

Dokumentacja projektu
‘Znajdowanie optymalnego miejsca parkingowego
w oparciu w ważony MaxSat Solver’

Studio Projektowe I

Twórcy : Bartosz Biegun , Paweł Hanzlik

Opiekun: prof. Radosław Klimek



27.06.2021

Spis treści

1.	Cel projektu	3
2.	Zasada działania	3
a.	Strefy	3
b.	Solver	5
3.	Architektura systemu	6
4.	Schemat bazy danych	7
5.	Przykłady rekordów w tabelach	8
6.	Znaczenie poszczególnych pól w tabelach	9
a.	Tabela Strefy:.....	9
b.	Tabela Kierowcy:	9
c.	Tabela Parkingi:	9
7.	Diagram klas	10
8.	Wykorzystane technologie	11
9.	Generowanie klauzul z dostępnych danych	11
a.	Określenie zmiennych zdaniowych	11
b.	Określenie klauzul	12
c.	Interpretacja wyników.....	15
10.	Biblioteka SAT4J	15
a.	Format danych wejściowych	15
11.	Jak działa nasz solver	16
12.	Przykłady działania programu	17
a.	Generowanie danych.....	17
b.	Wykonanie zapytania	17
c.	Przetworzenie zapytania i wywołanie Solvera	17
d.	Przykład nr 1	18
e.	Przykład nr 2	21
f.	Przykład nr 3	24
g.	Przykład nr 4	26

1. Cel projektu

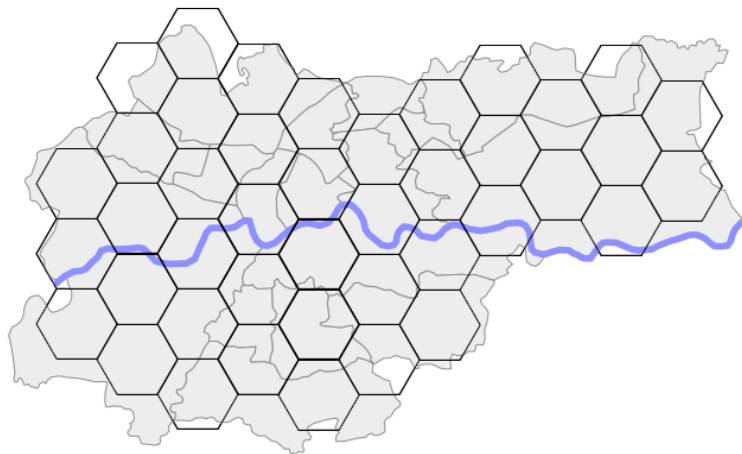
Celem projektu jest zaimplementowanie aplikacji wykorzystującej solver typu MaxSat do znajdowania miejsc parkingowych dla klientów wypożyczalni samochodów z uwzględnieniem popytu na następne wypożyczenia w danym sektorze miasta.

Aplikacja będzie symulowała obszar danego miasta podzielonego na wyznaczone strefy, wewnątrz których jest pewna liczba samochodów dostępnych do wypożyczenia. Każda strefa będzie mieć przewidywany popyt na samochody, na który aplikacja będzie odpowiadać przekierowując samochody do danej strefy gdy są tam potrzebne. Użytkownik będzie posiadał możliwość utworzenia zapytania odpytującego serwer o miejsce parkingowe w pobliżu pewnej lokacji lub uruchomić prostą symulację generującą wiele podobnych zapytań oraz modyfikującą stan bazy w zależności od pory dnia.

2. Zasada działania

a. Strefy

Miasto zostanie podzielone na heksagonalne strefy w celu zarządzania popytem i podażą samochodów na zróżnicowanym obszarze.



W celu optymalnego rozlokowania klientów dla każdej strefy jest regularnie obliczany współczynnik zajętości:

$$a_n = \frac{S_n}{D_n}$$

gdzie:

a_n - Współczynnik zajętości

S_n - Suma wag klientów chcących wypożyczyć samochód w odległości do 1 km od środka strefy

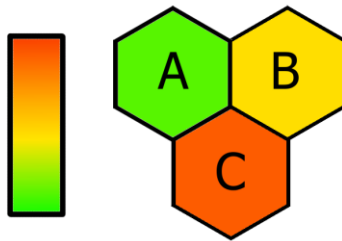
D_n - Suma wag wolnych samochodów w odległości do 1 km od środka strefy

Wagi klientów i miejsc parkingowych maleją wraz z odległością od środka strefy co odwzorowuje możliwość że klienci na skraju strefy zostaną przypisani do miejsc parkingowych w sąsiadującej strefie

Aplikacja będzie manipulować klientami chcącymi zaparkować w pobliżu strefy a przez to sumą wag wolnych samochodów aby utrzymać współczynnik zajętości zbliżony do 1.

Ważnym czynnikiem będzie również wskaźnik atrakcyjności strefy, który także wpłynie na rezultat pracy solvera.

Na przykład:



Samochody ze strefy A (O niskim współczynniku zajętości – samochodów jest za dużo) będą przekierowywane do strefy C (O wysokim współczynniku zajętości – samochodów brakuje)

b. Solver

Problem spełnialności to koncept związany z logiką matematyczną, zostanie on wykorzystany do rozwiązania problemu zarządzania wypożyczanymi samochodami w Krakowie.

Z problemem SAT mamy do czynienia gdy mając formułę zdaniową chce się określić, czy istnieje podstawienie wartościami '0' i '1' pod zmienne zdaniowe, by formuła była spełniona.

Problemy Max-Sat składają się z ważonych klauzul połączonych koniunkcjami :

$$(\neg p_1 \vee \neg p_2) \wedge (\neg p_1 \vee p_3) \wedge (\neg p_1 \vee \neg p_3)$$

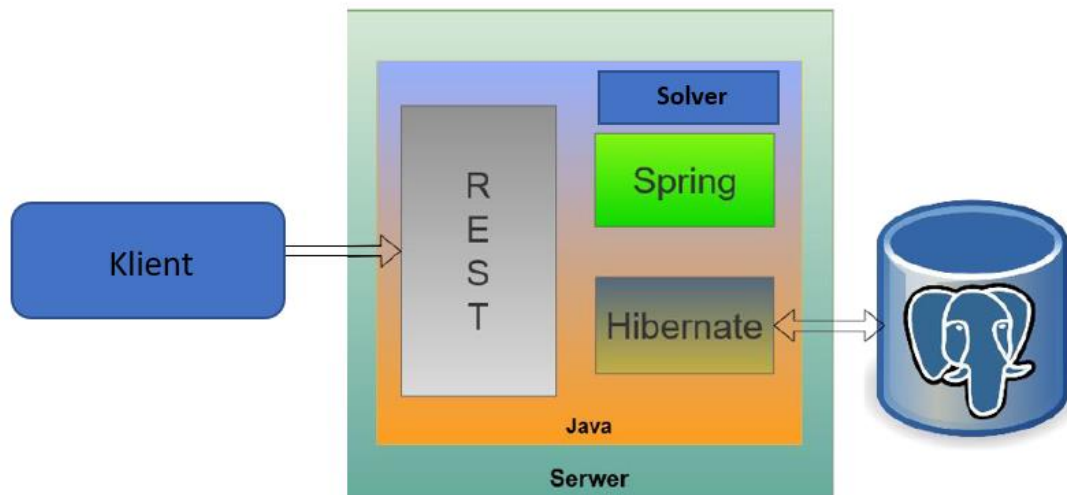
Ponieważ nie zawsze da się spełnić wszystkie klauzule, solver Max Sat znajduje rozwiązanie w którym największa liczba klauzul jest spełniona

Z problemem Max-SAT to rozszerzenie problemu SAT w taki sposób, aby w przypadku gdy nie da się dobrać wartości tak, aby spełniona była cała formuła dobiera się je tak, by zmaksymalizować ilość spełnionych formuł.

