

Uniwersytet Warszawski
Wyddział Matematyki, Informatyki i Mechaniki

**Magdalena
Grabowska**

Nr albumu:
372701

**Michał
Kukuła**

Nr albumu:
371127

**Klaudia
Laks**

Nr albumu:
371151

**Przemysław
Perkowski**

Nr albumu:
371308

Miejsca Obsługi Podróżnych

Praca licencjacka
na kierunku **INFORMATYKA**

Praca wykonana pod kierunkiem
dr. hab. Aleksego Schuberta

Czerwiec 2018

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data

Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data

Podpisy autorów pracy

Streszczenie

W pracy opisano implementację systemu dotyczącego Miejsc Obsługi Podróżnych przy autostradach i drogach ekspresowych w Polsce. Podstawowe składowe tego systemu to aplikacje *Mopnik* i *Mopsim* połączone wspólnym interfejsem graficznym. *Mopsim* służy do przeprowadzania symulacji ruchu na drogach oraz predykcji zajętości miejsc parkingowych na Miejscach Obsługi Podróżnych. Może być używany z wiersza polecen oraz z poziomu *Mopnika*. Aplikacja okienkowa *Mopnik* wyświetla na mapie wyniki tych symulacji oraz umożliwia przewidywanie liczby potrzebnych miejsc parkingowych na podstawie zdefiniowanych metodyk. Pozostałe dwie części projektu to aplikacja mobilna oraz strona internetowa *Mopsik* przeznaczone dla kierowców poruszających się po drogach. Zawierają informacje o MOP-ach oraz dostępnych na nich usługach. Informują także o aktualnej zajętości miejsc parkingowych.

Słowa kluczowe

symulacja, wizualizacja, Miejsce Obsługi Podróżnych, miejsce parkingowe, aplikacja okienkowa, aplikacja mobilna, strona internetowa, zajętość miejsc parkingowych, natężenie ruchu, Java, React Native, C#, OpenStreetMap

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

D. Software

Tytuł pracy w języku angielskim

Parking places at rest areas on highways and expressways in Poland

Spis treści

1. Słowniczek	5
2. Wprowadzenie	7
2.1. Rozwój sieci drogowej w Polsce	7
2.2. Miejsca Obsługi Podróżnych	8
2.3. Problemy związane z budową MOP-ów	9
2.4. Symulacja ruchu drogowego w Polsce	10
2.5. Przewidywanie liczby potrzebnych miejsc parkingowych	10
2.6. Zajętość MOP-ów i dostępne usługi	11
2.7. Nasze zadanie	11
3. Mopnik	13
3.1. Przypadki użycia	13
3.2. Architektura	13
3.2.1. Wykorzystywanie biblioteki	13
3.2.2. Dane wejściowe	14
3.2.3. Domyślne dane wejściowe	14
3.2.4. Wyświetlanie mapy	14
3.2.5. Integracja z programem Mopsim	14
3.2.6. Pikietaż a współrzędne geograficzne	14
3.3. Metodyki	15
3.3.1. Wyświetlanie wyników metodyk	15
3.3.2. Dodatkowe metodyki	15
3.4. Zrzuty ekranu	15
4. Mopsim	17
4.1. Przypadki użycia	17
4.2. Architektura	17
4.2.1. Modele mikro-, makro-, mezoskopowe	17
4.2.2. Model obliczeniowy oparty na agentach	18
4.2.3. System wieloagentowy	18
4.2.4. MATSim – symulator ruchu drogowego	19
4.2.5. MATSim – pętla programu	19
4.2.6. Dlaczego MATSim?	20
4.2.7. Mopsim – technologie	20
4.3. Opis działania	21
4.3.1. Generowanie planów podróży	21
4.3.2. Generowanie planu MOP-ów	22
4.3.3. Opis działania agenta w czasie symulacji	22

4.3.4. Mopsim - pliki wyjściowe	23
4.4. Instrukcja użytkownika	25
4.4.1. Konfiguracja	25
4.4.2. Przeprowadzanie symulacji	26
4.4.3. Generowanie siatki drogowej	27
4.4.4. Strategie	27
4.4.5. Mopsim - dodanie własnych strategii	28
4.5. Opis implementacji	28
4.5.1. Podział plików	28
4.5.2. Opis poszczególnych klas	28
5. Mopsik – Strona serwerowa - API	31
5.1. Przypadki użycia API	31
5.2. Architektura	31
6. Mopsik – Aplikacja mobilna	33
6.1. Przypadki użycia	33
6.2. Architektura i opis implementacji	33
6.2.1. Technologia	33
6.2.2. Zakładki	34
6.2.3. Konfiguracja aplikacji i ustawienia	34
6.2.4. Google Maps	36
6.2.5. Zapytanie do API	37
6.2.6. Użyte biblioteki	37
6.2.7. Nawigacja	38
6.2.8. Pozostałe komponenty	39
6.2.9. Pliki konfiguracyjne i funkcje	40
6.2.10. Zapisywanie danych	41
7. Mopsik – Strona internetowa	43
7.1. Przypadki użycia	43
7.2. Architektura i opis implementacji	43
7.2.1. Technologia	43
7.2.2. Zakładki	43
7.2.3. Zapytania do API i zapisywanie danych	46
7.2.4. Modele i modele widoków	47
7.2.5. Kontrolery i widoki	48
7.2.6. Użyte biblioteki	48
8. Organizacja pracy	49
9. Podział zadań	51
10. Napotkane problemy i niezrealizowane pomysły	53
10.1. Napotkane problemy	53
10.2. Niezrealizowane pomysły	54
11. Zawartość płyty	57
Bibliografia	59

Rozdział 1

Słowniczek

API Interfejs programowania aplikacji (ang. Application Programming Interface).

GDDKiA Generalna Dyrekcja Dróg Krajowych i Autostrad (nasz klient).

GUI Graficzny interfejs użytkownika.

MOP Miejsce Obsługi Podróżnych.

OSM OpenStreetMap[10].

RID Rozwój Innowacji Drogowych.

RN React Native[8].

SDR Średniodobowe natężenie ruchu.

ZPP Zespołowy Projekt Programistyczny.

Rozdział 2

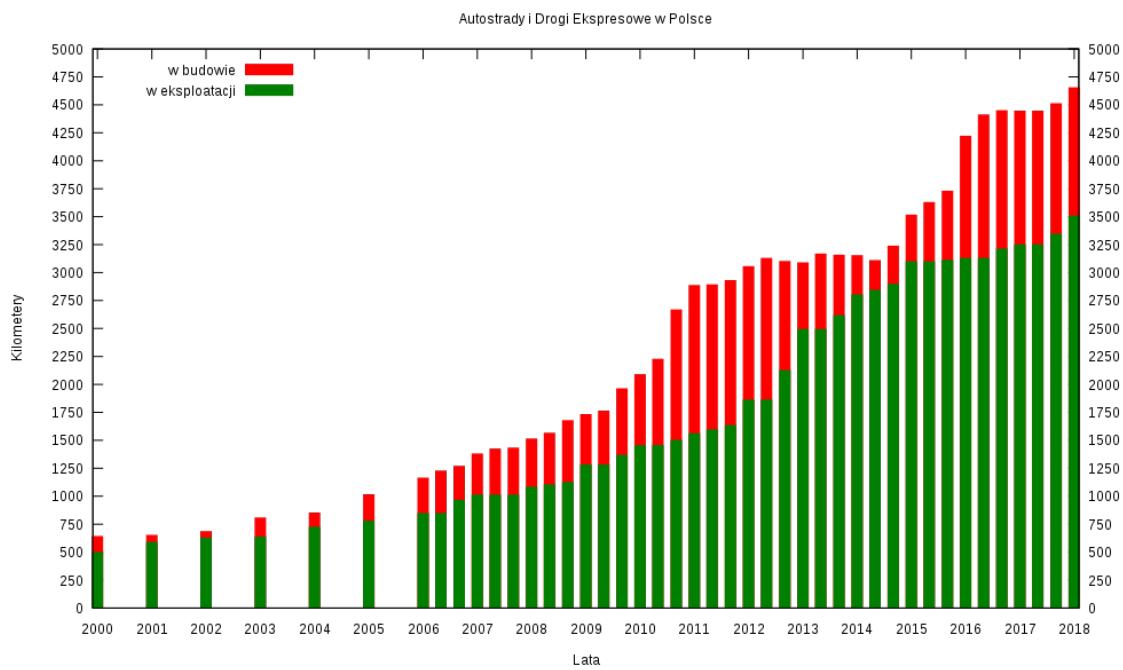
Wprowadzenie

2.1. Rozwój sieci drogowej w Polsce

Jeszcze pod koniec XX w. w Polsce znajdowało się zaledwie ok. 500km autostrad i dróg ekspresowych. Pomimo licznych planów rozwojowych, które przyjmowane były w okresie PRL i pierwszych latach III RP, większość dróg budowana była z dużymi opóźnieniami, a niektórych odcinków nie realizowano wcale.

Dopiero przyjęty pod koniec lat 90. program budowy sieci autostrad i dróg ekspresowych w Polsce, mimo wielu modyfikacji jest na bieżąco realizowany. Jako przyczyny[1] przyspieszenia budowy dróg szybkiego ruchu w Polsce wymienia się między innymi: duże zainteresowanie problemem, wzrost liczby pojazdów, dotacje z europejskiego Funduszu Spójności, a także dotacje Unii Europejskiej. Dodatkowym impulsem mobilizującym instytucje rządowe do przyspieszenia prac w tym kierunku stało się przyznanie organizacji Euro 2012 Polsce i Ukrainie.

Rysunek 2.1: Rozwój sieci dróg ekspresowych i autostrad w Polsce na początku XX w.



Ostatnie lata to okres szczególnie intensywnego rozwoju sieci drogowej w Polsce. Budowa infrastruktury wiąże się również z potrzebą zapewnienia podróźnym bezpieczeństwa i komfortu.

Rysunek 2.2: Sieć autostrad i dróg ekspresowych w Polsce (styczeń 2018r.). Na zielono – odcinki zrealizowane, na czerwono – odcinki w budowie, na szaro – odcinki planowane.



2.2. Miejsca Obsługi Podróżnych

Miejsce Obsługi Podróżnych (**MOP**) to teren wydzielony w pasie drogowym (w bliskim sąsiedztwie drogi), wyposażony w parking oraz w infrastrukturę zapewniającą komfort i odpoczynek podróżnym[2]. MOP-y powstają tylko przy autostradach i drogach ekspresowych. MOP-y w Polsce dzielimy na trzy kategorie:

1. **MOP kategorii I** – o funkcji wypoczynkowej, wyposażony w stanowiska postojowe (parking), jezdnie manewrowe, urządzenia wypoczynkowe, sanitarne i oświetlenie; dopuszcza się wyposażenie w obiekty małej gastronomii.

2. **MOP kategorii II** – o funkcji wypoczynkowo-usługowej, wyposażony w obiekty, o których mowa w punkcie 1. oraz w stację paliw, stanowiska obsługi pojazdów, obiekty gastronomiczno-handlowe, punkty informacji turystycznej.
3. **MOP kategorii III** – o funkcji wypoczynkowej i usługowej, wyposażony w obiekty, o których mowa w punkcie 2., obiekty noclegowe oraz inne obiekty handlowo-usługowe w zależności od potrzeb.



Rysunek 2.3: Aktualne (styczeń 2018r.) pozycje MOP-ów w Polsce (na niebiesko). MOP-y planowane (na czerwono).

2.3. Problemy związane z budową MOP-ów

Intensywny rozwój sieci drogowej, a co za tym idzie również szybki wzrost liczby MOP-ów w Polsce, stwarza szereg problemów z nimi związanych:

1. **Problemy administracyjno-prawne** – GDDKiA systematycznie prowadzi kolejne przetargi na dzierżawę MOP zlokalizowanych zarówno przy autostradach jak i drogach

ekspresowych. Dzierżawa nieruchomości MOP generuje przychody, które systematycznie zasilają budżet Krajowego Funduszu Drogowego. Nieatrakcyjne warunki umowy lub lokalizacja punktu mogą zniechęcać potencjalnych najemców.

2. **Lokalizacja MOP-ów** – efektywne rozmieszczenie MOP-ów powinno uwzględniać takie parametry jak: odległość od najbliższych MOP-ów, natężenie odcinka drogi, odległość od węzłów komunikacyjnych.
3. **Liczba miejsc parkingowych i ich układ** – MOP-y powinny dysponować taką liczbą miejsc parkingowych, by zapewnić możliwość odpoczynku podróżującym także w warunkach wzmożonego ruchu. Rozmieszczenie miejsc parkingowych powinno zapewnić kierowcom komfort podczas poruszania się pojazdem na terenie punktu.

2.4. Symulacja ruchu drogowego w Polsce

Efektywne rozmieszczenie coraz większej liczby MOP-ów jest jednym z głównych wyzwań stojących przed planistami dróg. Powinno ono uwzględniać wszystkie kwestie poruszane w poprzednim rozdziale. Jednak rozwój sieci drogowej w Polsce na niespotykaną wcześniej skalę sprawia, że zadanie to jest coraz trudniejsze. Może się okazać, że dane dotyczące natężenia ruchu na danym odcinku drogi, odległości od najbliższych MOP-ów czy węzłów komunikacyjnych są niewystarczające i trudno na ich podstawie określić wykorzystanie MOP-a w okresach wzmożonego ruchu sezonowego lub w przeszłości, wraz z dalszym rozwojem infrastruktury drogowej. Przydatnymi danymi, wykorzystywanyymi w modelowaniu ruchu drogowego, są macierze podróży, określające liczbę pojazdów poruszających się pomiędzy danymi parami punktów w zadanym okresie. Planista dróg, wyposażony w macierze podróży, chciałby na ich podstawie wiedzieć:

1. Jakie jest natężenie ruchu na poszczególnych odcinkach drogi?
2. Jak dodanie lub usunięcie (np. w wyniku tymczasowego zatrzymania ruchu) odcinka drogi wpłynie na to natężenie?
3. Jaka część podróżnych, jadąca danym odcinkiem drogi, chciałaby skorzystać z MOP-a?

Pomocny w przeprowadzeniu tej analizy może okazać się **Mopsim** – program komputerowy symulujący ruch pojazdów na sieci dróg krajowych i autostrad w Polsce.

2.5. Przewidywanie liczby potrzebnych miejsc parkingowych

Mając dane dotyczące natężenia ruchu na danym odcinku drogi (pochodzące z przeprowadzonych pomiarów lub symulacji) można spróbować określić sumaryczną liczbę miejsc parkingowych potrzebnych na tym odcinku. W tym celu powszechnie wykorzystuje się różne metodyki, będące zwykle prostymi wzorami matematycznymi.

Przykład metodyki:

$$P_T = N_T \times SDR_T \times WS_T$$

gdzie:

- T – typ pojazdu
- P_T – liczba potrzebnych miejsc parkingowych na 15km drogi

- N_T – wskaźnik przeliczeniowy uwzględniający np. typ MOP-u
- SDR_T – średni dobowy ruch w roku w analizowanym kierunku
- WS_T – wskaźnik zmienności sezonowej

Program **Mopnik** będzie miał za zadanie umożliwienie korzystania z takich metodyk – ręcznego wprowadzania niektórych parametrów, wczytywania potrzebnych danych (np. średnio-dobowego natężenia ruchu) oraz obliczania liczby potrzebnych miejsc parkingowych.

Program jest zintegrowany z systemem **Mopsim** dzięki czemu możliwe będzie zastosowanie wyżej wymienionych metodyk do danych pochodzących z symulacji, jeśli okaże się, że wyniki rzeczywistych pomiarów są niepełne.

2.6. Zajętość MOP-ów i dostępne usługi

Jednym ze wspomnianych wcześniej problemów związanych z budową MOP-ów jest dobór odpowiedniej liczby miejsc parkingowych. Dodatkowo, miejsca te należy odpowiednio podzielić pomiędzy różne typy pojazdów: samochody osobowe, samochody ciężarowe, autobusy, pojazdy przewożące substancje niebezpieczne itd. Podróżni mają też różne potrzeby związane z postojem – od szybkiego zatankowania samochodu, przez obiad z rodziną na świeżym powietrzu, do spędzienia nocy w kabinie.

Aktualnie informację o znajdujących się na MOP-ie usługach można uzyskać ze znaków informacyjnych umieszczonych kilka kilometrów przed zjazdem. Jednak takie oznaczenia nie odpowiadają na kilka ważnych pytań, które możemy mieć na dowolnym etapie podróży:

1. Za ile kilometrów znajduje się najbliższa stacja benzynowa?
2. Czy starczy mi paliwa do następnej, ponieważ tutaj jest duża kolejka?
3. Czy na tym MOP-ie jest monitoring?
4. Czy na tym MOP-ie są wolne miejsca parkingowe? (ten problem jest ważniejszy z punktu widzenia kierowców pojazdów wielkogabarytowych)
5. Jaka będzie zajętość miejsc parkingowych za godzinę?

