibyśmy, aby aplikacja wyznaczyła optymalny czas podróży tramwajami z podanych przez użytkownika punktów.

Jednakże mając w pełni działający podgląd na mapie, naturalna jest próba rozwinięcia również tej części projektu. Moja wizja jest następująca. Pobieramy lokalizacje przystanków tramwajowych, a także rozkłady jazdy, do których mamy dostęp poprzez "Dane po warszawsku". Tworzymy kolejną warstwę mapy na której zaznaczamy przystanki tramwajowe. Modyfikujemy też znaczniki tramwajów, ponieważ chcemy, aby były one "sprytne", czyli można było w nie kliknąć i pokazało nam się coś na mapie. W tym celu będziemy musieli dla każdej linii zdefiniować listę przystanków i wgrać dla niej rozkłady jazdy. Chcemy bowiem, aby po kliknięciu w ikonkę tramwaju przystanki, przez które przebiega jego linia, zmieniły swój kolor lub połączyły się łamaną, a także aby wyświetliło się okienko z czasem dojazdu do kolejnych przystanków przez dany tramwaj - oczywiście z uwzględnieniem opóźnień.

Tym samym przechodzimy do najważniejszej części. Pierwsza wersja obejmowałaby punkt startowy i docelowy w formie przystanków wybranych z listy, a druga wyszukanie najbliższego przystanku do podanego przez użytkownika adresu. Pracę nad pierwszą z nich podzieliłabym na dwa etapy - prototyp algorytmu wyznaczającego przewidywany czas dojazdu do przystanków z rozkładu począwszy od podanego przez użytkownika. Drugi będzie rozszerzeniem obejmującym złożenie trasy z kilku

odcinków. Jeżeli odcinki będą miały część wspólną, problemem będzie tylko znalezienie odpowiedniego tramwaju do przesiadki. Natomiast w przeciwnym przypadku konieczne będzie znalezienie najbliższego przystanku do końcowego danego odcinka, z którego odjeżdżają tramwaje w interesującym nas kierunku. Kluczowym elementem drugiej wersji rozwiązania jest algorytm wyszukiwania przystanku. Mając listy przystanków z ich lokalizacjami korzystamy z funkcjonalności biblioteki OpenLayers w następujący sposób. Otaczamy końcowy przystanek okręgiem o ustalonym promieniu, spośród pozostałych przystanków wybieramy te, które się w nim zawierają. Następnie filtrujemy je według przyjętych kryteriów i wyznaczamy dalszą część trasy, wykorzystując algorytm zaprojektowany w poprzednim etapie. Wtedy już będziemy mieli oczekiwany efekt - wyszukiwanie trasy z uwzględnieniem występujących na niej opóźnień.

Pewnym utrudnieniem jest brak numeru taboru w otrzymanych danych. Jednak już na tym etapie możemy naszkicować potencjalne rozwiązanie. Najprostszym pomysłem wydaje się być stworzenie heurystyki nadającej tramwajom identyfikatory. Moglibyśmy w niej wykorzystać fragment używanego w aktualnym projekcie sortowania wpisów z bazy, które daje nam uporządkowany zbiór "podzielony na sekcje" dotyczące konkretnych tramwajów.

BIBLIOGRAFIA

Bibliografia

[1] Serwis internetowy przeznaczony do wyszukiwania połączeń komunikacyjnych i planowania podróży komunikacją miejską w wybranych polskich miastach [dostęp 7 sierpnia 2018]

`https://jakdojade.pl`

[2] Platforma, na której znajdują się otwarte dane publiczne m.st. Warszawy [dostęp 7 sierpnia 2018]

`https://api.um.warszawa.pl/`

[3] Strona projektu "Dane po warszawsku"[dostęp 7 sierpnia 2018]

`http://www.danepowarszawsku.pl/`

[4] Dokumentacja wywołania API udostępnianego przez Tramwaje Warszawskie [dostęp 7 sierpnia 2018]

`https://api.um.warszawa.pl/files/7b867bc4-3584-443a-a028-c2606761fa01.pdf`

[5] Warszawski ninja [dostęp 7 sierpnia 2018]

`https://warszawskininja.pl/czesc/`

[6] Gdzie Ten Tramwaj? (Autobus też!) [dostęp 7 sierpnia 2018]

`https://play.google.com/store/apps/details?id=com.kksionek.gdzietentramwaj&hl=pl`

[7] System nawigacji satelitarnej [dostęp 7 sierpnia 2018]

`https://www.gps.gov/systems/gps/`

[8] System Nadzoru Ruchu Tramwajów [dostęp 7 sierpnia 2018]

`https://geoforum.pl/upload/files/site_catalog_text/0_NAWI_3_116_2005.pdf`

`http://www.infotron.com.pl/index.php/snrt-2000`

[9] Dokumentacja Django [dostęp 7 sierpnia 2018]

`https://docs.djangoproject.com/pl/2.0/intro/overview/`

[10] Schemat porównujący żądania AJAX oraz HTTP [dostęp 7 sierpnia 2018]

`http://www.dotnetcurry.com/jquery/1093/jquery-ajax-basics`

[11] Dokumentacja Git [dostęp 7 sierpnia 2018]

`https://git-scm.com/about`

[12] OpenStreetMap [dostęp 7 sierpnia 2018]

`https://www.openstreetmap.org/about`

[13] Dokumentacja biblioteki Openlayers [dostęp 7 sierpnia 2018]

`https://openlayers.org/en/latest/doc/`

[14] Dokumentacja biblioteki Tweepy [dostęp 7 sierpnia 2018]
http://docs.tweepy.org/en/v3.5.0/

[15] Integracja Twittera z Facebookiem [dostęp 7 sierpnia 2018]
https://help.twitter.com/en/managing-your-account/link-twitter-to-facebook

[16] Zmiany w polityce Facebooka [dostęp 7 sierpnia 2018]
https://developers.facebook.com/blog/post/2018/04/24/new-facebook-platform-product-cha

[17] Własne komendy do Django [dostęp 7 sierpnia 2018]
https://docs.djangoproject.com/pl/2.0/howto/custom-management-commands/

SPIS RYSUNKÓW

Spis rysunków

Spis tablic