Ważony Max-SAT to kolejne rozszerzenie, dodające tym razem odpowiadające wagi każdej z klauzul i uwzględnienie ich a procesie rozwiązywania w taki sposób aby suma wag niespełnionych klauzul była jak najmniejsza.

Aby dobrać najlepsze dla systemu rozwiązanie będzie on generować formułę w której zdania będą odpowiadać optymalnemu rozlokowaniu samochodów.

3. Architektura systemu



Główny moduł aplikacji zostanie napisany w języku Java z użyciem Spring Framework. Baza danych działać będzie na serwerze PostgreSQL, natomiast łączenie jej z projektem realizowane będzie przy użyciu Hibernate. Z zewnątrz klient wysyłać będzie zapytanie obsługiwane przez REST API, które na podstawie danych z bazy oraz obliczeń Solvera zwróci wynik.

4. Schemat bazy danych

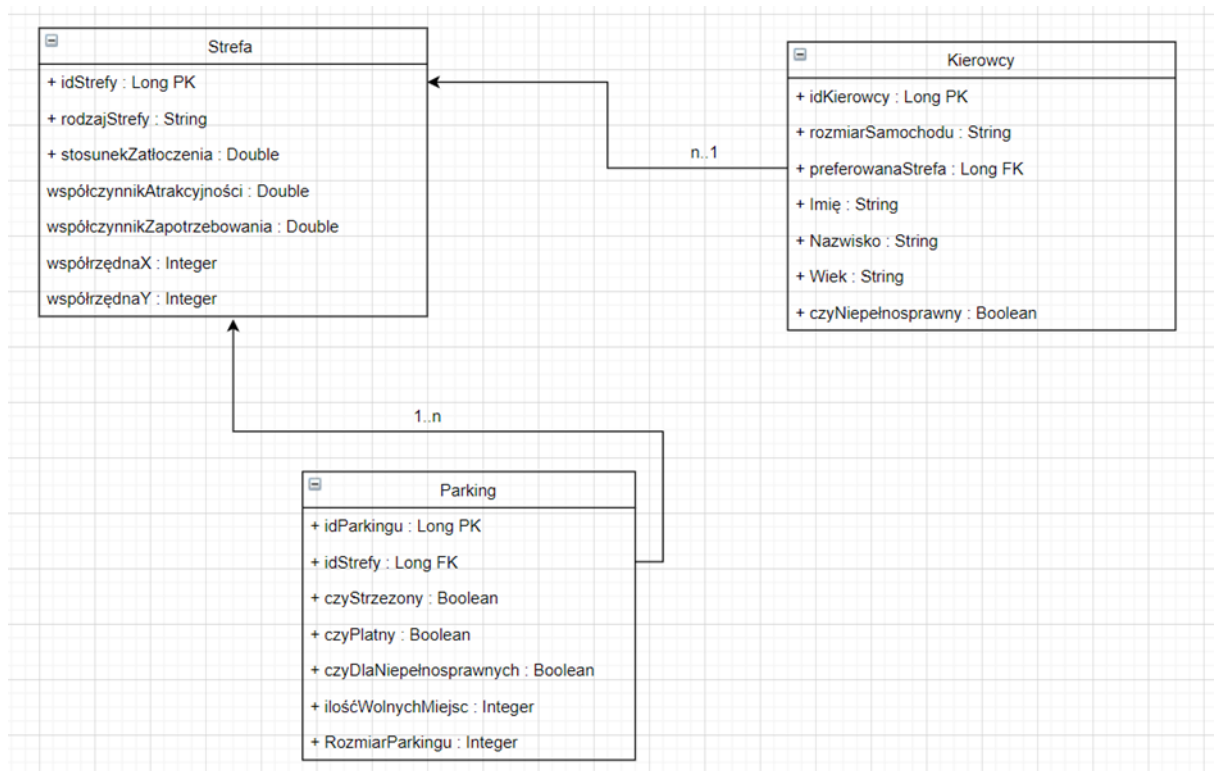


Tabela Strefa oznacza podział miasta na obszary posiadające wartość zatłoczenia w danym momencie, współczynnik atrakcyjności, współczynnik zapotrzebowania oraz rodzaj Strefy tj. obrzeża, centrum, poza miastem, przemysłowa, a także jej współrzędne. Zawiera wiele rekordów typu Parking.

Tabela Kierowcy zawiera dane klienta, informacje o rozmiarach ich samochodów oraz preferencjach dotyczących miejsca parkingowego.

Tabela Parking zawiera informacje dotyczące parkingu w danej strefie miasta.

5. Przykłady rekordów w tabelach

Tabela: strefy

idStrefy	rodzajStrefy	WspółczynnikZatloczenia	WspółczynnikAtrakcyjności	WspółczynnikZapotrzebowania	WspółrzędnaX	WspółrzędnaY
1	centrum	0.7	0.5	0.3	-4	-3
2	centrum	0.8	0.5	0.5	2	1
3	obrzeża	0.5	0.3	0.4	-1	2
4	obrzeża	0.6	0.7	0.7	1	0
5	poza miastem	0.3	0.2	0.4	0	-2
6	poza miastem	0.2	0.4	0.5	3	1

Tabela: kierowcy

idKierowcy	rozmiarSamochodu	preferowanaStrefa	Imię	Nazwisko	Wiek	czyNiepełnosprawny
1	mały		2 Anna	Nowak	35	nie
2	średni		4 Krzysztof	Kowalski	45	tak
3	mały		1 Michał	Kowalczyk	65	tak
4	duży		6 Jan	Nowakowski	24	nie
5	duży		3 Wojciech	Kwiatkowski	38	nie

Tabela: parkingi

idParkingu	idStrefy	czyStrzeżony	czyPłatny	czyDlaNiepełnosprawnych	ilośćWolnychMiejsc	RozmiarParkingu
1	4	tak	tak	tak	50	3
2	3	nie	tak	nie	45	4
3	1	nie	nie	tak	16	1
4	3	tak	nie	nie	47	2
5	6	tak	nie	tak	38	3

6. Znaczenie poszczególnych pól w tabelach

a. Tabela Strefy:

idStrefy – klucz podstawowy tabeli

rodzaj strefy – określenie położenia strefy na ‘mapie’. Będzie on mógł być wykorzystany do bardziej inteligentnego generowania wartości zapotrzebowania oraz zatłoczenia

współczynnik zatłoczenia – określa nam jak bardzo zatłoczona jest strefa, tj. ile wolnych, czekających na klienta samochodów jest w danej strefie. Będziemy starali się jak najbardziej dopasowywać te wskaźniki pomiędzy strefami, tak aby nie było w nich za dużo, ani za mało samochodów. Dlatego więc, gdy współczynnik zapotrzebowania będzie dużo większy niż współczynnik zatłoczenia będziemy starali się kierować tam klientów w celu parkowania nieużytkowanego już przez niego auta.