Odpowiedzią na te pytania będzie **Mopsik** (MOP – System Informowania Kierowców) – aplikacja mobilna i strona internetowa. Programy te pobierają dane z API wystawianego przez Mopsik-Serwer.

2.7. Nasze zadanie

W ramach zajęć ZPP postanowiliśmy podjąć się implementacji prototypów wyżej opisanych programów. Badania są finansowane ze środków projektu pt. „Miejsca parkingowe na MOP” finansowanego przez NCBiR/GDDKiA w ramach wspólnego przedsięwzięcia „RID”, umowa DZP/RID-I-44/8/NCBR/2016.

Rozdział 3

Mopnik

3.1. Przypadki użycia

Inżynier GDDKiA może:

1. Wprowadzać dane dotyczące SDR, sieć drogową oraz położenie MOP-ów z pliku.
2. Wyświetlić wprowadzone dane na mapie.
3. Wybrać metodykę, ustawić jej parametry i na jej podstawie wyznaczyć liczbę potrzebnych miejsc parkingowych na odcinku drogi.
4. Wyświetlić wyniki na mapie.
5. Edytować sieć drogową.
6. Uruchomić symulacje przeprowadzane za pomocą systemu **Mopsim** dotyczące:
 - (a) Predykcji SDR.
 - (b) Predykcji zajętości MOP-ów.

3.2. Architektura

Mopnik jest napisany w języku Java z użyciem narzędzi Apache Maven automatyzującego budowę programu.

3.2.1. Wykorzystywanie biblioteki

Implementacja opiera się na wykorzystaniu następujących bibliotek dla języka Java:

- Swing – biblioteka graficzna służąca do stworzenia GUI aplikacji.
- JXMapView2 – biblioteka zapewniająca swingowy JPanel renderujący kafelki mapy (<https://github.com/msteiger/jxmapviewer2>).
- osm-parser – biblioteka parsująca pliki .osm do obiektów w Javie (<https://github.com/imintel/osm-parser>).
- opencsv – biblioteka do parsowania plików CSV (<http://opencsv.sourceforge.net/>)
- org.json – biblioteka do parsowania plików JSON

Oprócz zewnętrznych bibliotek, program korzysta też z będącej częścią tego projektu biblioteki **Mopsim**.

3.2.2. Dane wejściowe

Użytkownik programu ma możliwość wprowadzenia danych dotyczących SDR z plików w formacie CSV oraz położenia MOP-ów z pliku w formacie xlsx. Ponadto użytkownik ma możliwość wczytania sieci drogowej w formacie OSM. Parsowaniem obu rodzajów plików zajmują się odpowiednie biblioteki wymienione w poprzedniej sekcji.

3.2.3. Domyślne dane wejściowe

Po uruchomieniu programu ładowane są domyślne dane wejściowe. Układ MOP-ów pochodzi z serwera (Mopsik-API). Dane dotyczące SDR oraz mapa w formacie OSM są ładowane z plików dołączonych do programu.

3.2.4. Wyświetlanie mapy

Mapa Polski, która jest wyświetlana po otwarciu programu, pochodzi z serwisu OpenStreetMap[10]. Kafelki (w formacie graficznym) są ściągane z serwera¹ według potrzeby, czyli przy każdym przesunięciu czy przybliżeniu mapy. Takie rozwiązanie sprawia, że użytkownik musi być przez cały czas korzystania z programu podłączony do internetu. Jest możliwość stworzenia lokalnego serwera, hostującego kafelki², a następnie zmiany w kodzie adresu serwera na *localhost*. Ze względu na powszechny dostęp do internetu zrezygnowaliśmy z tego rozwiązania, ale warto wiedzieć, że ono istnieje. Wczytywanie plików OSM daje następujące możliwości:

1. Nanoszenie własnych oznaczeń na mapę – na przykład kolorowanie konkretnych dróg.
2. Modyfikowanie sieci drogowej. W tym wypadku zmiany nie zostaną wyświetcone tak jak istniejące drogi, ponieważ kafelki pochodzą z innego źródła. Wymagałoby to renderowania map w czasie działania programu, co spowolniłoby jego działanie. Zamiast tego, nowe drogi wizualizowane są za pomocą prostych odcinków łączących punkty wyznaczone przez użytkownika. Możliwe jest jednak dodawanie MOP-ów do tych odcinków oraz obliczanie wyników metodyk.

3.2.5. Integracja z programem Mopsim

Mopsim jest dołączony do Mopnika jako jego biblioteka. Z poziomu GUI można wybrać pliki wejściowe dla symulacji, parametry (np. liczba samochodów biorących w niej udział, liczba wątków, na których będzie się ona odbywać) oraz uruchomić symulację. Dodatkowo, po przeprowadzonej symulacji w podględzie MOP-a pojawia się link do katalogu ze szczegółowymi wynikami dotyczącymi zjazdów na niego. W symulacji mogą również brać udział dodane przez użytkownika drogi i MOP-y.

3.2.6. Pikietaż a współrzędne geograficzne

Problem Dane dotyczące SDR oraz układów MOP-ów są podane za pomocą nazwy drogi oraz pikietażu. Dla obliczania wyników metodyk jest to pomocne – interesują nas bowiem wyniki dla konkretnych odcinków konkretnych dróg. Pojawił się jednak problem wizualizacji

¹tile.openstreetmap.org

²<https://switch2osm.org/manually-building-a-tile-server-16-04-2-lts/>

tych danych – jak ustalić jakie współrzędne geograficzne ma punkt na drodze określony za pomocą nazwy drogi oraz pikietażu.

Rozwiążanie Niektóre węzły w plikach OSM opisane są tagiem *distance* oznaczającym właśnie pikietaż oraz współrzędnymi geograficznymi. Dzięki temu możliwe jest naniesienie tych węzłów na mapę.

3.3. Metodyki

W ostatecznej wersji programu zaimplementowana jest tylko jedna metodyka (nazwana metodyką domyślną), pochodząca z pracy dr. Malwiny Spławińskiej[11]. W pracy tej zaproponowane są konkretne parametry liczbowe dla tej metodyki, które w Mopniku są ustawione jako parametry domyślne, ale mogą być zmienione z poziomu użytkownika.

3.3.1. Wyświetlanie wyników metodyk

Liczby miejsc parkingowych są obliczane dla konkretnych odcinków dróg. Uwzględniana przy tym jest ich długość, ponieważ metodyka zwraca liczbę potrzebnych miejsc parkingowych na odcinku 15km. Dla konkretnego odcinka dysponujemy więc następującymi informacjami:

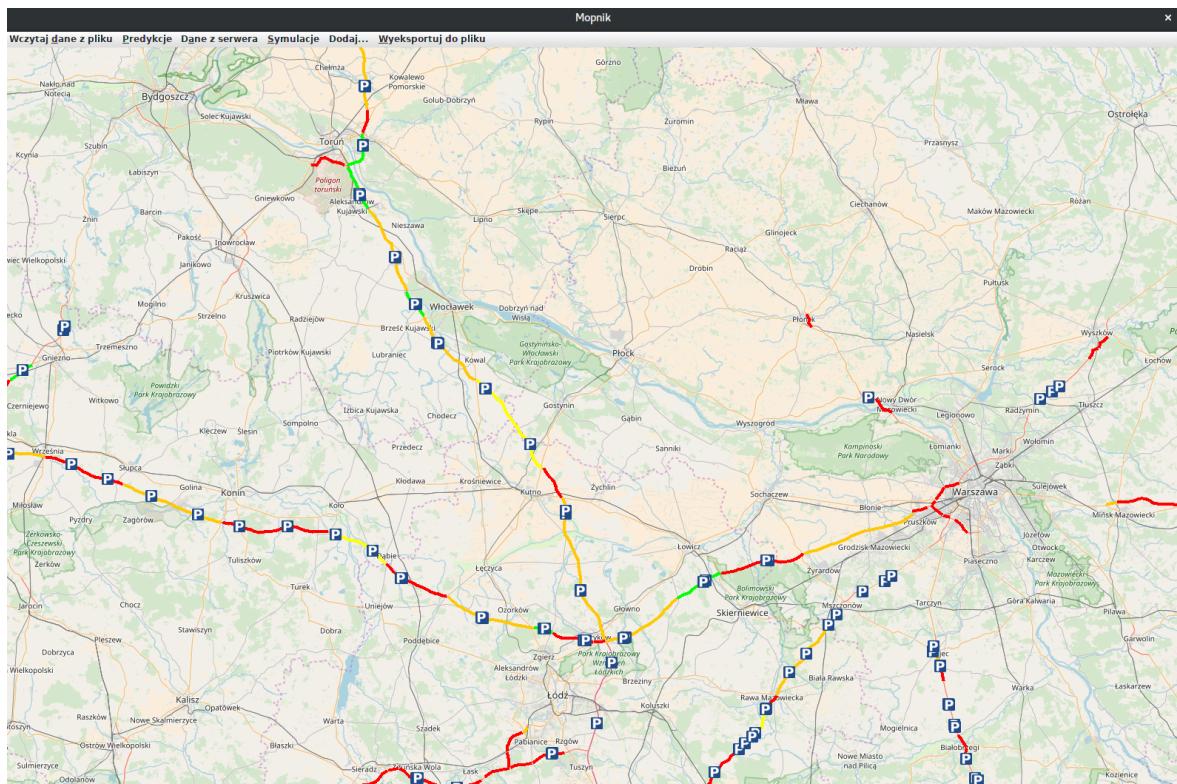
- SDR
- Liczba potrzebnych miejsc parkingowych
- Liczba istniejących miejsc parkingowych (suma liczb miejsc parkingowych na MOP-ach znajdujących się na tym odcinku).

Informacje te pozwalają na ustalenie stref priorytetowych: odcinków dróg, na których liczba miejsc parkingowych nie jest wystarczająca. Realizowane jest to przy uruchomieniu programu na podstawie domyślnych danych oraz domyślnych parametrów metodyk. Odcinki, na których brakuje miejsc parkingowych są zaznaczone na mapie drogowej kolorem czerwonym lub pomarańczowym (w zależności liczby brakujących miejsc). Przy wczytaniu nowych danych, dodaniu MOP-a, lub zmianie parametrów metodyki następuje przeładowanie mapy.

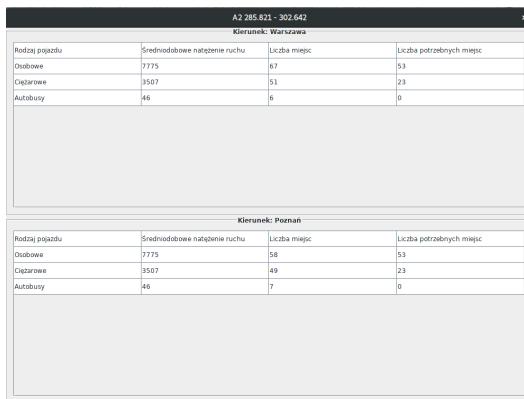
3.3.2. Dodatkowe metodyki

W ramach projektu rozważane było użycie wielu metodyk. Po konsultacjach z klientem ustalono, że należy wybrać jedną z nich. Istniejąca implementacja pozwala jednak na dodanie nowych metodyk z poziomu programisty.

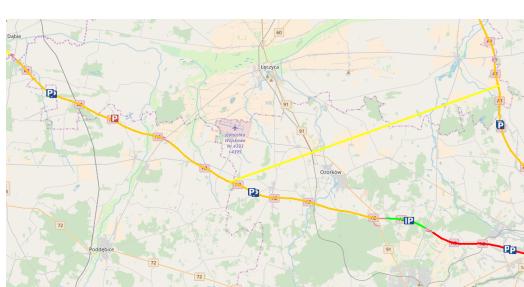
3.4. Zrzuty ekranu



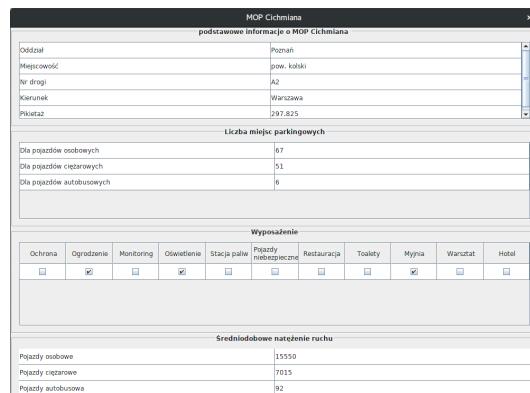
Rysunek 3.1: Główny widok programu z zaznaczonymi strefami priorytetowymi



Rysunek 3.2: Podgląd odcinka drogi



Rysunek 3.4: Widok główny po dodaniu MOP-a i drogi



Rysunek 3.3: Podgląd MOP-a

Ustal dane wejściowe symulacji

Plik z siecią drogową	poland_network.xml	Wybierz
Układ MOPów	src/main/resources/mop_data/mop_data.csv	Wybierz
Macierz samochodów osobowych	src/main/resources/travel_matrices/car_matrix.csv	Wybierz
Macierz samochodów ciężarowych	src/main/resources/travel_matrices/truck_matrix.csv	Wybierz
Macierz autobusów	src/main/resources/travel_matrices/bus_matrix.csv	Wybierz

Stale użyte w symulacji

Liczba pojazdów osobowych	
Liczba pojazdów ciężarowych	
Liczba pojazdów autobusowych	
Id symulacji	sim_20180615-123047
Liczba wątków	3

Przeprawdź symulację

Rysunek 3.5: Uruchamianie symulacji w programie Mopsim

Rozdział 4

Mopsim

4.1. Przypadki użycia

Inżynier GDDKiA może:

1. Wprowadzać dane dotyczące sieci drogowej z pliku osm.
2. Wprowadzać dane dotyczące MOP-ów i macierze podróży z plików CSV.
3. Edytować sieć drogową i dodawać lub usuwać MOP-y.
4. Przeprowadzać symulację ruchu drogowego.
5. Wyznaczać przewidywane natężenie ruchu na danym odcinku drogi.
6. Wyznaczać przewidywaną zajetość MOP-ów i liczbę potrzebnych miejsc parkingowych.
7. Generować raporty.

4.2. Architektura

4.2.1. Modele mikro-, makro-, mezoskopowe

Modelem obliczeniowym w informatyce nazywamy model matematyczny, wykorzystujący zasoby komputerowe do zbadania zachowania złożonego systemu za pomocą symulacji[3]. Modele obliczeniowe są obecnie wykorzystywane w bardzo wielu dziedzinach, m.in. do prognozy pogody, badania zmian klimatycznych, symulacji ruchu planet, a także w biologii czy medycynie. Jednym ze sposobów klasyfikacji modeli jest podział na modele mikro-, makro- i mezoskopowe[4]:

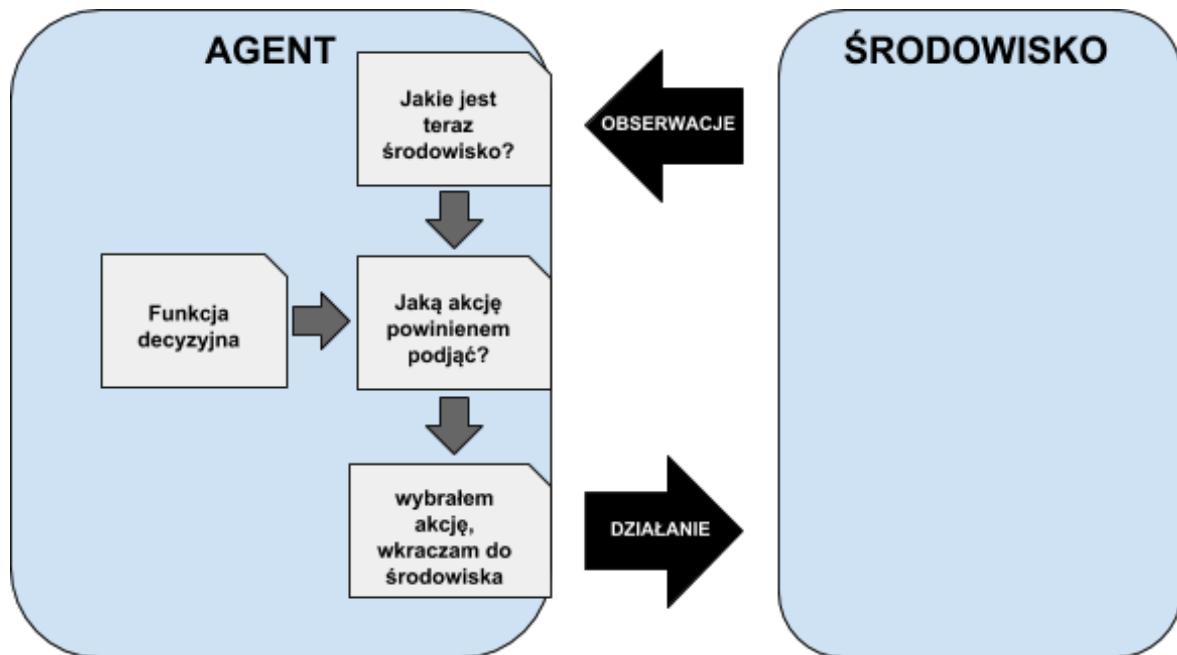
- w modelu mikroskopowym każda jednostka jest opisywana przez jej charakterystyczne atrybuty i zachowania
- w modelu makroskopowym analizuje się zagregowane cechy grupy jednostek i ich oddziaływanie na cały system
- w modelu mezoskopowym bada się jednostki połączone w niewielkie grupy, których elementy są traktowane jako jednolite.