współczynnik atrakcyjności – określa jak bardzo przyjazna dla użytkownika jest to strefa. Wpływać na to będzie kilka czynników takich jak: ilość i rodzaj parkingów w obrębie tej strefy oraz poszczególne ich parametry.

współczynnik zapotrzebowania – określa jak dużo klientów w danej strefie szuka samochodu do wynajęcia. Będziemy starali się właśnie w te strefy z najwyższym współczynnikiem kierować osoby parkujące swoje auta.

współrzędnaX – pierwsza współrzędna położenia strefy na mapie miasta

współrzędnaY – druga współrzędna położenia strefy na mapie miasta

b. Tabela Kierowcy:

idKierowcy – klucz podstawowy tabeli

rozmiar samochodu – wielkość auta, którym porusza się użytkownik. Może on posłużyć do urozmaicenia formuły naszego solvera, tak aby kierował go na taki parking w obrębie strefy, który jest dostosowany do rozmiaru jego auta.

preferowana strefa – strefa, którą użytkownik uważa za najodpowiedniejszą dla niego, solver będzie starał się dopasować parkingi, tak aby były jak najbliżej jego strefy.

imię, nazwisko, wiek – dane użytkownika

niepełnosprawność – określa czy kierowca ma orzeczenie o stopniu niepełnosprawności, solver przydzieli mu najlepszy parking posiadający

c. Tabela Parkingi:

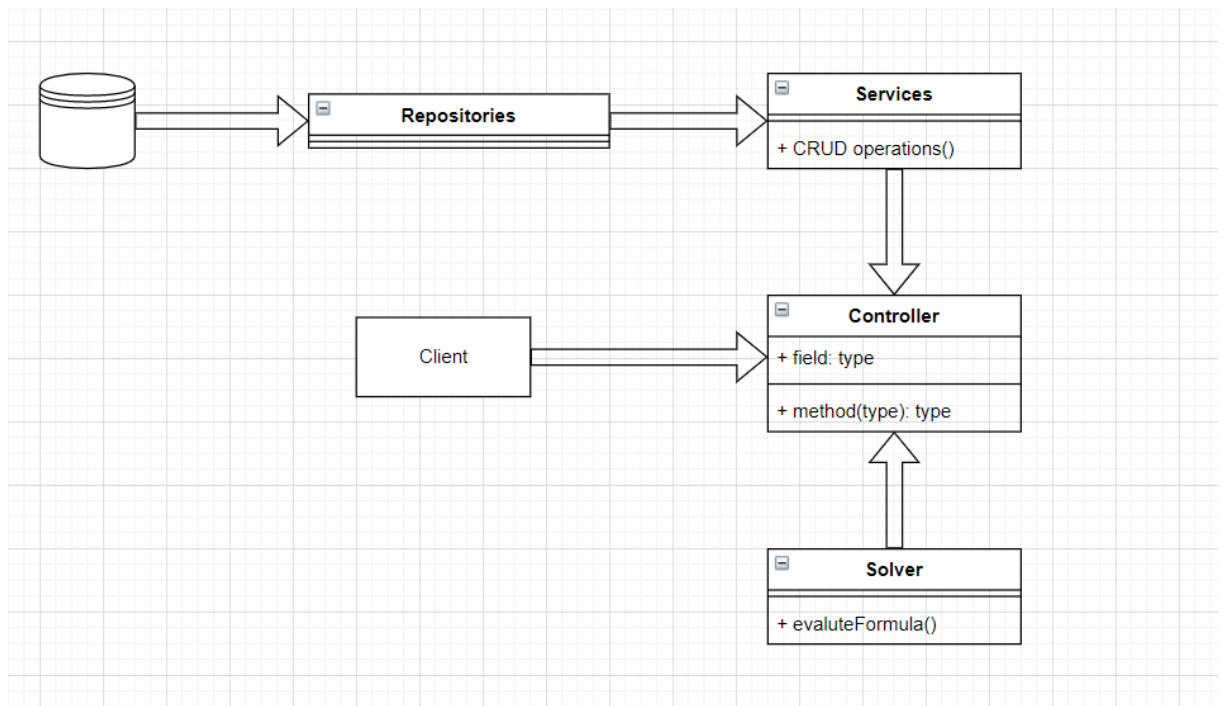
idParkingu – klucz podstawowy tabeli

idStrefy – strefa, w której znajduje się parking

strzeżony, płatny, dla niepełnosprawnych, rozmiar parkingu – parametry parkingu, które użytkownik będzie mógł wybrać w swoim zapytaniu

ilość wolnych miejsc – brane pod uwagę będą tylko te parkingi, które mają jeszcze jakieś wolne miejsca postojowe

7. Diagram klas



Sekcja **Repositories** to interfejsy dziedziczące po **JpaRepository**, realizujące połączenie z bazą. Serwisy będą przechowywały metody umożliwiające pobieranie, dodawanie czy też modyfikowanie oraz usuwanie danych z bazy. Solver będzie to zbiór klas, które implementują algorytm Max-Sat solver i na podstawie zapytania klienta zwróci najlepsze miejsce parkingowe. Kontroler będzie przetwarzał zapytanie, przekazywał do solvera i zwracał odpowiedź klientowi.

8. Wykorzystane technologie

- **Java**

Będziemy używać języka Java w wersji 15

- **Spring**

Użyjemy SpringBoot'a w wersji 2.4.

- **PostgreSQL**

Do zarządzania bazą danych. Wersja 13.

- **Hibernate**

Czyli framework realizujący dostęp do danych w bazie. Podstawową rzeczą jaką będziemy wykorzystywać to mapowanie O-R.

- **Lombok**

Biblioteka Javy udostępniająca adnotacje do tworzenia metod, głównie getterów i setterów.

9. Generowanie klauzul z dostępnych danych

Do przetwarzania danych za pomocą algorytmu Weighted Max-Sat wymagane jest przekształcenie je na zmienne zdaniowe a następnie przyporządkowanie do klauzul.

Każdej klauzuli przypisywana jest waga określająca jak ważne jest spełnienie klauzuli.

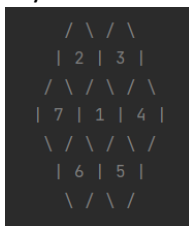
Każde dostępne miejsce parkingowe zostaje porównane z klauzulą i przedstawione użytkownikowi. Lista dostępnych miejsc zostaje posortowana po ilości spełnionych klauzul tak, aby najbardziej dopasowane miejsca znalazły się na górze listy.

a. Określenie zmiennych zdaniowych

Na podstawie bazy danych wyróżnimy 6 zmiennych zdaniowych:

- S1 – Parking znajduje się w strefie 1
- S2 – Parking znajduje się w strefie 2
- S3 – Parking znajduje się w strefie 3
- S4 – Parking znajduje się w strefie 4
- S5 – Parking znajduje się w strefie 5
- S6 – Parking znajduje się w strefie 6
- S7 – Parking znajduje się w strefie 7
- S8 – Parking znajduje się w preferowanej strefie klienta
- S9 – Parking ma więcej niż 10 wolnych miejsc
- S10 – Parking jest strzeżony
- S11 – Parking jest płatny
- S12 – Parking jest przystosowany dla osób niepełnosprawnych
- S13 – Rozmiar parkingu > 5

Rozpatrujemy tylko 7 stref znajdujących się dookoła pozycji użytkownika. Strefa nr 1 to ta, w której znajduje się użytkownik.