4.2.2. Model obliczeniowy oparty na agentach

Jednym z intensywnie stosowanych i rozwijanych modeli obliczeniowych jest model oparty na agentach (ang. *agent-based model*). Polega on na symulowaniu zachowań i oddziaływania między sobą zbioru autonomicznych agentów, w celu badania ich wpływu na cały system[5]. Zwykle agentom przyporządkowuje się uproszczone zachowania, a następnie umieszcza w pewnym miejscu i czasie. Następnie symuluje się podejmowane przez nich działania i oddziaływanie między sobą w celu odtworzenia lub próby wyjaśnienia bardziej złożonych zjawisk. Modele obliczeniowe oparte na agentach stosuje się m.in. w biologii, ekologii czy naukach społecznych.

4.2.3. System wieloagentowy

Nieco innym, choć pokrewnym pojęciem jest system wieloagentowy (ang. *multi-agent system*). Jest to system komputerowy, złożony z wielu oddziałujących między sobą we wspólnym środowisku inteligentnych agentów. W odróżnieniu od modelu opartego na agentach, którego celem jest badanie złożonych procesów na podstawie zachowań poszczególnych jednostek, system wieloagentowy jest koncepcją programistyczną, w której agenci są wykorzystywani do rozwiązywania określonych problemów praktycznych lub inżynierijnych. Systemy wieloagentowe często stosowane są w sytuacjach, gdy trzeba rozwiązać problemy o charakterze rozproszonym lub złożonych obliczeniowo, np. wyszukiwanie informacji w sieci, zarządzanie sieciami telekomunikacyjnymi, symulacja rynku, wspomaganie zarządzania w przedsiębiorstwie i kontrola ruchu lotniczego[6]. Poniższy schemat pokazuje ogólną zasadę działania systemu wieloagentowego. Każdy z agentów działa w środowisku, oddziałując na innych agentów i przeprowadzając obserwacje. Na ich podstawie, agent podejmuje decyzję o ewentualnej modyfikacji planu działania. Następnie agent ponownie wkracza do środowiska.



Rysunek 4.1: Schemat działania systemu wieloagentowego

4.2.4. MATSim – symulator ruchu drogowego

MATSim (*Multi-Agent Transport Simulation*) to open-sourcowy, rozszerzalny framework napisany w Javie służący do przeprowadzania symulacji ruchu drogowego. Jego głównymi założeniami jest mikroskopowe modelowanie ruchu oraz oparte na agentach symulowanie dziennych planów poszczególnych jednostek. MATSim został zaprojektowany, by modelować aktywności podczas pojedynczego dnia. MATSim oparty jest o zasadę koewolucyjności. Każdy z agentów, reprezentujących pojedynczych uczestników ruchu drogowego, optymalizuje swój codzienny plan, rywalizując z innymi agentami w różnych aspektach korzystania z infrastruktury drogowej.

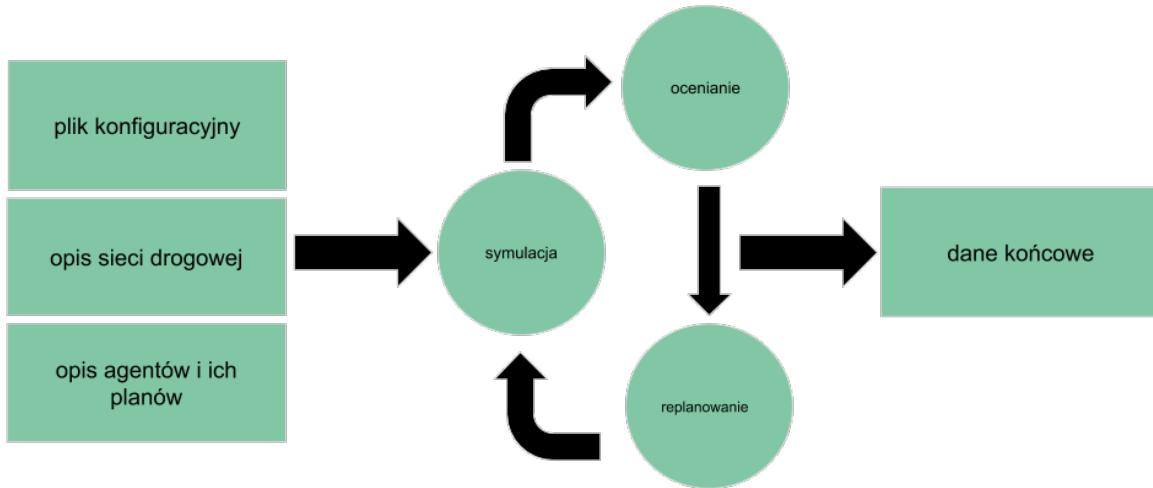
Projekt MATSim został zapoczątkowany w 2004r. na Politechnice Federalnej w Zurychu.^[7] Od tego czasu jest intensywnie rozwijany m.in. na tej uczelni, na Uniwersytecie Technicznym w Berlinie, a także przez dziesiątki innych osób reprezentujących różne gałęzie nauki. MATSim został wykorzystany do przeprowadzenia symulacji ruchu drogowego m.in. w Berlinie, Poznaniu, Seulu, Tel Avivie, Singapurze czy też dla całych Niemiec. Podejmowane były też próby wykorzystania go w celu modelowania ruchu lotniczego, symulowania ewakuacji czy badania emisji spalin.



Rysunek 4.2: Logo MATSim

4.2.5. MATSim – pętla programu

Działanie programu polega na przeprowadzeniu określonej liczby iteracji, reprezentowanych przez pętlę na poniższym schemacie. MATSim zaczyna swoje działanie, wczytując ustawienia konfiguracyjne dotyczące sieci drogowej, populacji agentów wraz z ich dziennymi planami oraz metadane o symulacji, takie jak wybrany układ współrzędnych, liczba wątków czy parametry poszczególnych modułów. Podczas iteracji, każdy z agentów optymalizuje swój początkowy plan. Każdy z nich posiada zdefiniowany zbiór planów, składających się z dziennego łańcucha aktywności i wyniku punktowego, który może być interpretowany jako użyteczność w pojęciu ekonometrycznym. Przed każdą iteracją, agenci wybierają plan ze swojego zbioru. Wybór ten zależy od wyników poszczególnych planów, które są obliczane po każdej pojedynczej symulacji. Pod koniec iteracji, część agentów (najczęściej ok. 10%) może zmodyfikować wybrany przez siebie plan i dodać go do swojego zbioru planów.



Rysunek 4.3: Schemat działania MATSim

4.2.6. Dlaczego MATSim?

Przeprowadzone próby pokazały, że MATSim może być dobrym narzędziem do symulowania ruchu na drogach w Polsce, w celu zbadania zajętości MOP-ów. Co istotne, wstępne symulacje pokazały, że już jedna iteracja działania głównej pętli programu jest wystarczająca – wynik jest bardzo zbliżony do optymalnego. Jest to zgodne z intuicją – w ruchu krajowym należy oczekiwać, że najkrótsza droga spośród dróg danego typu jest najbardziej optymalną, a two rzające się korki nie są tak częste i regularne, jak w intensywnym ruchu miejskim. Czynniki, które nas przekonały do wyboru frameworku MATSim to między innymi:

- licencja GNU GPLv2
- model ruchu umożliwiający symulację w skali sieci dróg krajowych i autostrad w Polsce
- istniejące podobne zastosowania i symulacje na sieciach dróg krajowych, m.in. w Niemczech i Szwajcarii
- możliwość łatwego rozszerzenia o dodatkowe funkcjonalności, wiele gotowych przykładów
- obszerna dokumentacja i dostęp do kodu źródłowego
- możliwość zrównoleglania

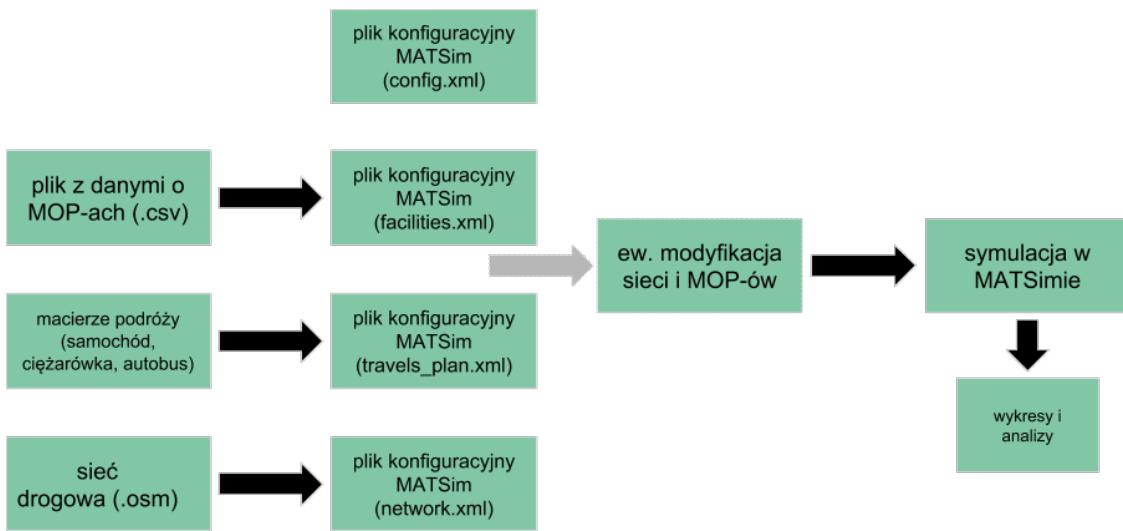
4.2.7. Mopsim – technologie

W programie Mopsim wykorzystano następujące technologie:

- MATSim – opisany wyżej
- Java – choć MATSim umożliwia też pisanie rozszerzeń w innym języku, zdecydowaliśmy się na wykorzystanie Javy jako języka programowania z uwagi na jej największą integrację z MATSimem. Dodatkowym atutem było nasze doświadczenie z pracy w tym języku, także w aspekcie programowania współbieżnego.

- OpenStreetMap[10] – Mopsim korzysta z map OpenStreetMap do opisu sieci drogowej. Jest to projekt społeczności internetowej mający na celu stworzenie darmowej, swobodnie dostępnej mapy całej kuli ziemskiej. Jej atutem są gotowe rozwiązania umożliwiające edycję oraz integrację z MATSimem, który posiada moduł umożliwiający konwersję na format używany do opisu dróg w symulacji.

4.3. Mopsim - opis działania



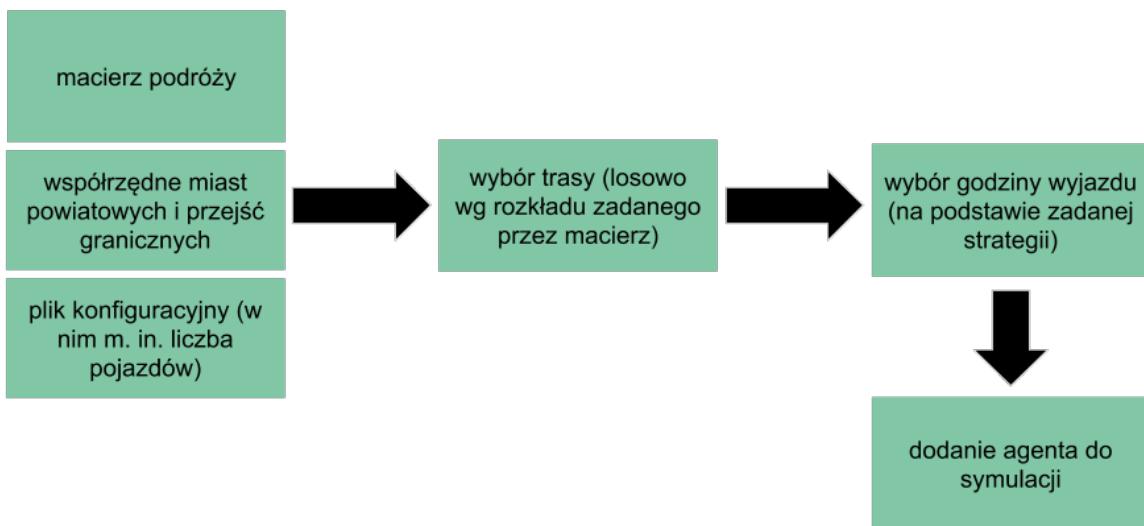
Rysunek 4.4: Schemat działania programu Mopsim

Program Mopsim wykorzystuje silnik programu MATSim do przeprowadzania symulacji. Poniższy schemat pokazuje zarys jego działania. Użytkownik programu Mopsim wprowadza dane wejściowe – plik z danymi o MOP-ach w formacie csv, macierze podróży pomiędzy poszczególnymi miastami powiatowymi dla trzech rodzajów pojazdów – samochodów osobowych, ciężarowych i autobusów w formacie csv oraz mapę sieci drogowej w formacie OSM (*OpenStreetMap*[10]). Dane te są przetwarzane kolejno na odpowiednie pliki konfiguracyjne wykorzystywane przez MATSim – opis obiektów powiązanych z odcinkami drogi (*facilities*), opis agentów i ich planów oraz opis sieci drogowej. Użytkownik może dodatkowo zmodyfikować sieć dróg lub siatkę MOP-ów, dodając lub usuwając odpowiednie obiekty. Następnie przeprowadzana jest symulacja, w wyniku której tworzone są wykresy i inne dane analityczne.

4.3.1. Generowanie planów podróży

Generowanie planów podróży odbywa się na podstawie macierzy dla trzech rodzajów pojazdów (samochody osobowe, pojazdy ciężarowe i autobusy), do których użytkownik podaje ścieżki w pliku konfiguracyjnym. Macierz podróży reprezentowana jest przez plik csv. Liczba w n -tym wierszu i m -tej komórce macierzy oznacza, ile pojazdów danego typu przejeżdża średnio w ciągu dnia z punktu n do punktu m . Punkty o odpowiednich indeksach odpowiadają miastom powiatowym i przejściom granicznym. Ich współrzędne umieszczone są w oddzielnym pliku.

W module generującym plan podróży macierze są traktowane jak rozkład prawdopodobieństwa, według którego losowane są punkty startowe i końcowe dla poszczególnych agentów,



Rysunek 4.5: Schemat działania generatora planów podróży

których jest tyle, ile zostało określone w pliku konfiguracyjnym. Po wylosowaniu odpowiednich punktów, na podstawie zadanej strategii czasu wyjazdu wybierana jest godzina, o której agent wyjedzie w trasę. Następnie agent dodawany jest do symulacji. Opis wygenerowanych planów podróży agentów zostaje zapisany do pliku `travel_plan.xml`.