Przykład:

```
// S1 = 0
// S2 = 0
// S3 = 0
// S4 = 1
// S5 = 0
// S6 = 0
// S7 = 0
// S8 = 0
// S9 = 1
// S10 = 1
// S11 = 0
// S12 = 1
// S13 = 0

// [S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13] = [0,0,0,1,0,0,0,0,0,1,1,0,1,0]
```

Takie ułożenie zmiennych zdaniowych zawiera informację o tym, że parking znajduje się w strefie 4, która nie jest jednak preferowaną strefą użytkownika oraz ma więcej niż 10 wolnych miejsc, dodatkowo jest strzeżony, a także przystosowany dla osób niepełnosprawnych lecz nie jest płatny, ani nie jest to bardzo duży parking.

b. Określenie klauzul

Analizując dane o kliencie możemy oszacować jego preferencje i dopasować do nich zmienne zdaniowe. Wagi klauzul związanych z atrakcyjnością zostały odpowiednio pomniejszone, aby bardziej liczyło się zapotrzebowanie na samochody niż wygoda klienta.

- U0 – Klauzula pomocnicza wymuszająca wybranie przynajmniej jednej strefy, w której klient może zaparkować.
- U1 – Nie wybieraj strefy nr 1
- U2 – Nie wybieraj strefy nr 2
- U3 – Nie wybieraj strefy nr 3
- U4 – Nie wybieraj strefy nr 4
- U5 – Nie wybieraj strefy nr 5
- U6 – Nie wybieraj strefy nr 6
- U7 – Nie wybieraj strefy nr 7
- U8 – Preferowana strefa klienta to ta sama strefa, którą wskazał solver
- U9 – Klient jest niepełnosprawny
- U10 – Rozmiar samochodu klienta > 5
- U11 – Klient jest skąpy
- U12 – Klient dba o wygodę parkowania

U0	->	U1 v U2 v U3 v U4 v U5 v U6 v U7	(Infinity)
U1	->	S1	(Waga obliczana dynamicznie)
U2	->	S2	(Waga obliczana dynamicznie)
U3	->	S3	(Waga obliczana dynamicznie)
U4	->	S4	(Waga obliczana dynamicznie)
U5	->	S5	(Waga obliczana dynamicznie)
U6	->	S6	(Waga obliczana dynamicznie)
U7	->	S7	(Waga obliczana dynamicznie)
U8	->	S8	(15)

Klauzula U0 jest tworem sztucznym, stworzonym tylko po to, żeby solver wybrał przynajmniej jedną strefę, dlatego jej waga to nieskończoność.









Waga obliczana dynamicznie oznacza:

$$\text{Math.round}((\text{occupiedRatio} - \text{requestRatio}) * 100),$$

gdzie occupiedRatio to współczynnik zajętości, a requestRatio to współczynnik zapotrzebowania. Oznacza to, że im większe zapotrzebowanie na samochody w danej strefie, tym większą wagę otrzyma klauzula chcąc wybrać tą strefę jako tą, w której klient chce zaparkować.

W przyszłości uwzględnimy jeszcze współczynnik atrakcyjności.

Powyższe klauzule są niezależne od użytkownika, ponieważ to my narzucamy, by kierujący w pierwszej kolejności zostawił samochód w strefie, w której my chcemy, gdyż brakuje tam aut dla innych wypożyczających. Mimo wszystko delikatnie idziemy użytkownikowi na rękę i lekko faworyzujemy jego strefę obniżając wagę odpowiedniej klauzuli.

U9		S12 ∧ (25) S9 v S13	U10		S10 ∧ (20) S13
U11		¬S11 ∧ (15) S9	U12		S10 v S9 ∧ (20) S8 v ¬S11
¬U9		¬S12 ∧ (15) ¬S13	¬U10		¬S10 ∧ (10) ¬S13 v ¬S9
¬U11		S11 ∧ (25) ¬S9 v S10	¬U12		¬S9 ∧ (10) ¬S8

$U9 \rightarrow S12 \wedge (S9 \vee S13)$ – Użytkownik posiada orzeczenie o niepełnosprawności, będzie więc chciał parkować na parkingach dostosowanych do niego oraz takich, które są duże i mniej oblegane.

$U10 \rightarrow S10 \wedge S13$ – Użytkownik prowadzi duży samochód, będzie więc chciał parkować na dużym, strzeżonym parkingu.

$U11 \rightarrow \neg S11 \wedge S9$ – Użytkownik jest skąpy, wybierze on więc niepłatny, mało oblegany parking.

$U12 \rightarrow (S10 \vee S9) \wedge (S8 \vee \neg S11)$ – Użytkownik chce parkować bezpiecznie i wygodnie, wybierze on więc strzeżony, obszerny parking, w dogodnej dla niego strefie, chyba, że parking nie będzie płatny.

$\neg U9 \rightarrow \neg S12 \wedge \neg S13$ – Użytkownik nie posiada orzeczenia o niepełnosprawności, będzie więc chciał parkować na parkingach bez stref dla niepełnosprawnych oraz nie będzie musiał mieć zapewnionego dużego parkingu.

$\neg U10 \rightarrow \neg S10 \wedge (\neg S13 \vee \neg S9)$ – Użytkownik nie prowadzi dużego samochód, nie będzie więc musiał parkować na dużym, strzeżonym parkingu, lub na mniej uczęszczanym.

$\neg U11 \rightarrow S11 \wedge (\neg S9 \vee S10)$ – Użytkownik nie jest skąpy, nie musi on więc wybierać niepłatnego, mało obleganego parking, pod warunkiem, że jest on strzeżony.

$\neg U12 \rightarrow \neg S9 \wedge \neg S8$ – Użytkownik nie musi parkować bezpiecznie i wygodnie, nie musi więc być to parking w strefie klienta lub być mało uczęszczany.

Przykład:

```
// U9 = 1
// U10 = 0
// U11 = 0
// U12 = 1
// [U9,U10,U11,U12] = [1,0,0,1]
```

Podane parametry przez użytkownika. Oznaczają one, że kierujący jest niepełnosprawny oraz ceni sobie wygodę parkowanie, nie kieruje on natomiast dużym samochodem, ani nie jest skąpy.

Podane klauzule dołączą w solverze do już obecnych tworząc formułę:

```
/* U0 => (S1 v S2 v S3 v S4 v S5 v S6 v S7)
   U9 => (S12 ^ (S9 v S13))
   ~U10 => (~S10 ^ (~S13 v ~S9))
   ~U11 -> (S11 ^ (~S9 v S10))
   U12 -> ((S10 v S9) ^ (S8 v ~S11))
*/

/* (S1 v S2 v S3 v S4 v S5 v S6 v S7) ^ (S12 ^ (S9 v S13)) ^ (~S10 ^ (~S13 v ~S9))
   (S11 ^ (~S9 v S10)) ^ ((S10 v S9) ^ (S8 v ~S11))
*/
```

c. Interpretacja wyników

Algorytm iteruje po wszystkich miejscach parkingowych i wyodrębnia te z największą ilością spełnionych klauzul, a następnie przedstawia je użytkownikowi jako listę. W tym momencie powinny to być głównie parkingi znajdujące się w strefie z najwyższą wagą, dopiero wtedy staramy się jak najlepiej dopasować parametry parkingu, tak aby spełniały one jak najwięcej klauzul U9-12

10. Biblioteka SAT4J

SAT4J to biblioteka open source stworzona dla języka Java w celu rozwiązywania problemów logicznych i optymalizacyjnych. Idealnie nadają się do obliczania zadań związanych z SAT oraz jego wariacjami. Użyte w niej wzorce projektowe takie jak strategia oraz dekorator sprawiają, iż można dopasować solver do swoich potrzeb. Mimo wszystko jest ona jednak stosunkowo wolna w działaniu co jest jej znacznym minusem.

a. Format danych wejściowych

Biblioteka wymaga od użytkownika specjalnego wejścia.

```
c
c WCNF w formacie DIMACS
c
p wcnf 2 3
3 1 -2 0
7 1 2 0
4 -1 2 0
```

Gdzie w linii p wcnf 2 3 trzeci argument to nvars czyli liczba zmiennych zdaniowych, a czwarty to nclauses czyli liczba klauzul. Następnie następuje przekazanie właściwych danych tj. nclauses linii zawierających wagę danej klauzuli, po której występuje nvars liczb z zakresu [-nvars; nvars] przy czym dodatnie liczby odpowiadają danej formule zdaniowej, a ujemna jej zaprzeczeniu. Każda linia kończy się liczbą 0.

11. Jak działa nasz solver

W pierwszym będziemy wybierali użytkownika z bazy, dla którego obliczony zostanie najbardziej optymalny parking, na którym będzie on mógł zaparkować. Wywołamy więc dla niego też Solver podając listę parametrów tj. listę wartości klauzul. Będzie ona wyglądała np. w taki sposób:

```
[U9, U10, U11, U12] = [0,0,1,1]
```

Następnie zostaną wygenerowane dla niego koordynaty, na których się on znajduje. Wtedy także wyszukane zostaną strefy otaczające użytkownika, wraz z tą, w której się znajduje. Dla nich zostanie wygenerowanych i dodanych 7 klauzul do Solvera, po jednej dla każdej strefy. Ich wagi będą uzależnione od parametrów danej strefy, tj. większą wagę dostaną strefy z większym współczynnikiem zapotrzebowania oraz współczynnikiem atrakcyjności, a mniejszą te z mniejszym współczynnikiem zapotrzebowania i większym współczynnikiem zajętości, jako, że nie potrzeba tam w tej chwili więcej samochodów.

Do stworzonych już klauzul dodane zostaną te, które podał użytkownik.

To co będzie się działo, to tak naprawdę w pierwszej kolejności wybór strefy z najwyższą wagą, a później dopasowanie parkingu o parametrach spełniających klauzule o najwyższych wagach.

Odpowiedzą solvera będzie lista zmiennych zdaniowych z dodatnim lub ujemnym współczynnikiem dla każdej zmiennej, oznaczającym czy taka zmienna zdaniowa powinna być spełniona, czy też nie. Taki zestaw danych określał będzie parametry idealnego parkingu.

```
[-1, -2, -3, -4, -5, -6, 7, -9, 10, -11, 12]
```

Jednakże taka reprezentacja nas nie interesuje. My bowiem chcemy uzyskać id tego parkingu, na którym klient powinien zaparkować. W tym celu wykonujemy naszą metodę test, obliczającą wartość Score, dla każdego parkingu. Jeśli znajduje się on w strefie wskazanej przez solver otrzymuje 10 punktów, a także po 1 punkcie za każdą kolejną spełnioną zmienną zdaniową. Program więc w końcowej fazie zwraca posortowaną listę parkingów oraz ich Score. Oczywiście najlepszy parking to ten z najwyższym Score.

```
[ParkingId: 27      Score: 12
, ParkingId: 25      Score: 3
, ParkingId: 28      Score: 3
, ParkingId: 32      Score: 3
, ParkingId: 34      Score: 3
, ParkingId: 45      Score: 3]
```


12. Przykłady działania programu

a. Generowanie danych

Metoda generująca dane pobiera 1 argument określający ilość generowanych stref. Następnie generowanie stref odbywa się w 2 zagnieżdżonych pętlach, tak aby ich współrzędne zawierały się w zakresie $[-ilość; ilość]$, zarówno dla x-owej oraz y-owej współrzędnej. Otrzymujemy więc $(2*ilość)^2$ wygenerowanych stref z losowymi współczynnikami zapotrzebowania oraz zajętości. Będzie to symulowało otrzymywanie rzeczywistych danych z innego źródła. Na podstawie parametru ilość generujemy także $10*ilość$ parkingów rozmieszczonych losowo we wcześniej wygenerowanych strefach oraz $10*ilość$ użytkowników. Wszelkie parametry tychże rekordów również są losowane.

b. Wykonanie zapytania

Zapytanie wykonujemy z użyciem programu Postman, lub zamiennie Swaggera. W parametrach zapytania podajemy id użytkownika, dla którego będziemy obliczali optymalny parking, a także jego współrzędne na mapie miasta oraz 2 parametry specyfikujące rodzaj parkingu preferowany przez klienta, tj. czy klient jest skąpy – faworyzując niepłatne parkingi, oraz czy klient dba o wygodę parkowania – faworyzując tym parkingi w większej ilości wolnych miejsc. Pozostałe parametry pobieramy z danych przechowywanych w rekordzie klienta w tabeli Użytkownicy.

c. Przetworzenie zapytania i wywołanie Solvera

Metoda kontrolera wybiera na podstawie podanych w parametrach współrzędnych użytkownika strefy przylegające do strefy klienta oraz ją samą. Następnie solver oblicza parametry najlepszego parkingu, a dalej metoda porównująca istniejące parkingi zwraca posortowaną listę.

d. Przykład nr 1

Generujemy 16 strefy, parkingi oraz użytkowników. Wykonujemy poniższe zapytanie:

czyDba_o_Wygone integer(\$int32) (query)	czyDba_o_Wygone <input type="text" value="0"/>
czySkapy integer(\$int32) (query)	czySkapy <input type="text" value="1"/>
Lat integer(\$int32) (query)	Lat <input type="text" value="0"/>
Lon integer(\$int32) (query)	Lon <input type="text" value="0"/>
userId integer(\$int32) (query)	userId <input type="text" value="63"/>

Jak widać, chcemy znaleźć optymalne miejsce parkingowe dla użytkownika o id 63, znajdującego się w strefie o współrzędnych (0,0), który wskazał iż szuka niepełnego parkingu, a także takiego, który nie musi mieć więcej niż 10 wolnych miejsc.

	user_id [PK] bigint	age integer	car_size integer	handicaped boolean	name character varying (255)	preferable_zone bigint	surname character varying (255)
5		61	65	6 true	Jack	4	aaa
6		62	22	2 false	Jack	3	aaa
7		63	63	7 true	Jack	10	aaa
8		64	39	1 false	Jack	12	aaa

Dodatkowo klient jest osobą niepełnosprawną, więc parking przystosowany dla takich osób otrzyma wyższą notę.