4.3.2. Generowanie planu MOP-ów

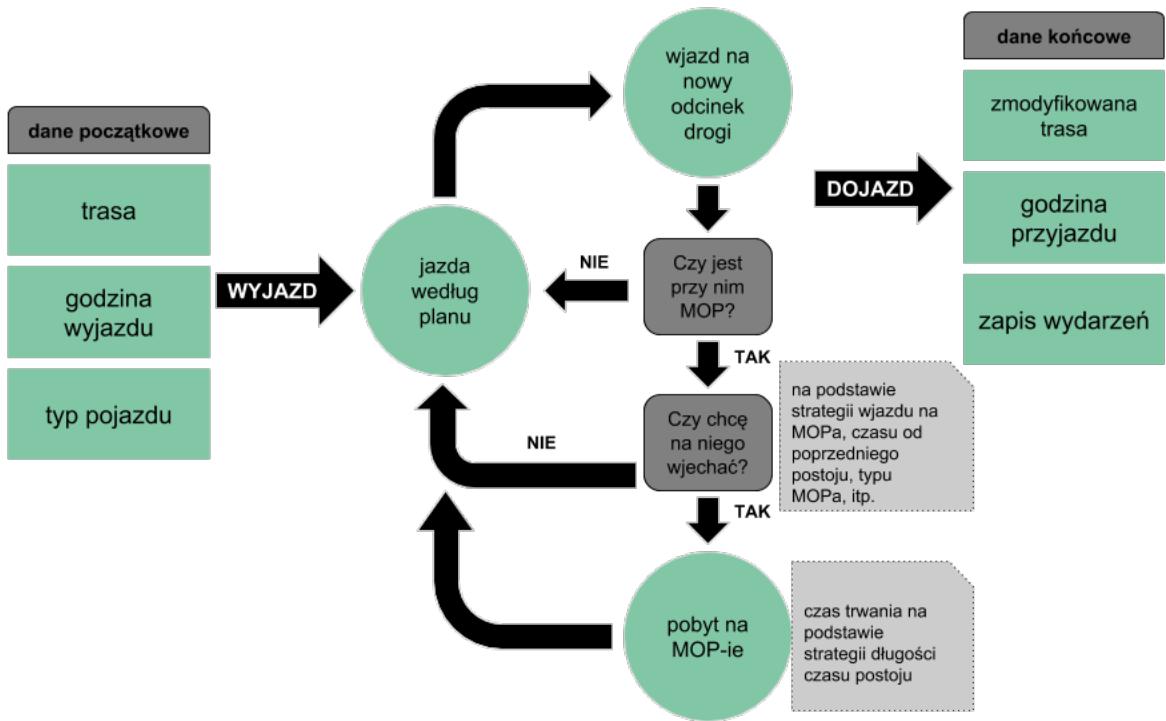
Generowanie planu MOP-ów odbywa się na podstawie danego pliku csv, w którym kolejne kolumny oznaczają współrzędne MOP-a (w układzie 92 lub CH1903, w zależności od wybranej opcji w pliku konfiguracyjnym), lokalizację, nazwę, liczby miejsc parkingowych dla samochodów, pojazdów ciężarowych, busów i inne dane o obiektach znajdujących się na MOP-ie. Każdy z MOP-ów zostaje przyporządkowany do najbliższego odcinka drogi, który znajduje się po jego lewej stronie. Opis MOP-ów ze wspólnymi zostaje zapisany w pliku `facilities.xml`, zaś ich atrybutów – w pliku `facilities_attributes.xml`.

4.3.3. Opis działania agenta w czasie symulacji

Każdy z agentów biorących udział w symulacji wyposażony jest w dane początkowe:

- Trasę podróży, wygenerowaną na podstawie punktu startowego, końcowego i siatki drogowej przez moduł MATSimu – *Router*. W pliku konfiguracyjnym istnieje możliwość określenia algorytmu wyboru ścieżki (Dijkstra, A*).
- Godzinę wyjazdu, na podstawie wygenerowanego planu.
- Typ pojazdu – samochód osobowy, ciężarowy lub autobus.

Symulacja w MATSimie przeprowadzana jest w sekundowych iteracjach. Pojazd wyrusza w trasę o ustalonej godzinie i jedzie zgodnie z określoną trasą. Za każdym razem, gdy agent wejeździ na nowy odcinek drogi sprawdzane jest, czy znajduje się przy nim MOP. Jeżeli tak, to na podstawie wybranej funkcji decyzyjnej, czasu od poprzedniego postoju i typu pojazdu podejmowana jest decyzja, czy pojazd zamierza skorzystać z MOP-a. Jeżeli tak, to na podstawie typu pojazdu, czasu od poprzedniego postoju, godziny i wybranej strategii



Rysunek 4.6: Schemat działania agenta w czasie symulacji

ustalana jest długość postoju na MOP-ie. Plan agenta ulega modyfikacji – zostaje do niego dodany pobyt na MOP-ie. Po ustalonym czasie agent wraca na trasę i podąża dalej zgodnie z pierwotnym planem.

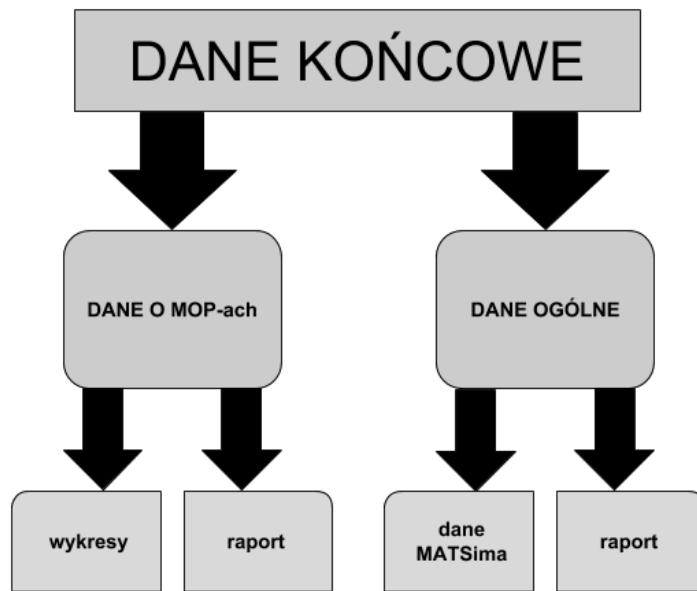
Końcowe dane symulacji zawierają informacje o zmodyfikowanych planach agentów, godzinie przyjazdu i innych wydarzeniach na drodze – między innymi wjazdach na poszczególne odcinki drogi.

4.3.4. Mopsim - pliki wyjściowe

Dla każdej symulacji w programie Mopsim tworzony jest katalog, w którym umieszczone są dane wyjściowe. Jest on nazwany podanym w pliku konfiguracyjnym id symulacji lub, jeżeli nie zostało ono określone, domyślną nazwą postaci `sim_yyyyMMdd-HHmmss`, a zatem znacznikiem czasu. W folderze utworzone są dwa podkatalogi - `MOPs` i `simulation_data`. W pierwszym z nich utworzony jest katalog dla każdego z MOP-ów. Umieszczone są w nim raport w postaci pliku tekstowego i csv, a także wykresy przedstawiające w podziale na godziny takie dane jak:

- Liczba pojazdów danego typu, które wjechały na MOP-a.
- Liczba pojazdów przejeżdżających obok MOP-a.
- Procentowe wykorzystanie MOP-a.
- Procentowy udział pojazdów wjeżdżających na MOP-a wśród wszystkich uczestników ruchu.

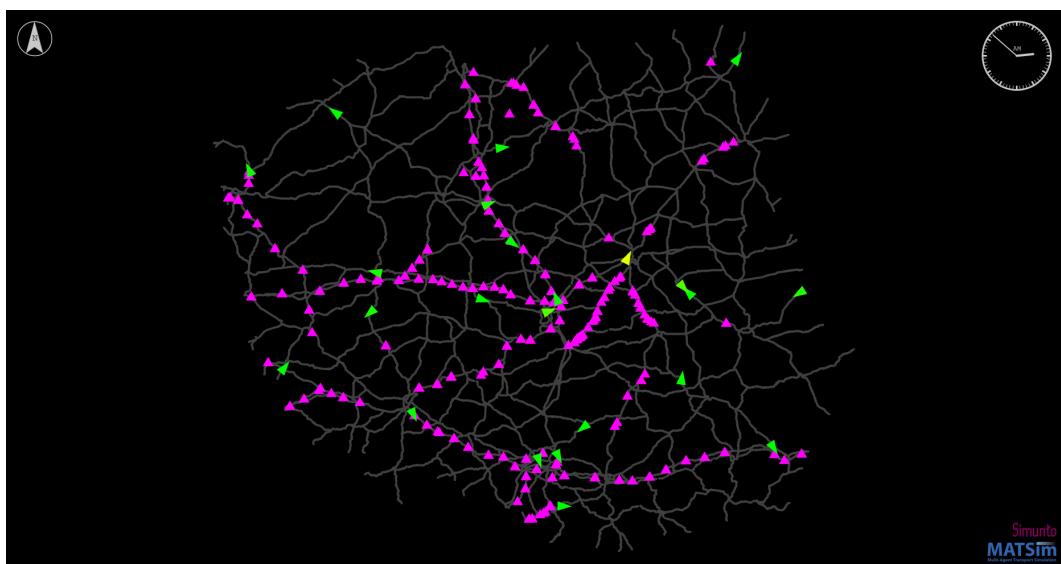
W raportach umieszczone są zsumowane dane dla całego dnia. W folderze `simulation_data`



Rysunek 4.7: Podział plików wyjściowych symulacji

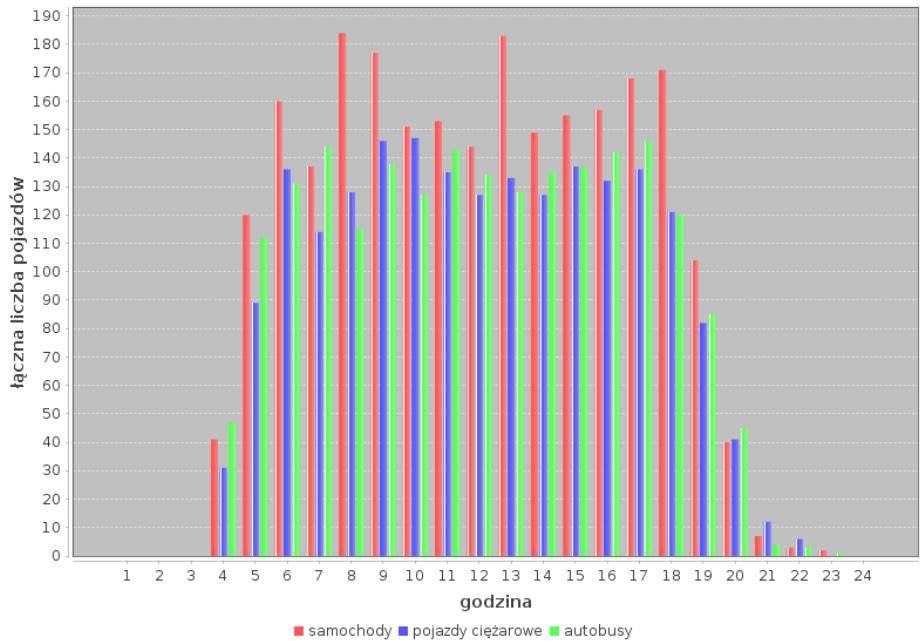
znajduje się raport, w którym znajdują się informacje o ustawieniach i ścieżki do plików wejściowych symulacji. Znajdują się w nim również dane dotyczące łącznej liczby pojazdów wjeżdżających na parkingi, przejeżdżających obok nich i średniej długości pobytu na MOP-ie. Oprócz tego w katalogu `simulation_data` umieszczony jest folder z danymi wyjściowymi MATSima, w którym znajdują się między innymi:

- Plik z logiem symulacji, zawierający komunikaty o czasie działania, przejściach do kolejnych etapów, ostrzeżeniach i ewentualnych błędach.
- Wykres przedstawiający czas spędzony w poszczególnych etapach symulacji.



Rysunek 4.8: Zrzut ekranu z symulacji programem Via. Różowe trójkąty oznaczają Miejsca Obsługi Podróżnych, zielone – poruszające się pojazdy.

Łączna liczba pojazdów wjeżdżających i przejażdżających obok MOPa nr 70



Rysunek 4.9: Przykładowy wykres wygenerowany podczas symulacji

- Skompresowane pliki konfiguracyjne - główny, z siatką drogową, planami podróży, a także *facilities*, czyli MOP-ami wykorzystanymi w symulacji.
- Skompresowany plik z *eventami*, które miały miejsce podczas symulacji.

Pliki te mogą zostać wykorzystane w jednym z programów służących do wizualizacji danych z symulacji - Simunto Via lub OTF Visualizer. Pierwszy z nich to program komercyjny, który w bezpłatnej wersji próbnej umożliwia wizualizację maksymalnie pięciuset pojazdów. Jego zalety to prosty interfejs, bogate możliwości konfiguracji i ciekawe efekty graficzne. OTF Visualizer to program będący składową MATSim, będący aktualnie w fazie eksperymentalnej. Jego zaletą jest możliwość przeprowadzania symulacji z dowolną liczbą pojazdów, zaś wadami - częściowa niestabilność i mniejsze możliwości graficzne.

4.4. Mopsim - instrukcja użytkownika

4.4.1. Konfiguracja

W celu przeprowadzenia symulacji należy poprawnie skonfigurować program Mopsim. Główny plik konfiguracyjny aplikacji znajduje się w katalogu `src/main/CONF` pod nazwą `config.xml`. Wybrane pola tego pliku mogą zostać zmodyfikowane przez użytkownika:

- Pod parametrem `inputNetworkFile` z modułu `network` należy podać ścieżkę do pliku z opisem sieci drogowej dla programu MATSim. Domyślna wartość parametru to `poland_network.xml`. Pod tym adresem znajduje się wygenerowana siatka uwzględniająca wszystkie drogi krajowe w Polsce. W dalszej części tekstu znajduje się instrukcja, jak wygenerować własną siatkę.

- Parametr `numberOfThreads` z modułu `qsim` oznacza liczbę wątków wykorzystywanych przez symulację. Powinien zostać ustawiony z uwzględnieniem parametrów maszyny, na której ma zostać przeprowadzona symulacja.
- Parametr `inputCRS` z modułu `facilities` powinien zostać ustawiony zgodnie z układem współrzędnych, jaki został wykorzystany w pliku z opisem MOP-ów. Układ jest określony na podstawie kodów **EPSG** (*European Petroleum Survey Group*). Domyślną wartością parametru jest `EPSG:2180`, co odpowiada polskiemu układowi współrzędnych 92. Sieć drogowa opisana jest w układzie CH1903, posiadającym kod `EPSG:21781`. Współrzędne MOP-ów są automatycznie konwertowane przez aplikację na ten układ.

Oprócz powyższych parametrów, modyfikujących pracę biblioteki MATSim, oddzielnego modułu `mopsim` służy do konfiguracji elementów ściśle powiązanych z Mopsimem. W module `mopsim` można ustawić wartości następujących parametrów:

- `simulationId` - id symulacji. Jeżeli nie zostanie podane, ustawione jest na opisany wcześniej znacznik czasu.
- `carNr`, `truckNr`, `busNr` - parametry określające liczbę pojazdów danego typu biorących udział w symulacji. Domyślne wartości – `100000`, `40000`, `0`.
- `carPath`, `truckPath`, `busPath` - ścieżki do plików z macierzami podróży dla poszczególnych pojazdów. Domyślna wartość – `src/main/resources/travel_matrices/car_matrix.csv` i analogicznie dla pozostałych typów pojazdów.
- `timeDistribution` – nazwa strategii wyboru czasu wyjazdu dla pojazdów. Domyślna wartość - `BASIC_DISTRIBUTION`.
- `carEnter`, `truckEnter`, `busEnter` – nazwy strategii decyzji o wjeździe na MOP-a dla poszczególnych typów pojazdów. Domyślne wartości to kolejno `BASIC_STRATEGY`, `TRUCK_STRATEGY`, `BUS_STRATEGY`.
- `carStay`, `truckStay`, `busStay` – nazwy strategii długości czasu pobytu na MOP-ie dla poszczególnych typów pojazdów. Domyślne wartości to kolejno `CAR_STAY_STRATEGY`, `TRUCK_STAY_STRATEGY`, `BUS_STAY_STRATEGY`.

Więcej o poszczególnych strategiach, ich nazwach i dodawaniu nowych w podrozdziale Strategie.

4.4.2. Przeprowadzanie symulacji

Program Mopsim został napisany z wykorzystaniem **Apache Maven**. Aby pobrać i zainstalować Mavena na systemie Debian, należy w wierszu poleceń wpisać polecenia:

```
sudo apt-get install default-jdk
sudo apt install maven
```

Szczegóły instalacji na inne platformy można znaleźć na oficjalnej stronie programu.

Aby skompilować aplikację, należy w katalogu zawierającym plik `pom.xml` uruchomić polecenie `mvn compile`. Podczas pierwszego wykonania tej komendy zostają pobrane wszystkie wymagane przez aplikację pakiety.

Po skompilowaniu, aplikację można uruchomić poleceniem `mvn exec:java`. W czasie wykonywania symulacji, na standardowe wyjście zostaną wypisane komunikaty, które będzie można odczytać po jej wykonaniu w logu. Folder danej symulacji, wraz z opisanymi wcześniej plikami wyjściowymi, został wygenerowany pod ścieżką `src/main/SIMULATIONS/id_symulacji`.

4.4.3. Generowanie siatki drogowej

Sugerowanym sposobem generowania sieci drogowej jest użycie interfejsu graficznego programu Mopnik. Innym sposobem jest wygenerowanie siatki z poziomu programisty. Aby to zrobić, należy stworzyć obiekt klasy programu Mopsim o nazwie *NetworkCreator*, znajdującej się w pakiecie `network`, podając jako parametry konstruktora ścieżkę do pliku wejściowego osm i ścieżkę, do której ma zostać zapisany wyjściowy plik.