	zone_id [PK] bigint	cordx integer	cordy integer	attractiveness_ratio double precision	occupied_ratio double precision	request_ratio double precision	zone_type character varying (255)
1	1	-2	-2	0.28	0.14	0.03	test
2	2	-2	-1	0.45	0.58	0.26	test
3	3	-2	0	0.44	0.3	0.83	test
4	4	-2	1	0.57	0.73	0.48	test
5	5	-1	-2	0.16	0.85	0.69	test
6	6	-1	-1	1	0.7	0.64	test
7	7	-1	0	0.07	0.51	0.44	test
8	8	-1	1	0.37	0.58	0.86	test
9	9	0	-2	0.1	0.64	0	test
10	10	0	-1	0.35	0.35	0.34	test
11	11	0	0	0.29	0.12	0.18	test
12	12	0	1	0.21	0.42	0.18	test
13	13	1	-2	0.46	0.79	0.01	test
14	14	1	-1	0.87	0.33	0.84	test
15	15	1	0	0.88	0.64	0.43	test
16	16	1	1	0.79	0.08	0.96	test

Jak widać, klient znajduje się w strefie o id 11. Jego współczynnik zajętości jest bardzo niski, jednak w otoczeniu użytkownika znajduje się strefa o id 16 z jeszcze niższym współczynnikiem, a także o wiele wyższym współczynnikiem zapotrzebowania i atrakcyjności. Tam więc Solver powinien pokierować klienta.

Tabela priorytetów:

IdStrefy	Priorytet
7	11
8	10
10	11
11	11
12	12
15	12
16	7

Interesuje nas bowiem najniższy priorytet, gdyż strefy dostają ujemne wagi, więc mniejsza liczba dodatnia to większa ujemna.

Solver zwraca nam taki oto wynik:

```
-1 0
7
-1 1
8
0 -1
10
0 0
11
0 1
12
1 0
15
1 1
16
[-1, -2, -3, -4, -5, -6, 7, -9, 10, -11, 12]
```








Pierwsze ciągi wartości zostawione są na potrzeby lepszej wizualizacji pracy Solvera. Jak widać bierze on tylko strefy z otoczenia klienta tj. o id odpowiednio 7, 8, 10, 11, 12, 15 oraz 16. Następnie zwraca listę wartości zmiennych zdaniowych. Pierwsze 7 z nich to wskaźnik strefy, którą wskazał solver jako najlepszą. Jest nią ta, której index jest dodatni, czyli strefa nr 7 w rozumieniu solvera, co tak naprawdę odpowiada ostatniej strefie, którą do niego dodaliśmy czyli strefie o id 16. Ze widoku tabeli z wyżej można odczytać, iż faktycznie jest to strefa o najmniejszej zajętości, czyli to o co nam chodzi. Dalej solvera szuka parkingu, takiego, który nie ma więcej niż 10 miejsc parkingowych, jest strzeżony, ale niepłatny, a także jest przystosowany dla niepełnosprawnych.

Szukamy więc wśród wszystkich naszych parkingów i porównujemy z naszym ideałem.

```
[ParkingId: 27    Score: 12
, ParkingId: 25    Score: 3
, ParkingId: 28    Score: 3
, ParkingId: 32    Score: 3
, ParkingId: 34    Score: 3
, ParkingId: 45    Score: 3
, ParkingId: 46    Score: 3
, ParkingId: 21    Score: 2
, ParkingId: 26    Score: 2
, ParkingId: 29    Score: 2
, ParkingId: 35    Score: 2
, ParkingId: 37    Score: 2
, ParkingId: 38    Score: 2
, ParkingId: 40    Score: 2
, ParkingId: 43    Score: 2
, ParkingId: 44    Score: 2
, ParkingId: 47    Score: 2
, ParkingId: 48    Score: 2
, ParkingId: 49    Score: 2
, ParkingId: 50    Score: 2
, ParkingId: 53    Score: 2
, ParkingId: 17    Score: 1
, ParkingId: 18    Score: 1
, ParkingId: 19    Score: 1
, ParkingId: 22    Score: 1
, ParkingId: 23    Score: 1]
```

Jak widać wskazany został parking o id 27 i otrzymał Score wynoszący 12.

Przjrzyjmy się więc tabeli parkingów.

	 parking_lot_id [PK] bigint	 free_spaces integer	 is_for_handicapped boolean	 is_guarded boolean	 is_paid boolean	 spot_size integer	 zone_id bigint
1	27	2	false	false	false	5	16
2	41	99	false	false	true	4	15
3	38	96	true	false	false	4	15
4	46	93	true	true	false	1	15
5	17	91	false	true	true	2	15
6	34	6	true	false	false	2	15
7	25	80	true	true	false	4	14
8	54	48	false	true	true	1	14
9	49	79	true	false	false	1	14

Widać wyraźnie, że jest tylko 1 parking znajdujący się w strefie nr 16. I to właśnie on został wskazany jako najlepszy. Mimo wszystko sprawdzimy, czy otrzymał poprawną ilość punktów. Liczymy +10 za bycie w pożądanej strefie, +1 za bycie niepłatnym parkingiem oraz +1 za ilość wolnych miejsc <10. Nie otrzymuje on jednak punktów za to, że jest strzeżony oraz dla niepełnosprawnych, ponieważ ewidentnie nie jest.

e. Przykład nr 2

Rozpatrzmy następane zapytanie na identycznej bazie danych z poprzedniego przykładu.

czyDba_o_Wygode integer(\$int32) (query)	czyDba_o_Wygode 1
czySkapy integer(\$int32) (query)	czySkapy 1
Lat integer(\$int32) (query)	Lat -1
Lon integer(\$int32) (query)	Lon 1
userId integer(\$int32) (query)	userId 71

Tym razem obiektem testu jest użytkownik i id 71 znajdujący się w strefie o współrzędnych (-1,1), który wybrał opcje tego, że parking powinien być niepłatny, a także powinien posiadać więcej niż 10 wolnych miejsc parkingowych.

	user_id [PK] bigint	age integer	car_size integer	handicaped boolean	name character varying (255)	preferable_zone bigint	surname character varying (255)
13	69	33	7	false	Jack	15	aaa
14	70	56	9	false	Jack	14	aaa
15	71	29	6	false	Jack	8	aaa
16	72	29	6	false	Jack	7	aaa
17	73	50	11	false	Jack	15	aaa

W tym przypadku klient nie jest osobą niepełnosprawną.

	zone_id [PK] bigint	cordx integer	cordy integer	attractiveness_ratio double precision	occupied_ratio double precision	request_ratio double precision	zone_type character varying (255)
1	1	-2	-2	0.28	0.14	0.03	test
2	2	-2	-1	0.45	0.58	0.26	test
3	3	-2	0	0.44	0.3	0.83	test
4	4	-2	1	0.57	0.73	0.48	test
5	5	-1	-2	0.16	0.85	0.69	test
6	6	-1	-1	1	0.7	0.64	test
7	7	-1	0	0.07	0.51	0.44	test
8	8	-1	1	0.37	0.58	0.86	test
9	9	0	-2	0.1	0.64	0	test
10	10	0	-1	0.35	0.35	0.34	test
11	11	0	0	0.29	0.12	0.18	test
12	12	0	1	0.21	0.42	0.18	test
13	13	1	-2	0.46	0.79	0.01	test
14	14	1	-1	0.87	0.33	0.84	test
15	15	1	0	0.88	0.64	0.43	test
16	16	1	1	0.79	0.08	0.96	test

Tym razem klient znajduje się w strefie nr 8 z sąsiadującymi strefami: 4, 7, 12. Jesteśmy więc na obrzeżach miasta. Jak widać najniższy współczynnik zajętości ma strefa nr 7, więc mogłoby się wydawać, że to właśnie ona będzie najlepszą strefą, jednak strefa nr 7 mająca prawie identyczny współczynnik zajętości, ma o wiele wyższy współczynnik zapotrzebowania, co sprawi, że to właśnie ona będzie miała wyższy priorytet.

Tabela priorytetów:

IdStrefy	Priorytet
4	12
7	11
8	10
12	12

Interesuje nas bowiem najniższy priorytet, gdyż strefy dostają ujemne wagi, więc mniejsza liczba dodatnia to większa ujemna.

Solver zwraca nam taki oto wynik:








```
-2 1
4
-1 0
7
-1 1
8
0 1
12
[-1, -2, 3, -4, -9, 10, -11, -12]
```

Oczywiście w wyniku mamy tylko 4 strefy, ponieważ tylko one otaczają klienta. Idealny parking w tym przypadku to taki, który znajduje się w strefie nr 8 oraz nie posiada więc niż 10 miejsc parkingowych, jest strzeżony, niepłatny oraz nie dla niepełnosprawnych.

Szukamy więc wśród wszystkich naszych parkingów i porównujemy z naszym ideałem.

```
[ParkingId: 28      Score: 12
, ParkingId: 21      Score: 11
, ParkingId: 29      Score: 11
, ParkingId: 26      Score: 3
, ParkingId: 27      Score: 3
, ParkingId: 37      Score: 3
, ParkingId: 43      Score: 3
, ParkingId: 47      Score: 3
, ParkingId: 50      Score: 3
, ParkingId: 17      Score: 2
, ParkingId: 18      Score: 2
, ParkingId: 22      Score: 2
, ParkingId: 23      Score: 2
, ParkingId: 24      Score: 2
, ParkingId: 25      Score: 2
, ParkingId: 32      Score: 2
, ParkingId: 34      Score: 2
, ParkingId: 39      Score: 2
, ParkingId: 42      Score: 2
, ParkingId: 45      Score: 2
, ParkingId: 46      Score: 2
, ParkingId: 51      Score: 2
, ParkingId: 54      Score: 2]
```

Jak widać wskazany został parking o id 28 i otrzymał Score wynoszący 12.

	 parking_lot_id [PK] bigint	 free_spaces integer	 is_for_handicapped boolean	 is_guarded boolean	 is_paid boolean	 spot_size integer	 zone_id bigint
5	21	85	true	false	false	4	8
6	22	98	false	false	false	1	7
7	23	20	false	false	false	1	10
8	24	39	false	true	true	3	13
9	25	80	true	true	false	4	14
10	26	46	false	true	false	3	4
11	27	2	false	false	false	5	16
12	28	41	true	true	false	1	8
13	29	46	true	true	true	3	8
14	30	54	true	false	true	4	6
15	31	65	false	false	true	3	3

Tym razem aż 3 parkingi znajdują się w najlepszej dla klienta strefie. Są to parkingi 28, 21, 29, które otrzymują +10 za bycie w strefie nr 8. Teraz do gry wchodzi parametry parkingu. Parking 28 okazał się być najlepszym z nich wszystkich otrzymując +1 za to, że jest parkingiem strzeżonym, a także za to, że jest bezpłatny. Parking 21 także jest bezpłatny lecz nie jest strzeżony, a parking 29 jest płatny. Wszystkie zaś są dla osób niepełnosprawnych, a także mają więcej niż 10 wolnych miejsc, a my szukaliśmy parkingu o dokładnie odwrotnych parametrach, dlatego nie otrzymują one punktów za te rzeczy.

f. Przykład nr 3

Za pomocą zapytania /GenerateData generujemy 100 stref ,150 parkingów i 250 użytkowników.

Użytkownik dużego samochodu o id 351

user_id [PK] bigint	age integer	car_size integer	handicaped boolean	name character varying (255)	preferable_zone bigint	surname character varying (255)
351	41	8	false	Jack	74	aaa

Poszukuje bezpłatnego i wygodnego miejsca parkingowego w pobliżu strefy o współrzędnych (-1,-4). Aby znaleźć odpowiednie miejsce parkingowe wykonujemy zapytanie /sfps o następujących parametrach:

<input checked="" type="checkbox"/>	Lat	-1
<input checked="" type="checkbox"/>	Lon	-5
<input checked="" type="checkbox"/>	userId	351
<input checked="" type="checkbox"/>	czySkapy	1
<input checked="" type="checkbox"/>	czyDbao_Wygode	1

Api rozpoczyna konstrukcje Solvera, i w tym celu wyróżnia strefy w pobliżu klienta

```
32 : -2 -4
33 : -2 -3
41 : -1 -5
42 : -1 -4
43 : -1 -3
52 : 0 -4
53 : 0 -3
```

Następnie solver sprawdza wartości zajętości, zapotrzebowania i atrakcyjności dla stref i decyduje do której strefy powinien należeć parking wybrany dla klienta, oraz jakie powinien mieć inne cechy.

	zone_id [PK] bigint	cordx integer	cordy integer	attractiveness_ratio double precision	occupied_ratio double precision	request_ratio double precision	zone_type character varying (255)	
1		32	-2	-4	0.0551063888647466	0.276346569611499	0.941145222000478	test
2		33	-2	-3	0.904023915511799	0.335167365751443	0.677824394708195	test
3		41	-1	-5	0.361249767709071	0.982333386731565	0.795595319419931	test
4		42	-1	-4	0.382818110515683	0.539018584973172	0.0970593957563165	test
5		43	-1	-3	0.355183281167909	0.72555191992145	0.182910611045846	test
6		52	0	-4	0.1097093358625	0.117723439814062	0.481663132676592	test
7		53	0	-3	0.31417076629165	0.352965711789939	0.411349979881608	test

Zwraca on następujący wynik:








```
[1, -2, -3, -4, -5, -6, -7, -9, 10, -11, -12]
```


W powyższym równaniu zmienne od 1-7 reprezentują strefę, do której może należeć parking, solver wybrał strefę 1 ponieważ ma ona najgorszy stosunek zapotrzebowania do zapełnienia co oznacza że z wszystkich otaczających stref potrzeba samochodów jest tam największa a pozostałe zmienne opisują cechy parkingu. Zmienna 10 symbolizuje fakt że parking powinien być strzeżony co odpowiada preferencji wygody użytkownika. Zaprzeczona zmienna 11 odpowiada faktowi że parking jest niepłatny.

Następnie Api porównuje każdy z dostępnych parkingów z obliczonymi cechami idealnego parkingu i na podstawie tych cech wystawia każdemu parkingowi ocenę odpowiadającą jego przydatności dla systemu i klienta.