4.4.4. Strategie

Kluczowymi elementami programu Mopsim, odgrywającymi ważną rolę w przeprowadzaniu symulacji są strategie:

- *TimeDistribution* – strategia wyboru godzin wyjazdu pojazdów. Obecnie zaimplementowana jest strategia `BASIC_DISTRIBUTION`, w której czasy wyjazdu kolejnych pojazdów są losowane zgodnie z rozkładem jednostajnym.
- *MOPEnterStrategy* – strategia decyzji o zjeździe na MOP-ie. Obecnie zaimplementowane strategie to:
 - `RANDOM_STRATEGY`, która ma charakter testowy – pojazdy zjeżdżają na MOP-a z prawdopodobieństwem równym $\frac{1}{2}$.
 - `BASIC_STRATEGY` – domyślna strategia dla samochodów. Losowana jest jedna liczba z przedziału 3600-32400 według rozkładu jednostajnego. Pojazd wjeżdża na MOP, jeżeli od czasu poprzedniego zatrzymania minęło więcej sekund, niż wynosi wylosowana liczba.
 - `TRUCK_STRATEGY` – domyślna strategia dla pojazdów ciężarowych i autobusów. Losowana jest liczba według rozkładu normalnego o średniej 14400 i odchyleniu standardowym 3600. Pojazd wjeżdża na parking, jeżeli od poprzedniego zatrzymania minęło więcej czasu, niż wynosi wylosowana liczba.
- *MOPStayStrategy* – strategia długości pobytu na MOP-ie. Obecnie zaimplementowane strategie:
 - `CAR_STAY_STRATEGY` – domyślna strategia dla samochodów. Czas pobytu losowany jest według rozkładu wykładniczego o wartości oczekiwanej wynoszącej 23 minuty.
 - `TRUCK_STAY_STRATEGY` – domyślna strategia dla pojazdów ciężarowych. Najpierw, ze względu na godzinę i czas od poprzedniego postoju, wybierany jest charakter postoju – krótki (wymuszony przepisami, zwykle w celu tankowania, spożycia posiłku lub skorzystania z toalety) lub długi (zwykle połączony z noclegiem). Jeżeli został wybrany postój krótki, to długość pobytu jest losowana zgodnie z rozkładem normalnym o wartości oczekiwanej wynoszącej 45 minut i odchyleniu standardowym 15 minut. Dla pobytu długiego, parametry rozkładu normalnego wynoszą odpowiednio 9 godzin i półtorej godziny.

Czasy postoju i rozkłady zostały dobrane na podstawie danych pochodzących z pomiarów i badań ankietowych.

4.4.5. Mopsim - dodanie własnych strategii

By przeprowadzać efektywne symulacje, zachowanie agentów powinno jak najbardziej odpowiadać rzeczywistości. Z poziomu programisty do programu Mopsim można dodać kolejne strategie. W pakiecie `strategies` znajdują się interfejsy dla każdego z rodzajów strategii – odpowiednio `TimeDistribution`, `MOPEnterStrategy` i `MOPStayStrategy`. By dodać strategię, należy zaimplementować dany interfejs (w każdym przypadku składający się z dwóch metod – `getIdentifier()`, zwracającej unikalny identyfikator strategii oraz odpowiedniej funkcji decyzyjnej) oraz rozszerzyć klasę `StrategyUtils`, nadpisując metody zwracające odpowiednie strategie.

4.5. Mopsim - opis implementacji

Apache Maven, z którego korzysta Mopsim, to narzędzie automatyzujące budowę oprogramowania na platformę Java. Poszczególne funkcje Mavena realizowane są poprzez wtyczki, które są automatycznie pobierane przy ich pierwszym wykorzystaniu. Plik określający sposób budowy aplikacji nosi nazwę **POM-u**.

4.5.1. Podział plików

W katalogu głównym aplikacji umieszczony jest plik POM oraz podkatalog `src/main`. Znajdują się w nim 3 kolejne podkatalogi:

- `CONF`, w którym umieszczone są pliki konfiguracyjne aplikacji.
- `java`, w którym umieszczone są pliki źródłowe.
- `resources`, w którym umieszczone są dane potrzebne do działania programu:
 - katalog `mop_data` zawierający plik z danymi o MOP-ach.
 - katalog `town_coordinates` zawierający plik ze współrzędnymi miast powiatowych i przejść granicznych.
 - katalog `travel_matrices` zawierający macierze podróży dla różnych typów pojazdów według danych z Generalnego Pomiaru Ruchu.

Dodatkowo, przy pierwszej symulacji w katalogu `src/main` tworzony jest podkatalog `SIMULATIONS`, w którym umieszczone zostaną pliki wyjściowe kolejnych symulacji.

4.5.2. Opis poszczególnych klas

- `default_package`
 - `MOPSimRun` – klasa uruchamiająca aplikację.
 - `MOPSimulator` – główna klasa odpowiedzialna za działanie aplikacji i sterowanie nią.
 - `ControllerModifier` – klasa modyfikująca kontroler – moduł biblioteki MATSim odpowiedzialny za kontrolę symulacji. Jej metody umożliwiają dodanie do symulacji odpowiednich obiektów typu `EventHandler`, reagujących na zdarzenia oraz `MopsimListener`, reagujących na konkretne etapy symulacji.
- `config_group`

Rysunek 4.10: Struktura plików źródłowych programu Mopsim

```
src/main/java
    └── ControlerModifer.java
    └── MOPSimRun.java
    └── MOPSimulator.java
    └── config_group
        └── MOPSimConfigGroup.java
    └── events
        └── MOPAfterSimStepListener.java
        └── MOPBeforeSimStepListener.java
        └── MOPEEnterEvent.java
        └── MOPLeaveEvent.java
        └── MOPLinkEnterEventHandler.java
        └── MOPLinkLeaveEventHandler.java
    └── handlers
        └── MOPHandler.java
    └── mop
        └── MOP.java
    └── network
        └── modifiers
            └── NetworkModifiers.java
        └── NetworkCreator.java
    └── plancreator
        └── FacilityPlanCreator.java
        └── TravelPlanCreator.java
    └── strategies
        └── mop_enter
            └── MOPEEnterStrategy.java
            └── BasicStrategy.java
            └── RandomStrategy.java
            └── TruckStrategy.java
        └── mop_stay
            └── MOPStayStrategy.java
            └── ExpStayStrategy.java
            └── TruckStayStrategy.java
        └── time_distribution
            └── TimeDistribution.java
            └── BasicDistribution.java
        └── StrategyUtils.java
    └── utils
        └── FileUtils.java
        └── ReportUtils.java
        └── TimeUtils.java
```

- *MOPSimConfigGroup* – klasa implementująca moduł `mopsim` w pliku konfiguracyjnym i obsługująca go.
- **events**

- *MOPEnterEvent*, *MOPLeaveEvent* – implementacje interfejsu *Event* z biblioteki MATSim. Odpowiadają zdarzeniom wjazdu i wyjazdu z MOP-a.
- *MOPLinkEnterEventHandler*, *MOPLinkLeaveEventHandler* – implementacje interfejsów obsługi zdarzeń z biblioteki MATSim. Definiują zachowanie aplikacji w momencie wjazdu i wyjazdu agentów z odcinków dróg.
- *MOPBeforeSimStepListener*, *MOPAfterSimStepListener* – implementacje interfejsów obsługi momentów iteracji z biblioteki MATSim. Definiują zachowanie agentów na wjazdach i wyjazdach z parkingów.

- **handlers**

- *MOPHandler* – klasa zarządzająca siatką MOP-ów.

- **mop**

- *MOP* – implementacja MOP-a.

- **network**

- *modifiers*/*NetworkModifier* – klasa z metodami umożliwiającymi modyfikację siatki drogowej.
- *NetworkCreator* – klasa tworząca plik z siatką drogową na podstawie pliku osm.

- **plancreator**

- *FacilityPlanCreator* – klasa tworząca odpowiednie pliki konfiguracyjne na podstawie pliku z danymi o MOP-ach.
- *TravelPlanCreator* – klasa tworząca plik konfiguracyjny z planami poszczególnych agentów.

- **strategies** – został opisany wyżej, w opisie strategii.

- **utils**

- *FileUtils* – szereg narzędzi pomocnych przy obsłudze plików.
- *ReportUtils* – narzędzia pomocnicze do tworzenia raportów.
- *TimeUtils* – narzędzia pomocnicze do obsługi czasu.

Rozdział 5

Mopsik – Strona serwerowa - API

Jednymi z podstawowych danych wczytywanych przez pisane przez nas programy (Mopsik–Mobile oraz Mopsik–Web) są dane o MOP-ach. Dane te są przedstawiane przez GDDKiA w postaci ogólnie dostępnego skoroszytu programu Excel (rozszerzenie .xlsx) zamieszczonego na stronie internetowej[12]. Sensownym wydał się pomysł zbudowania wspólnego interfejsu do przekazywania tych danych naszym aplikacjom. W tym momencie z danych z API korzystają w czasie rzeczywistym Mopsik–Mobile oraz Mopsik–Web. Ponadto w programie Mopnik, oprócz wczytywania danych o MOP-ach z plików, istnieje także opcja pobrania tych danych z serwera.

Podczas tworzenia aplikacji pewnym problemem stało się posługiwanie różnymi układami współrzędnych przez różne komponenty, z których korzystamy. Na przykład dane o lokalizacji MOP-ów, które otrzymaliśmy przez GDDKiA są w układzie 92[14] a Google Maps, którego używamy w naszej aplikacji mobilnej przyjmuje dane w układzie WSG84. Postanowiliśmy dodać do naszego API konwerter pomiędzy tymi układami.

5.1. Przypadki użycia API

Serwer ze stroną internetową, klient aplikacji mobilnej oraz program Mopnik może:

1. Pobrać informacje na temat pozycji oraz dostępnych usług na MOP-ach na podstawie najnowszego stanu przedstawionego do tej pory na stronie GDDKiA.
2. Pobrać dane o bieżącej zajętości MOP-ów.
3. Przekonwertować współrzędne pomiędzy układami WSG84 i układem 92.

5.2. Architektura

Na implementację API serwera składają się dwie części:

1. Skrypt cron
2. Aplikacja Django

Cron

Na serwerze zostało umieszczone polecenie, które regularnie sprawdza, czy na stronie GDDKiA nie pojawił się nowy arkusz .xlsx z MOP-ami. Jeśli tak, to jest wywoływana zdefiniowana przez nas w Django komenda *update_mops_db*, która aktualizuje bazę danych.

Aplikacja Django

Implementacja oparta jest o wersję 2.0 frameworku Django, Python 3.5 oraz bibliotekę django REST framework. Dane są przechowywane w bazie SQLite3. Na ten moment API odpowiada na następujące zapytania:

- */mops* – zwraca wszystkie dane o MOP-ach agregowane w bazie danych
- */taken* – zwraca dane o zajętości MOP-ów agregowane w bazie danych
- */wsg_to_92* – dla danej pary współrzędnych w formacie WSG84 zwraca parę współrzędnych w formacie 92
- */92_to_wsg* – dla danej pary współrzędnych w formacie 92 zwraca parę współrzędnych w formacie WSG84

Rozdział 6

Mopsik – Aplikacja mobilna

6.1. Przypadki użycia

Zarówno w aplikacji mobilnej, jak i na stronie internetowej podróżny może:

1. Znaleźć MOP-a w wyszukiwarce lub na mapie.
2. Poznać procentową zajętość miejsc parkingowych na MOP-ie dla każdego z trzech typów pojazdów (samochód osobowy, samochód ciężarowy, autobus).
3. Poznać liczbę wolnych miejsc parkingowych oraz ich liczbę ogółem.
4. Znaleźć kontakt do operatora MOP-a.
5. Sprawdzić jakie usługi znajdują się na MOP-ie.
6. Wyświetlić wizualizację danych o zajętości miejsc parkingowych.
7. Edytować preferencje wyświetlania danych dla konkretnych typów pojazdów.

Dodatkowo, w aplikacji mobilnej:

1. Mapa wyświetla aktualną lokalizację użytkownika, co umożliwia wybranie najbliższego MOP-a.
2. Istnieje możliwość dodanie MOP-a do ulubionych i późniejsze wybranie go z listy zapisanych.

6.2. Architektura i opis implementacji

6.2.1. Technologia

Aplikacja mobilna została napisana w języku **JavaScript** z użyciem biblioteki **React Native**[8] (dalej: RN). Dzięki temu pisząc jeden kod, otrzymujemy aplikację zarówno na telefony z Androidem, jak i iOS.



Aplikacja w RN zbudowana jest z komponentów, które posiadają stan (obiekt słownikowy) inicjowany w konstruktorze. Funkcja `render()` wywoływana jest przy pierwszym ładowaniu komponentu oraz przy każdej zmianie jego stanu.

Z komponentów budujemy widoki, czyli wszystko co jest widoczne na ekranie w danym momencie.

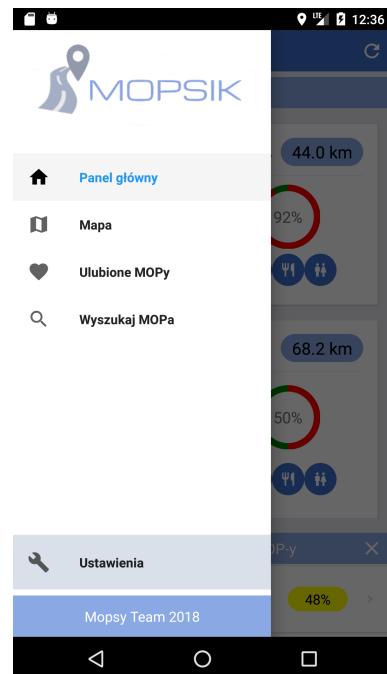
Cała aplikacja objęta jest nawigatorem, który odpowiedzialny jest za generowanie menu bocznego oraz przechodzenie między widokami zgodnie z wolą użytkownika.

6.2.2. Zakładki

Menu boczne zawiera 5 zakładek:

1. *Konfiguracja/Ustawienia* – umożliwia wybór typów pojazdów, którymi jest zainteresowany użytkownik
2. *Home* – wyświetla dwa najbliższe i trzy ostatnio wyświetlane MOP-y
3. *Mapa* – wyświetla mapę z zaznaczonymi MOP-ami
4. *Ulubione MOP-y* – wyświetla listę MOP-ów zapisanych do ulubionych
5. *Wyszukiwarka MOP-ów* – umożliwia wyszukiwanie MOP-a po nazwie, miejscowości, numerze drogi

Istnieje także szósty widok - *Szczegółowe informacje o MOP-ie*, do którego można przejść z *Ulubionych*, *Wyszukiwarki* i *Mapy*. Zawiera on wszystkie informacje o MOP-ie, jego wyposażenie oraz zajętości i liczby miejsc parkingowych dla różnych typów pojazdów.



Rysunek 6.1: Menu boczne

6.2.3. Konfiguracja aplikacji i ustawienia

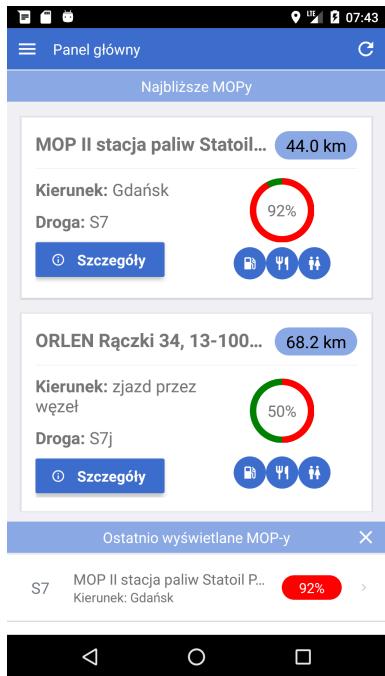
Przy pierwszym uruchomieniu aplikacji otwiera się panel konfiguracyjny, w którym należy ustawić dwa parametry:

1. *Główny typ pojazdu* (pole jednokrotnego wyboru) – w miejscach, w których znajdują się tylko ogólne informacje o MOP-ie wyświetlamy zajętość tylko dla głównego typu pojazdu
2. *Wszystkie typy pojazdów* (pole wielokrotnego wyboru), którymi zainteresowany jest użytkownik – w szczegółowych informacjach o MOP-ie wyświetlane są zajętości dla wszystkich zaznaczonych typów

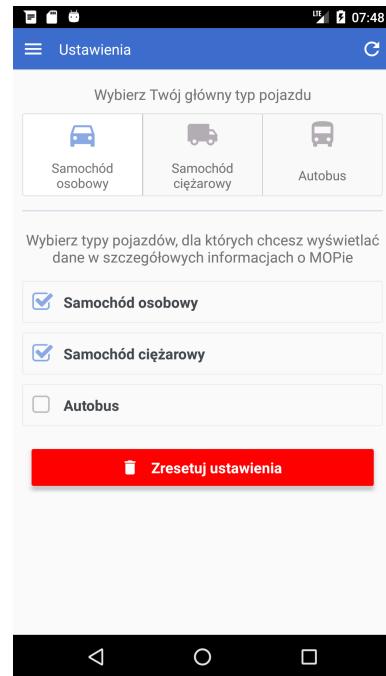
Typy pojazdów

1. Samochód osobowy
2. Samochód ciężarowy
3. Autobus

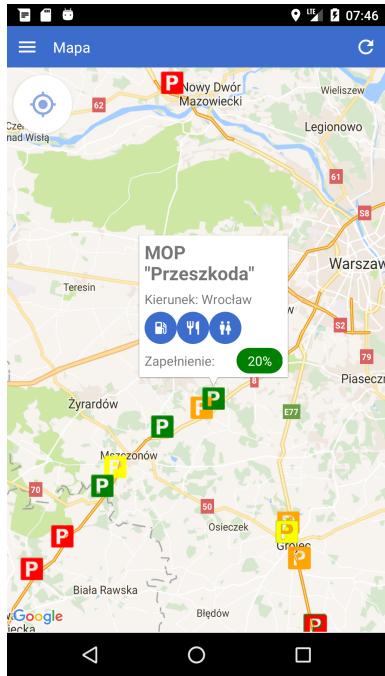
Podział został wykonany w oparciu o zróżnicowane potrzeby użytkowników tych pojazdów, a także znaczące różnice w gabarytach, a co za tym idzie – wymiarach potrzebnych miejsc parkingowych. Konfigurację aplikacji można zmienić w każdej chwili w zakładce *Ustawienia*. W ustawieniach znajduje się także przycisk *Zresetuj ustawienia*, który wymazuje z urządzenia wszystkie informacje zapisane przez aplikację i uruchamia ją ponownie.



Rysunek 6.2: Panel główny



Rysunek 6.3: Ustawienia



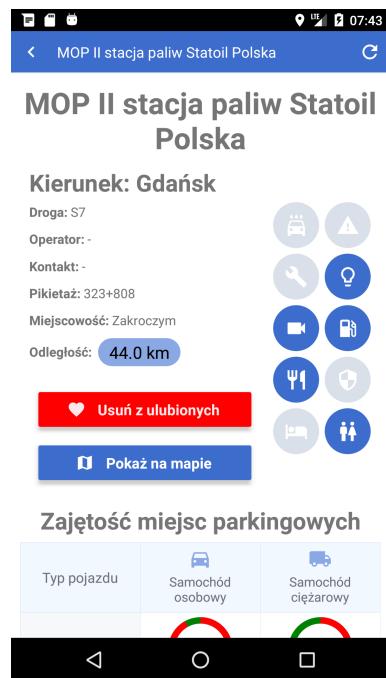
Rysunek 6.4: Mapa



Rysunek 6.5: Ulubione



Rysunek 6.6: Wyszukiwarka



Rysunek 6.7: Szczegóły

6.2.4. Google Maps

Warunki

Jedną z podstawowych funkcji aplikacji Mopsik jest wyświetlanie mapy Polski z zaznaczonymi MOP-ami. Do tego celu wykorzystaliśmy Google Maps API. Dostęp do samej mapy i oznaczanie na niej punktów są nieodpłatne, dopóki aplikacja jest darmowa i ogólnodostępna[9].

Działanie

Połączenie do API uwierzytelniane jest przez klucz wygenerowany na stronie <https://console.developers.google.com/apis/credentials>. Z każdym przeładowaniem widoku z mapą do API wysyłane jest zapytanie z prośbą o konkretny fragment mapy świata, który jest zwracany w odpowiedzi, a następnie wyświetlany w aplikacji. Na mapie znajdują się markery oznaczające MOP-y. Kolor markera jest uzależniony od zajętości MOP-a dla wybranego *głównego typu pojazdu*. Aplikacja wyświetla aktualną lokalizację użytkownika oraz umożliwia jej śledzenie (mapa porusza się wraz z użytkownikiem).

Wydajność

Dodawanie markerów na mapie, a szczególnie tych z własną ikoną, znacząco obniża wydajność mapy. Aby ją poprawić zastosowaliśmy kilka zabiegów:

1. Do mapy dodawane są tylko markery znajdujące się w obecnie widocznym obszarze. Wraz z przesuwaniem mapy ładowane są kolejne.
2. Mapa przeładowywana jest nie częściej niż co 2 sekundy. Zapobiega to ciągłe odświeżaniu gdy odczyty GPS urządzenia są niedokładne i nieprzerwanie drgają.

3. Wykorzystanie *onRegionChangeComplete* zamiast *onRegionChange* – obie funkcje wywoływane są gdy użytkownik porusza mapą, czyli przy zmianie aktualnie widocznego fragmentu. *onRegionChangeComplete* wywoływanie jest tylko raz – po zakończonym przeciągnięciu. Animacje są nieco mniej płynne, ale aplikacja nie zawiesza się przez ciągłe przeładowywania mapy i znaczników MOP-ów.

Bezpieczeństwo

Śledzenie lokalizacji użytkownika jest realizowane dzięki geolokacji urządzenia przenośnego – mapy Google nie biorą w niej udziału. Zapytanie wysyłane do Google zawierają jedynie obszar (współrzędne środka oraz przybliżenie), dla którego chcemy wyświetlić mapę. Jeżeli użytkownik aktywuje podążanie mapy za jego aktualnym położeniem, siłą rzeczy zapytania API będą wysyłać jego kolejne lokalizacje. Jednak zakładając korzystanie z map online – nie ma innej możliwości.

6.2.5. Zapytanie do API

Po otwarciu aplikacji wykonywane jest zapytanie do API */mops*, które zwraca wszystkie informacje o MOP-ach. Dane te są przetwarzane i zapamiętywane. Po wciśnięciu przycisku *odśwież* oraz po otwarciu każdej z zakładek, wykonywane jest zapytanie */taken* zwracające tylko aktualną liczbę zajętych miejsc parkingowych dla każdego z MOP-ów. Aplikacja aktualizuje te dane w zapisanych MOP-ach oraz przelicza nowe zajętości dla wszystkich typów pojazdów. Do każdego typu pojazdu, w każdym MOP-ie, przyporządkowywany jest kolor zgodnie z ustaloną legendą (np. dla małej zajętości, do 30% – zielony, a dla największej – czerwony). Ten kolor jest następnie wykorzystywany przy wizualizacji danych.

6.2.6. Użyte biblioteki

W aplikacji skorzystaliśmy z gotowych bibliotek RN, co znaczaco przyspieszyło i ułatwiło pracę.

1. **lodash**

<https://www.npmjs.com/package/lodash>

Dostarcza funkcji użytkowych w paradygmacie funkcyjnym. Przydatna w mapowaniu MOP-ów.

2. **react-native-elements**

<https://react-native-training.github.io/react-native-elements/>

Biblioteka zawierająca wiele gotowych komponentów. Skorzystaliśmy m.in. z *Drawer* (menu boczne), *Header* (nagłówek) i *earchBar* (wyszukiwarka).

3. **react-native-maps**

<https://github.com/react-community/react-native-maps>

Biblioteka umożliwiająca połączenie do Google Maps API i renderowanie map.

4. **react-native-restart**

<https://github.com/avishayil/react-native-restart>

Biblioteka umożliwia restart aplikacji z poziomu kodu. Wykorzystana przy resetowaniu ustawień.

5. react-native-svg

<https://github.com/react-native-community/react-native-svg>

Służy do grafiki wektorowej.

6. react-native-svg-circular-progress

<https://github.com/stssoftware/react-native-svg-circular-progress>

Wykorzystana do rysowania wykresów zajętości MOP-ów. Wymaga biblioteki *react-native-svg*.

7. react-native-swipeout

<https://github.com/dancormier/react-native-swipeout>

Umożliwia dodawanie przycisków pojawiających się po przesunięciu w bok komponentu. W naszej aplikacji odpowiada za pojawianie się przycisku *Usuń* na liście ulubionych MOP-ów.

8. react-native-table-component

<https://github.com/Gil2015/react-native-table-component>

Generuje tabelę z liczbą wolnych miejsc parkingowych w zakładce *Szczegółowe informacje o MOP-ie*.

9. react-native-vector-icons

<https://github.com/oblador/react-native-vector-icons>

Dostęp do ikon z <https://material.io/icons/>.

10. react-navigation

<https://reactnavigation.org>

Otoczka całej aplikacji. Odpowiada za nawigowanie wewnątrz aplikacji i poruszanie pomiędzy widokami.

6.2.7. Nawigacja

W aplikacji wykorzystaliśmy dwa typy nawigacji - stos (*StackNavigator*) i menu boczne (*DrawerNavigator*). Architektura nawigacji została przedstawiona na rysunku 6.8.

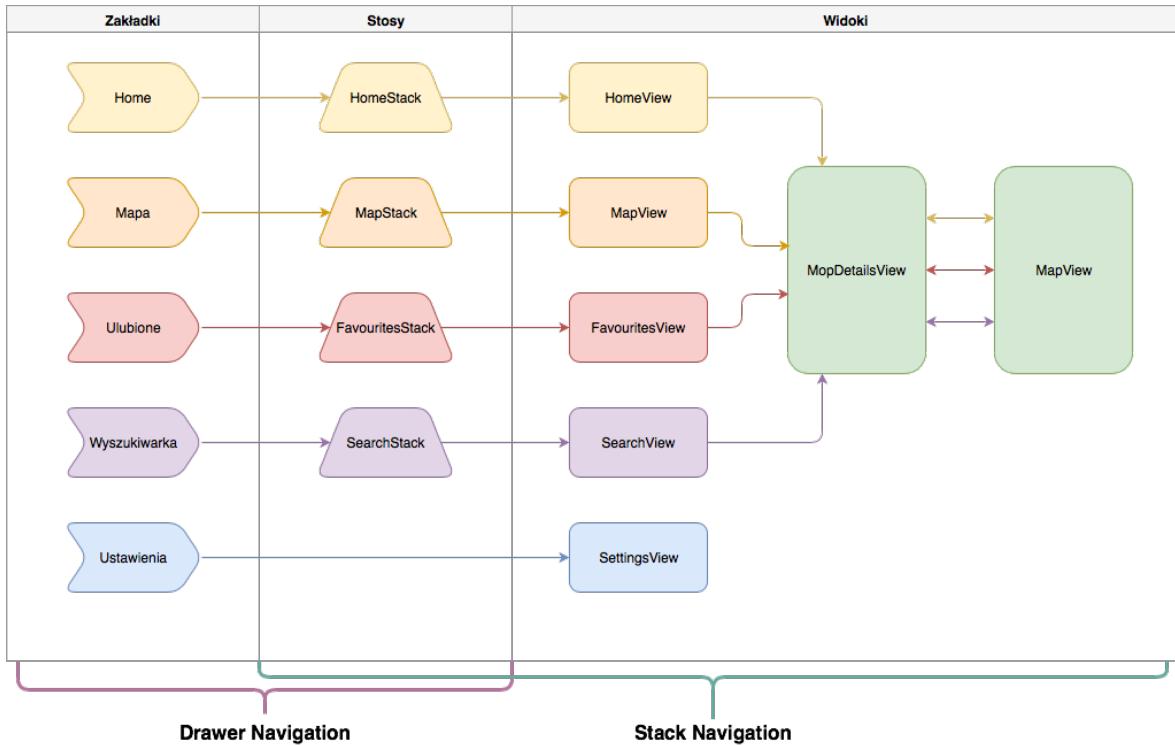
Najbardziej zewnętrznym elementem aplikacji jest komponent *DrawerNavigator*. Dodany jest do niego widok *SettingsView*, a także stosy *HomeStack*, *MapStack*, *FavouritesStack* i *SearchStack*. Menu boczne jest utworzone przez dostosowany do naszych potrzeb komponent *DrawerContent* – dodane zostało logo i autorzy.

Z zakładek *Home*, *Ulubione*, *Wyszukiwarka* i *Mapa* można przejść do widoku *MopDetailsView*, a stamtąd do *MapView* w trybie *Pokaż MOP-a na mapie* (focused). Za tą część nawigacji odpowiedzialny jest *StackNavigator*. Będąc w szczegółowych informacjach o MOP-ie i skupionej mapie w nagłówku pojawia się przycisk cofania.

Komponenty Stack obudowują nawigację stosową. Wewnątrz każdego ze stosów można poruszać się po trzech widokach - głównym, szczegółowym i mapie (wyśrodkowanej na wybranym MOP-ie, z otwartym dymkiem). W widokach głównych (tj. domowym, mapie, wyszukiwarce i ulubionych) widoczne są listy wielu MOP-ów. Po kliknięciu w wybrany MOP, aplikacja przenosi do widoku ze szczegółami wraz z odpowiednim parametrem. Na jego podstawie renderowane są informacje o tym wybranych MOP-ie. Ze szczegółów można poprzez kliknięcie w przycisk *Pokaż na mapie* przejść do *MapView* – o ile wcześniej nie przeszliśmy z mapy.

Czyli nie jest możliwe poruszanie się

mapa -> szczegóły(A) -> mapa -> szczegóły(B) -/-> mapa -> ... ,
ponieważ w widoku szczegóły(B) nie pojawia się przycisk *Pokaż na mapie*.



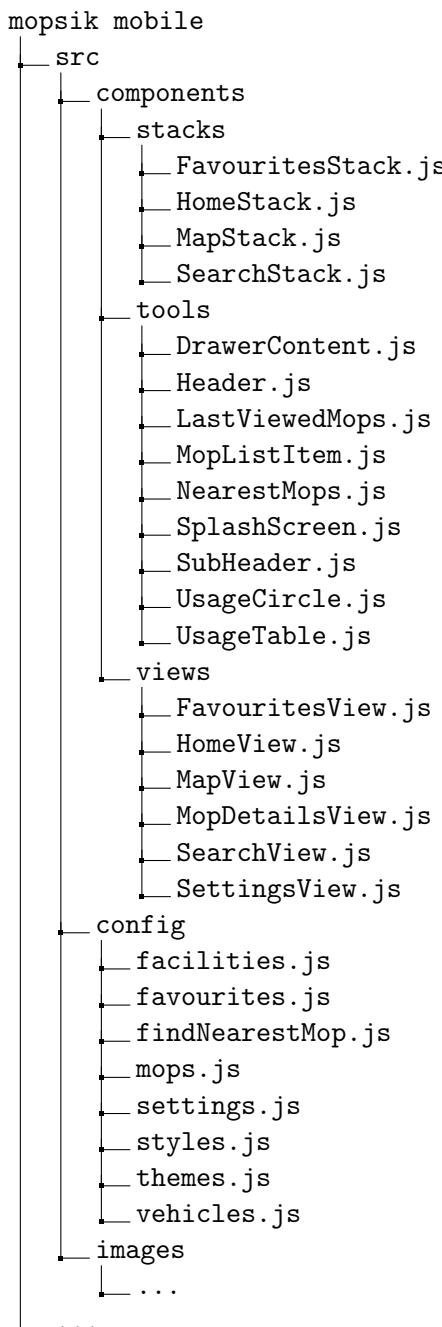
Rysunek 6.8: Schemat nawigacji aplikacji mobilnej Mopsik

6.2.8. Pozostałe komponenty

Na rysunku 6.9 przedstawiona została struktura plików Mopsika. Poza komponentami opisanymi w poprzednim punkcie, aplikacja korzysta z narzędzi znajdujących się w folderze *tools*.

1. *DrawerContent* – menu boczne
2. *Header* – nagłówek
3. *LastViewedMops* – komponent wyświetlający do trzech ostatnio wyświetlanych MOP-ów
4. *MopListItem* – element listy wyświetlający nazwę, drogę, kierunek i zajętość dla głównego typu pojazdu; wykorzystywany w *Ulubionych*, *Wyszukiwarce* i *Ostatnio wyświetlanych*
5. *NearestMops* – komponent wyświetlający dwa MOP-y znajdujące się najbliżej aktualnej lokalizacji użytkownika
6. *SplashScreen* – ekran powitalny; w trakcie jego wyświetlania aplikacja pobiera dane o MOP-ach z api oraz zapisane informacje z *AsyncStorage*
7. *SubHeader* – nagłówek sekcji; używany w ekranie domowym
8. *UsageCircle* – wykres kołowy zajętości miejsc parkingowych

9. *UsageTable* – tabela z liczbą i zajętością miejsc parkingowych dla typów pojazdów wybranych w ustawieniach



Rysunek 6.9: Struktura plików aplikacji mobilnej Mopsik

6.2.9. Pliki konfiguracyjne i funkcje

W folderze *config* znajdują się pliki, w których zdefiniowane są dane statyczne oraz funkcje odpowiedzialne za przetwarzanie danych.

1. *facilities* – zdefiniowane usługi dostępne na MOP-ach, odpowiadające im polskie nazwy oraz ikony, funkcje generujące komplet ikon do wyświetlania w informacjach o MOP-ie
2. *favourites* – funkcje odpowiedzialne za pobieranie MOP-ów zapisanych do ulubionych, zapisywanie, usuwanie
3. *findNearestMop* – funkcja, która po otrzymaniu współrzędnych punktu, zwraca dwa MOP-y znajdujące się najbliżej tego punktu oraz odległości do niego w kilometrach
4. *mops* – funkcje pobierające informacje o MOP-ach z API, dobierające kolory według legendarnej
5. *settings* – plik ze zmiennymi przechowującymi ustawienia aplikacji
6. *styles* – zdefiniowane style komponentów
7. *themes* – zdefiniowane motywy kolorystyczne
8. *vehicles* – typy pojazdów wraz z polskimi nazwami i ikonami

6.2.10. Zapisywanie danych

Podczas działania aplikacji dane zapisywane są w zmiennych znajdujących się m.in. w plikach mops.js i settings.js. Można nazwać je pseudo-globalnymi, ponieważ ich wartość można zmieniać w każdym komponencie, o ile zostały odpowiednio zimportowane. Informacje te są tracone wraz z zamknięciem aplikacji.

Dane dotyczące ustawień użytkownika oraz ulubionych MOP-ów nie znikają. Do tego celu wykorzystujemy AsyncStorage. Dane pobierane i zapisywane są asynchronicznie. Zasób ma postać słownika – każda wartość znajduje się pod nadanym jej kluczem.

Rozdział 7

Mopsik – Strona internetowa

7.1. Przypadki użycia

Przypadki użycia strony internetowej *Mopsik* zostały przedstawione w poprzednim rozdziale 6.1, razem z przypadkami użycia aplikacji mobilnej.

7.2. Architektura i opis implementacji

7.2.1. Technologia

Aplikacja internetowa została napisana w technologii ASP.NET MVC, w języku obiektowym C#. Frontend wykorzystuje także HTML, JavaScript oraz CSS. W aplikacji znajdują się trzy kontrolery odpowiedzialne za poszczególne elementy: Mapa, Wyszukiwarka, Szczegóły. W każdym kontrolerze zdefiniowane są akcje. Na podstawie aktualnego adresu wywoływany jest odpowiedni kontroler i akcja:

`<domena>/<nazwa kontrolera>/<nazwa akcji>/<parametry>.`



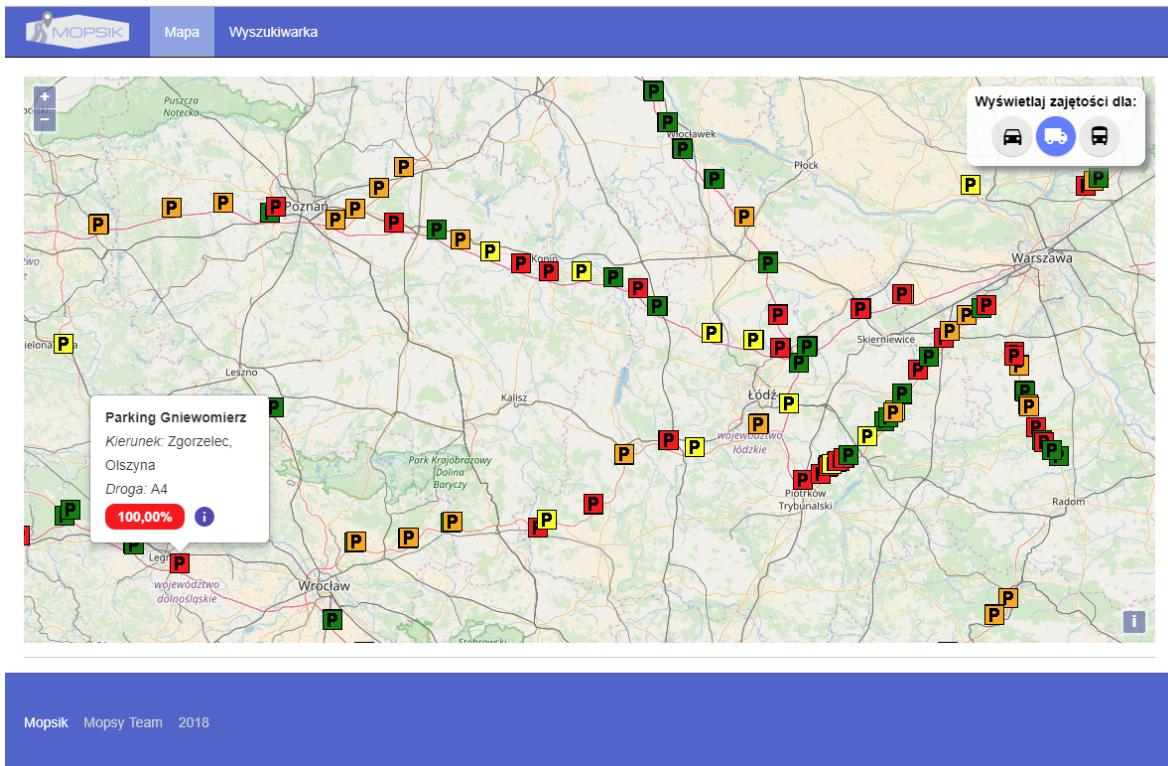
7.2.2. Zakładki

W górnej części strony znajduje się menu, w którym są dwie zakładki: *Mapa* i *Wyszukiwarka*. *Mapa* jest zakładką domyślną. Jest także trzeci widok – *Szczegóły*, do którego można przejść z obu zakładek, po wyborze jednego z MOP-ów.

Mapa

W tej zakładce znajduje się mapa Polski z OpenStreetMap[10]. Do jej wyświetlania wykorzystana została OpenLayers - biblioteka JavaScript do dynamicznych map. Na mapie oznaczone są MOP-y, a kolor znacznika odpowiada zajętości miejsc parkingowych dla wybranego typu pojazdu (skala analogiczna jak w aplikacji mobilnej). Typ pojazdu można zmienić w prawym, górnym rogu mapy, wybierając odpowiednią ikonę. Wysyła to zapytanie POST z odpowiednią wartością typu pojazdu i odświeża stronę z naniesionymi zmianami.

Po kliknięciu w znaczek wyświetla się dymek z podstawowymi informacjami o MOP-ie, a guzik z ikoną informacji przenosi do szczegółów – wywołuje *DetailsController* z akcją *Details* oraz parametrem id MOP-a.



Rysunek 7.1: Mapa

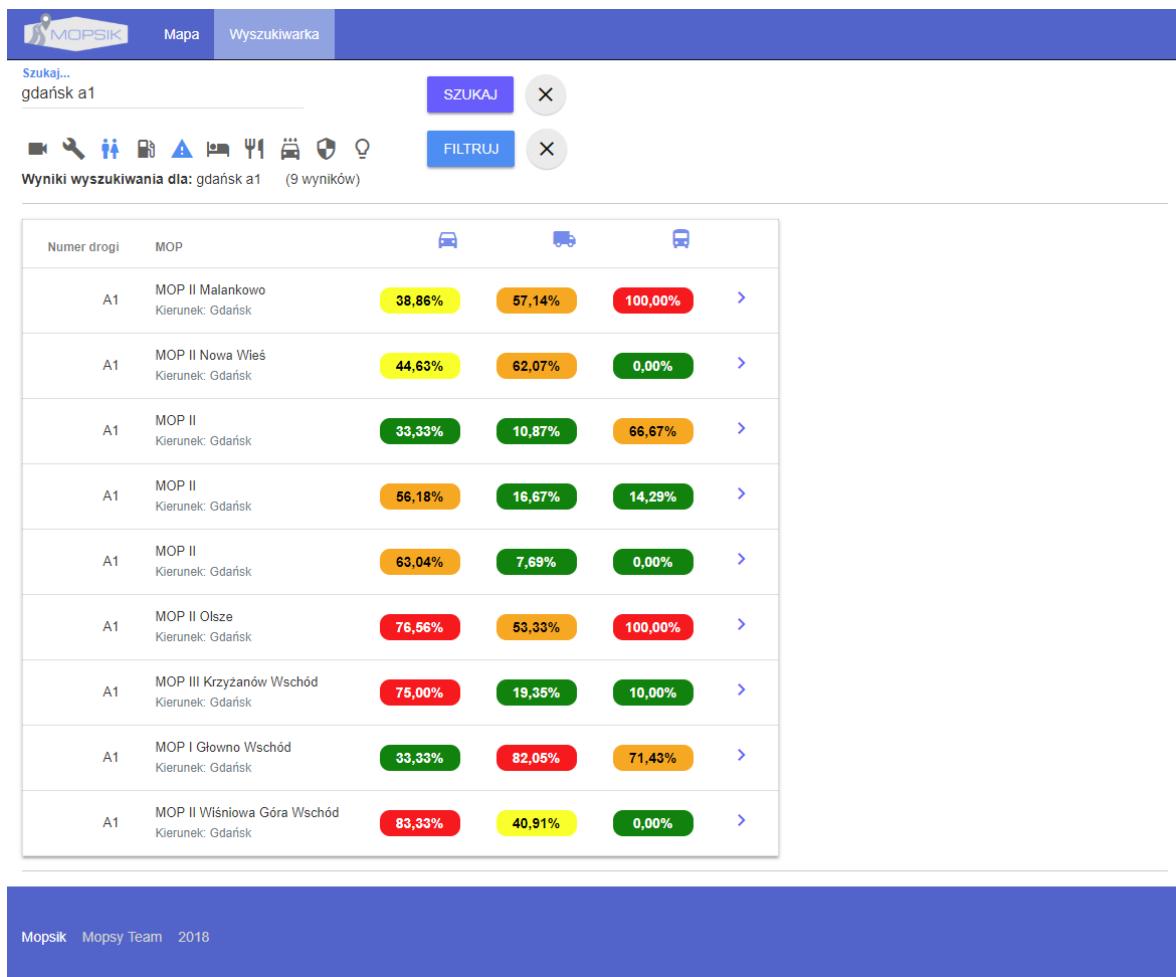
Wyszukiwarka

Aplikacja umożliwia wyszukiwanie MOP-ów po nazwie, miejscowości, numerze drogi i kierunku oraz filtrowanie po obecnych usługach. Przyciski *Szukaj* i *Filtruj* wysyłają zapytanie POST wraz z aktualną frazą wyszukiwania i zaznaczonymi usługami. Wyszukiwarka posiada *form* z polem *input* na wyszukiwany tekst oraz ukrytym polem *input*, które przesyła aktualnie zaznaczone filtry. W formularzu odpowiadającym za filtrowanie jest odwrotnie. MOP uznawany jest za pasujący do zapytania jeśli:

1. oferuje wszystkie usługi zaznaczone w filtrach
2. dla każdego ze słów zawartych w wyszukiwanym haśle, MOP zawiera to słowo w jednym z poniższych:
 - (a) nazwa
 - (b) miejscowość
 - (c) numer drogi
 - (d) kierunek

(wielkość liter nie jest istotna)

W liście rezultatów znajdują się wszystkie MOP-y, które spełniają warunki zapytania. Dla każdego z nich wyświetlane są: nazwa, numer drogi, kierunek, miejscowości oraz zajętość dla wszystkich trzech typów pojazdów. Kliknięcie w wiersz z MOP-em przenosi na stronę ze szczegółami.



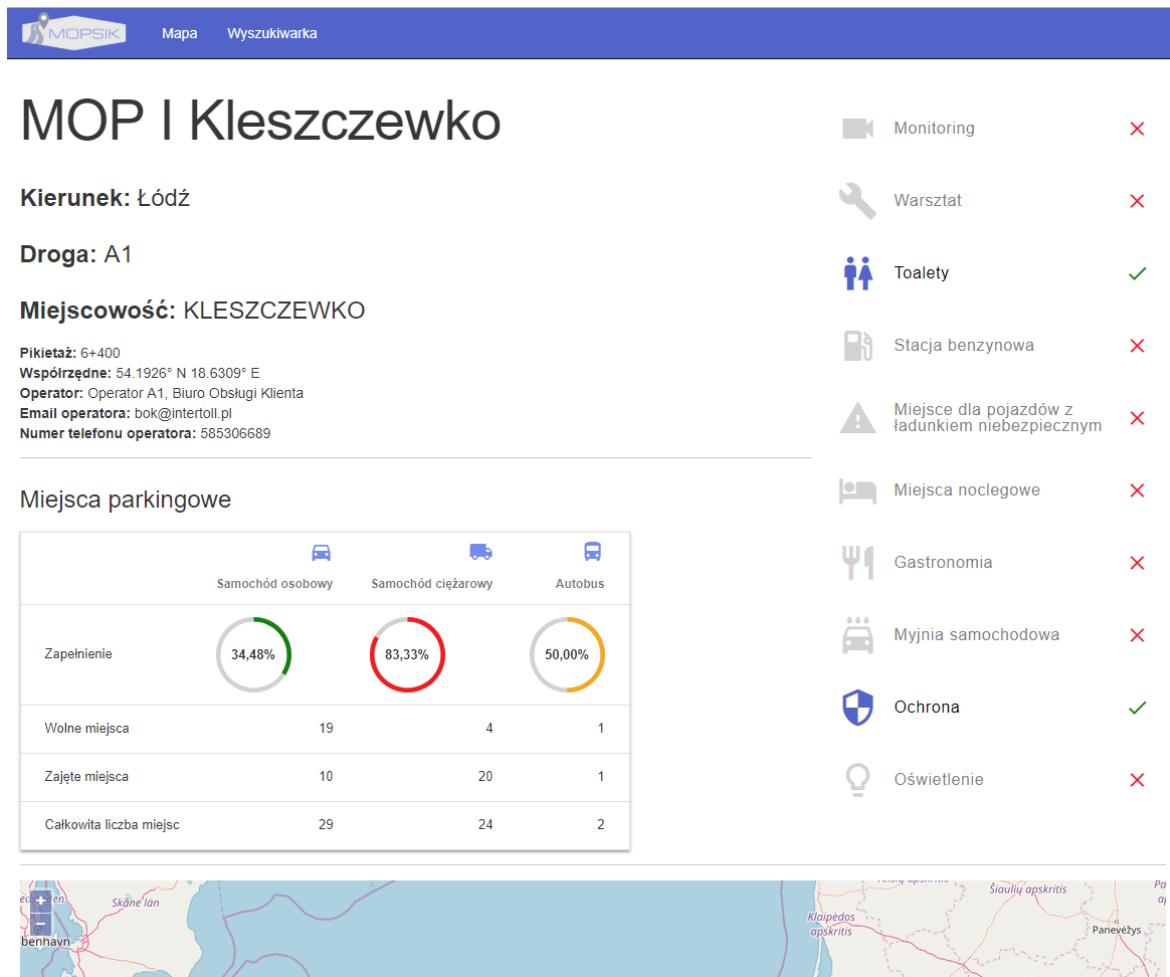
Rysunek 7.2: Wyszukiwarka

Szczegóły

W tej zakładce wyświetlane są szczegółowe informacje o konkretnym MOP-ie:

1. nazwa
2. kierunek
3. numer drogi
4. miejscowości
5. pikietaż
6. współrzędne geograficzne
7. operator i kontakt do niego – numer telefonu oraz e-mail
8. oferowane usługi
9. położenie na mapie
10. dla każdego typu pojazdu:

- (a) zajętość miejsc parkingowych w procentach + wizualizacja danych na wykresie
- (b) liczba wolnych miejsc parkingowych
- (c) liczba zajętych miejsc parkingowych
- (d) liczba miejsc parkingowych ogółem



Rysunek 7.3: Szczegóły

7.2.3. Zapytania do API i zapisywanie danych

Aplikacja pobiera wszystkie dane o MOP-ach z opisanego wcześniej API. Wykonuje zapytania do `/mops` i zapisuje je w pamięci aplikacji. Przy każdym przeładowaniu wyświetlanej strony sprawdzane jest jak dawno było wykonane ostatnie zapytanie do API. Jeśli czas ten jest większy niż 2 minuty, zapytanie jest wykonywane ponownie. W przeciwnym przypadku, dane są pobierane z pamięci aplikacji.

Parsowanie danych z API

Klasy `Mop`, `Operator`, `FacilitiesParser`, `CoordinatesParser` i `SpacesCount` biorą udział w parsowaniu danych z API, które są w formacie JSON. Przy każdym swoim polu zawierają `JsonPro-`

property z nazwą pod jaką występują w danych. Dzięki temu, metoda *JsonConvert.DeserializeObject* może sparsować dane w formacie JSON i utworzyć odpowiednie klasy.

7.2.4. Modele i modele widoków

Modele

1. *ApiManager* – odpowiedzialny za zapytania do API, pobiera dane w formacie JSON i zwraca w postaci listy modeli *Mop*
2. *AppDataStorage* – istnieje jeden obiekt tej klasy, zapisany w aplikacji, można traktować go jako zbiór zmiennych globalnych. Wartości są przechowywane na serwerze, czyli ich wartość jest taka sama dla wszystkich użytkowników. Przechowuje zapisane dane z API oraz czas ostatniego zapytania.
3. *Coordinates* – dla obu współrzędnych przechowuje ich wartość w formacie *double* oraz w formacie *string* w dwóch wersjach – zaokrągloną do 4 miejsc po przecinku (do wyświetlenia) oraz z pełną dokładnością (do używania w JavaScript).
4. *CoordinatesParser* – zawiera obie współrzędne w formacie *double*, używane tylko w parsowaniu danych z API
5. *FacilitiesConfig* – słownik, który dla kodu symbolizującego usługę na MOP-ie, zwraca obiekt klasy *Facility*
6. *FacilitiesParser* – zawiera pola *bool* dla każdej usługi, używana tylko w parsowaniu danych z API
7. *Facility* – zawiera dwie informacje: polską nazwę usługi oraz nazwę ikony
8. *Mop* – przechowuje wszystkie informacje o MOP-ie; zawiera funkcję *DeserializeMops*, która dla formatu JSON zwraca listę obiektów klasy *Mop*
9. *Operator* – zawiera pola dla nazwy operatora i kontaktu do niego
10. *SpacesCount* – zawiera trzy pola *int* (dla każdego z typów pojazdów)
11. *SpacesUsage* – dla każdego z typów pojazdów zawiera pole klasy *Usage*
12. *Usage* – zawiera zajętość miejsc parkingowych w % (w formacie *double* i *string*) oraz odpowiadające tej wartości: kolor tła i tekstu.

Modele widoków

1. *DetailsViewModel* – zawiera *MopViewModel* dla wybranego MOP-a, niezmienny obiekt *FacilitiesConfig* oraz cztery słowniki z odpowiednimi kolorami ikon i tekstu w zależności od obecności usługi
2. *MapViewModel* – zawiera obiekt klasy *MopListViewModel* z informacjami o wszystkich MOP-ach, aktualny typ pojazdu, dla którego mają być wyświetlane zajętości oraz HTML-owe klasy dla wszystkich trzech przycisków typów pojazdów
3. *MopListViewModel* – przerabia listę obiektów klasy *Mop* na listę obiektów klasy *MopViewModel*

4. *MopViewModel* – obudowuje informacje zawarte w klasie *Mop* i przygotowuje je do wyświetlania w widokach
5. *SearchViewModel* – zawiera m.in informacje o wszystkich MOP-ach, aktualnie wyszukiwane hasło, aktualny stan zaznaczonych ikon w filtrach

7.2.5. Kontrolery i widoki

Dla każdego kontrolera (plik **Controller.cs*) istnieje plik z HTML (**.cshtml*)

1. *Details*
2. *Map*
3. *Search*

7.2.6. Użyte biblioteki

1. OpenLayers

<https://openlayers.org>

Biblioteka napisana w języku JavaScript, umożliwiająca dodawanie dynamicznych map na stronach internetowych.

2. Material Design Lite

<https://getmdl.io>

Biblioteka CSS udostępniająca gotowe style podstawowych elementów HTML. Gwarantuje czysty, spójny wygląd strony internetowej.

3. ProgressBar.js

<https://github.com/kimmobrunfeldt/progressbar.js>

Biblioteka JavaScript do tworzenia animowanych wykresów kołowych. Wykorzystana do rysowania wykresów zajętości MOP-ów.

Rozdział 8

Organizacja pracy

Nasz projekt składa z sie z pięciu częściowo niezależnych programów. Niektóre z nich komunikują lub łączą się ze sobą, ale duża część kodu mogła być napisana bez znajomości szczegółowej implementacji pozostałych aplikacji. Dlatego powstał u nas dość wyraźny podział pracy – każdy program posiadał swojego koordynatora, który był autorem większości kodu i nadzorował rozwój aplikacji. Wszystkie zmiany w kodzie były recenzowane przez co najmniej jednego członka zespołu.

Dokładny podział prac został przedstawiony w następnym rozdziale.

Slack (komunikacja)

Do komunikacji wewnętrz zespołu używaliśmy komunikatora Slack. Daje on możliwość utworzenia wielu kanałów - publicznych i prywatnych, dzięki czemu mogliśmy łatwo rozdzielić rozmowy na poszczególne tematy. Slack umożliwia także przesyłanie plików, do których łatwo dotrzeć w przyszłości. Komunikator można zintegrować z innymi serwisami, np. GitHubem, z którego można dostawać powiadomienia o zmianach w repozytorium.

GitHub (repozytorium)

Kody źródłowe naszych aplikacji przechowywaliśmy na GitHubie. Utworzyliśmy tam zespół *mopsy-team*[13], w którym powstało sześć repozytoriów:

1. Mopnik
2. Mopsim
3. Mopsik-Server
4. Mopsik-Mobile
5. Mopsik-WWW
6. dokumentacja (m.in. praca licencjacka)

Todoist (menadżer zadań)

Todoist to menadżer zadań, w którym można utworzyć wiele projektów i podprojektów. Wewnątrz każdego z nich można wyznaczać zadania o różnym priorytecie i hierarchii. Każde zadanie można przydzielić jednemu z użytkowników, ustawić termin w jakim ma być wykonane, dodać etykiety i komentować. Aplikacja umożliwia wysyłanie powiadomień o nadchodzących terminach i wykonanych zadaniach.

Rozdział 9

Podział zadań

Mopnik

1. Magdalena Grabowska
2. Przemysław Perkowski - integracja z *Mopsimem*
3. Michał Kukuła - integracja z *Mopsimem*
4. Klaudia Laks - poprawki

Mopsim

1. Michał Kukuła
2. Przemysław Perkowski - generowanie planów podróży

Mopsik - serwer

1. Przemysław Perkowski

Mopsik - aplikacja mobilna

1. Klaudia Laks
2. Przemysław Perkowski - testowanie, poprawki

Mopsik - strona internetowa

1. Klaudia Laks

Dokumentacja / Praca licencjacka

1. Magdalena Grabowska
2. Michał Kukuła
3. Klaudia Laks
4. Przemysław Perkowski

Rozdział 10

Napotkane problemy i niezrealizowane pomysły

Przystępując do zadania, nasza wiedza z zakresu transportu drogowego, w szczególności ciężarowego, była nikła. Podczas realizacji projektu, wdrażając się w tematykę ruchu na autostradach i drogach szybkiego ruchu, dopiero po pewnym czasie byliśmy w stanie zrozumieć dokładnie jakie funkcjonalności realnie może zapewnić tworzone przez nas oprogramowanie.

W trakcie pisania programów napotkaliśmy wiele problemów oraz wraz z upływem czasu i postępem prac musieliśmy weryfikować, początkowo bardzo ambitny, plan zadań, które udało nam się wykonać.

10.1. Napotkane problemy

Główne problemy, które napotkaliśmy, możemy podzielić na kilka podstawowych kategorii:

1. Problemy z danymi
2. Problemy z komunikacją
3. Problemy techniczne

Opiszemy w skrócie każdy z powyższych problemów:

Problemy z danymi i komunikacją w zespole

W ramach projektu RID współpracuje wiele osób, które są odpowiedzialne za różne elementy projektu. Dane wymagane do działania naszych programów były pozyskiwane z wielu źródeł, za pośrednictwem wielu osób. Dane te często okazywały się niekompletne lub niewystarczające do naszych zastosowań. Jednym z przykładów są dane o długości postoju na MOP-ach. Były one zbierane między innymi z:

- systemu viaTOLL, za pomocą którego były generowane rozkłady czasów przejazdu pomiędzy parą punktów kontroli
- ankiet przeprowadzanych na MOP-ach.

Okazało się, że otrzymane zbiory danych nie były wystarczające do wiarygodnego wyznaczenia rozkładów czasu podróży. Jedną z czynności, które podjęliśmy w celu uzupełnienia tych danych było stworzenie ankiet internetowych i zamieszczenie ich w portalu Facebook

na grupach poświęconych kierowcom samochodów ciężarowych. Na podstawie zebranych danych wyznaczyliśmy pewne rozkłady czasu podróży dla różnych typów pojazdów. Wydaje się, że w przypadku posiadania dokładniejszych danych, dałoby się te rozkłady wyznaczyć jeszcze dokładniej. Innym przykładem są dane o współrzędnych MOP-ów oraz ilości miejsc na nich. Dane te, otrzymane przez nas na początku roku akademickiego okazały się posiadać liczne błędy i braki. Informacje o tych błędach przesyłałyśmy mailowo naszym zleceniodawcom. Niestety do czasu zakończenia pisania pracy (czerwiec 2018) nie udało nam się uzyskać poprawionych danych.

Problemy techniczne

Podczas pracy nad dużym projektem zawsze zdarzają się także problemy techniczne. Nie inaczej było i tym razem. Jednym z przykładów jest praca z React Native, podczas której korzystaliśmy z wielu zewnętrznych bibliotek. Czasem okazywało się, że kolejne wersje tych bibliotek, pobierane z Internetu podczas pracy nad programem zawierały nowe błędy lub luki bezpieczeństwa.

10.2. Niezrealizowane pomysły

Z wyżej wymienionych powodów, nie wszystkie planowane początkowo funkcjonalności udało się wdrożyć do naszych programów.

Preidykcje krótkoterminowe

Planowaliśmy zaangażować się w analizę danych o zajętości MOP-ów we współpracy z innymi członkami zespołu RID. Na tej podstawie nasza aplikacja mobilna mogłaby nie tylko pokazywać stan zajętości parkingów „na żywo”, ale także prognozowaną zajętość za 15, 30, 60 minut. Do analizy tych danych chcieliśmy wykorzystać algorytmy uczenia maszynowego, które dobrze działają przy analizowaniu odcinków czasu, jak np. rekurencyjne sieci neuronowe, czy LSTM. W związku z niewystarczającymi lub trudnymi do obrobienia danymi, nie zdecydowaliśmy się wprowadzać takiej możliwości.

Rozszerzona interakcja użytkowników z aplikacją mobilną

W końcowej fazie tworzenia projektu wpadliśmy na pomysł, jak rozszerzyć aplikację, aby była ona bardziej przydatna dla użytkowników. Można to zrobić w oparciu o crowdsourcing (czyli zaangażowanie dużej grupy ludzi – użytkowników aplikacji do jakiegoś działania – dzielenia się informacjami na temat parkingów). Przykładowe funkcjonalności, które sprawiałyby, że użytkownicy pomagaliby sobie wzajemnie byłyby np.

- ocenianie danego MOP-a poprzez przyznanie mu odpowiedniej liczby gwiazdek
- podanie jaka konkretnie restauracja znajduje się na danym MOP-ie
- możliwość dodawania komentarzy do MOP-ów

Jeśli pomyślimy o aplikacji w szerszym kontekście, to można byłoby np. dodać też opcję dodawania parkingów niebędących MOP-ami. Nie zdecydowaliśmy się na wprowadzenie takich usprawnień ze względu na duży czas potrzebny do dobrego wykonania takiego zadania, biorąc pod uwagę, że do dopracowania pozostało nam wiele innych funkcjonalności o większym priorytecie.

Precyzyjne dodawanie planowanych dróg do programu Mopnik

W początkowych fazach projektu planowaliśmy, aby można było nanieść projekt planowanej drogi w ustalonym formacie na mapę w Mopniku. Okazało się, że jest to trudne:

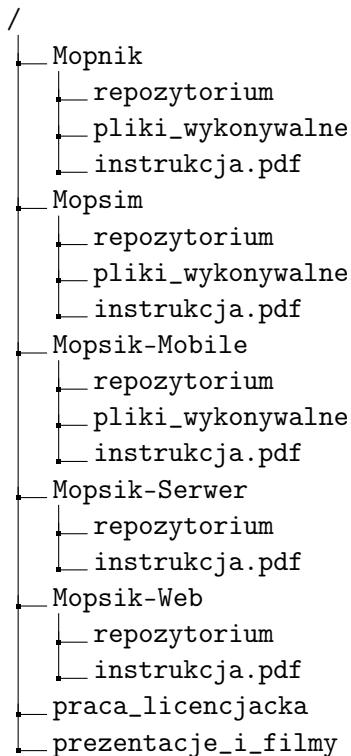
- W początkowych fazach planowania GDDKiA nie posiada dokładnych planów dróg.
- Aktualne, udostępnione nam plany dróg są w formacie, którego użycie w naszym programie (przy obecnej implementacji wykorzystującej siatkę drogową OSM) byłoby bardzo trudne.

Jako rozwiązanie tego problemu wprowadziliśmy opcję dodawania prostych odcinków pomiędzy punktami A i B na mapie. Ta opcja okazała się o wiele mniej problematyczna w implementacji, a spełnia swój podstawowy cel. Dla podanej długości drogi jesteśmy w stanie poprawnie wyznaczyć liczbę potrzebnych miejsc parkingowych oraz do dodanego odcinka możemy przypisać MOP-a. Można także uruchomić symulację na zmodyfikowanej sieci drogowej.

Rozdział 11

Zawartość płyty

Na płycie znajdują się katalogi dla pięciu programów, które stworzyliśmy. W każdym z tych katalogów znajduje się folder *repozytorium*. Z kolei w każdym takim folderze znajduje się plik *README.md*, który opisuje w jaki sposób należy skompilować kod źródłowy oraz uruchomić program na systemie Linux. Dodatkowo każdy program zawiera dołączony plik *instrukcja.pdf*, w którym znajduje się krótkie wyjaśnienie jak należy korzystać z danej aplikacji. Programy Mopnik i Mopsim oraz Mopsik-Mobile zawierają także folder *pliki_wykonywalne*. W przypadku programów Mopnik i Mopsim zawiera on plik wykonywalny o rozszerzeniu .jar, który umożliwia uruchomienie go na dowolnym systemie operacyjnym zainstalowaną Javą 8. Dla aplikacji mobilej folder ten zawiera plik .apk, który pozwala na bezpośrednie zainstalowanie aplikacji na urządzeniu z systemem Android w wersji co najmniej 4.1.



Rysunek 11.1: Struktura plików umieszczonych na załączonej płycie DVD

Bibliografia

- [1] Wikipedia, *Program budowy w III RP*,
https://pl.wikipedia.org/wiki/Autostrady_i_drogi_ekspresowe_w_Polsce
- [2] GDDKiA, *Generalna Dyrekcja Dróg Krajowych i Autostrad – Serwis Informacyjny*,
<https://www.gddkia.gov.pl/pl/963/miejsca-obslugi-podroznych-mop>
- [3] Melnik, Roderick. “Mathematical and Computational Modeling: With Applications in Natural and Social Sciences, Engineering, and the Arts. Wiley.” ISBN 978-1-118-85398-6., 2015
- [4] Gustafsson, Leif; Sternad, Mikael. “Bringing consistency to simulation of population models: Poisson Simulation as a bridge between micro and macro simulation”. Mathematical Biosciences. 209: 361–385, 2007
- [5] Niazi, Muaz; Hussain, Amir. “Agent-based Computing from Multi-agent Systems to Agent-Based Models: A Visual Survey”, 2011
- [6] Wikipedia, *System wieloagentowy*,
https://pl.wikipedia.org/wiki/System_wieloagentowy
- [7] Horni, A, Nagel, K and Axhausen, K W. 2016. “Introducing MATSim.” In: Horni, A, Nagel, K and Axhausen, K W. (eds.) “The Multi-Agent Transport Simulation MATSim”, Pp. 3–8. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw>
License: CC-BY 4.0
- [8] React Native, *Dokumentacja*,
<https://facebook.github.io/react-native/>
- [9] Developers Google, *FAQ – Google Maps API*,
<https://developers.google.com/maps/faq>
- [10] Projekt mający na celu stworzenie darmowej mapy kuli ziemskiej. <https://www.openstreetmap.org/>
- [11] M. Spławińska, K. Solecka,
http://www.autobusy-test.com.pl/images/stories/Do_pobrania/2017/nr%2012/Bezp%20i%20ekol/073_013_A_BiE_SPLAWINSKA_SOLECKA.pdf
- [12] GDDKiA, *Dane o MOP-ach*,
<https://www.gddkia.gov.pl/pl/24/pliki-do-pobrania>
- [13] GitHub, *mopsy-team*
<https://github.com/mopsy-team>

- [14] Układ współrzędnych 92 w bazie *Spatial Reference*
<http://www.spatialreference.org/ref/epsg/2180/>