2	[ParkingId: 188	Score: 12
3	,	ParkingId: 331	Score: 12
4	,	ParkingId: 259	Score: 11
5	,	ParkingId: 117	Score: 4
6	,	ParkingId: 129	Score: 4
7	,	ParkingId: 255	Score: 4
8	,	ParkingId: 284	Score: 4
9	,	ParkingId: 315	Score: 4
10	,	ParkingId: 104	Score: 3

Klientowi są proponowane w pierwszej kolejności parkingi z najwyższym wynikiem

	 parking_lot_id [PK] bigint	 free_spaces integer	 is_for_handicapped boolean	 is_guarded boolean	 is_paid boolean	 spot_size integer	 zone_id bigint
1	117	2	false	true	false	4	28
2	129	2	false	true	false	4	11
3	188	4	true	false	false	3	32
4	255	6	false	true	false	3	6
5	259	8	true	false	true	0	32
6	284	2	false	true	false	2	78
7	331	29	false	true	true	5	32

g. Przykład nr 4

Niepełnosprawny użytkownik małego samochodu o id 352

	user_id [PK] bigint	age integer	car_size integer	handicaped boolean	name character varying (255)	preferable_zone bigint	surname character varying (255)
1	352	31	3	true	Jack	57	aaa

Poszukuje jakiegokolwiek miejsca parkingowego w pobliżu strefy o współrzędnych (0,3). Aby znaleźć odpowiednie miejsce parkingowe wykonujemy zapytanie /sfps o następujących parametrach:

	KEY	VALUE	DI
<input checked="" type="checkbox"/>	Lat	0	
<input checked="" type="checkbox"/>	Lon	3	
<input checked="" type="checkbox"/>	userId	352	
<input checked="" type="checkbox"/>	czySkapy	0	
<input checked="" type="checkbox"/>	czyDbA_o_Wygode	0	

Api przygotowuje Solver i znajduje strefy sąsiadujące z samochodem Użytkownika oraz ich współrzędne:

```
49 : -1 3
50 : -1 4
58 : 0 2
59 : 0 3
60 : 0 4
69 : 1 3
70 : 1 4
```

Następnie solver sprawdza wartości zajętości, zapotrzebowania i atrakcyjności dla stref i decyduje do której strefy powinien należeć parking wybrany dla klienta, oraz jakie powinien mieć inne cechy.

	zone_id [PK] bigint	cordx integer	cordy integer	attractiveness_ratio double precision	occupied_ratio double precision	request_ratio double precision	zone_type character varying (255)
1	49	-1	3	0.275700093646901	0.590140812866817	0.550791121376829	test
2	50	-1	4	0.500872022656173	0.467126194398332	0.954574760609397	test
3	58	0	2	0.234789005049007	0.633257767740826	0.712643872526088	test
4	59	0	3	0.068258660889454	0.213679213541164	0.479677228219686	test
5	60	0	4	0.622162519290615	0.319569166550603	0.863167862290061	test
6	69	1	3	0.399305398264419	0.0688406339781474	0.89302066870482	test
7	70	1	4	0.0188653240067137	0.241872297664667	0.0961934634124422	test

Zwraca on następujący wynik:

```
[-1, -2, -3, -4, -5, 6, -7, -9, -10, 11, 12]
```








W powyższym równaniu zmienne od 1-7 reprezentują strefę, do której może należeć parking, a pozostałe opisują cechy parkingu. Zmienna 10 symbolizuje fakt że parking powinien być

strzeżony co odpowiada preferencji wygody użytkownika. Zmienna 11 odpowiada faktowi że parking może być płatny.

Następnie Api porównuje każdy z dostępnych parkingów z obliczonymi cechami idealnego parkingu i na podstawie tych cech wystawia każdemu parkingowi ocenę odpowiadającą jego przydatności dla systemu i klienta.

```
2  [ParkingId: 163      Score: 13
3  , ParkingId: 259      Score: 4
4  , ParkingId: 102      Score: 3
5  , ParkingId: 114      Score: 3
6  , ParkingId: 115      Score: 3
7  , ParkingId: 118      Score: 3
8  , ParkingId: 121      Score: 3
9  , ParkingId: 133      Score: 3
10 , ParkingId: 134      Score: 3
11 , ParkingId: 140      Score: 3
12 , ParkingId: 142      Score: 3
13 , ParkingId: 147      Score: 3
14 , ParkingId: 153      Score: 3
15 , ParkingId: 157      Score: 3
16 , ParkingId: 158      Score: 3
17 , ParkingId: 164      Score: 3
18 , ParkingId: 178      Score: 3
19 , ParkingId: 179      Score: 3
```

Parkingi na górze listy są najbardziej odpowiednie dla klienta

	 parking_lot_id [PK] bigint	 free_spaces integer	 is_for_handicapped boolean	 is_guarded boolean	 is_paid boolean	 spot_size integer	 zone_id bigint
1	102	58	true	false	true	3	34
2	114	14	true	false	true	5	78
3	115	55	true	false	true	3	83
4	118	13	true	false	true	4	23
5	121	0	true	false	false	0	79
6	163	25	true	false	true	1	69
7	259	8	true	false	true	0	32

h. Przykład nr 5

Niepełnosprawny użytkownik małego samochodu o id 352

	user_id [PK] bigint	age integer	car_size integer	handicapped boolean	name character varying (255)	preferable_zone bigint	surname character varying (255)
1	360	65	9	false	Jack	90	aaa

Poszukuje jakiegokolwiek miejsca parkingowego w pobliżu strefy o współrzędnych (0,3). Aby znaleźć odpowiednie miejsce parkingowe wykonujemy zapytanie /sfps o następujących parametrach:

	KEY	VALUE
<input checked="" type="checkbox"/>	Lat	-1
<input checked="" type="checkbox"/>	Lon	-1
<input checked="" type="checkbox"/>	userId	360
<input checked="" type="checkbox"/>	czySkapy	0
<input checked="" type="checkbox"/>	czyDba_o_Wygode	1

Api przygotowuje Solver i znajduje strefy sąsiadujące z samochodem Użytkownika oraz ich współrzędne:

```
35 : -2 -1
36 : -2 0
44 : -1 -2
45 : -1 -1
46 : -1 0
55 : 0 -1
56 : 0 0
```

Następnie solver sprawdza wartości zajętości, zapotrzebowania i atrakcyjności dla stref i decyduje do której strefy powinien należeć parking wybrany dla klienta, oraz jakie powinien mieć inne cechy.

	zone_id [PK] bigint	cordx integer	cordy integer	attractiveness_ratio double precision	occupied_ratio double precision	request_ratio double precision	zone_type character varying (255)
1	35	-2	-1	0.103116616335745	0.889294159397506	0.766544116595869	test
2	36	-2	0	0.110402947592096	0.0927412510147047	0.784274526760515	test
3	44	-1	-2	0.86633829916126	0.893966113229307	0.446410192172608	test
4	45	-1	-1	0.794446871864174	0.478648436492477	0.522878836572374	test
5	46	-1	0	0.0322225160474648	0.443882994115743	0.684786468802912	test
6	55	0	-1	0.481791025363965	0.699519359262707	0.685239975650634	test
7	56	0	0	0.813834631049389	0.198376595271549	0.615026916160341	test

Zwraca on następujący wynik:

```
[-1, 2, -3, -4, -5, -6, -7, -9, 10, 11, -12]
```

W powyższym równaniu zmienne od 1-7 reprezentują strefę, do której może należeć parking, a pozostałe opisują cechy parkingu. Zmienna 10 symbolizuje fakt że parking powinien być strzeżony co odpowiada preferencji wygody użytkownika. zmienna 11 odpowiada faktowi że parking może być płatny.

Następnie Api porównuje każdy z dostępnych parkingów z obliczonymi cechami idealnego parkingu i na podstawie tych cech wystawia każdemu parkingowi ocenę odpowiadającą jego przydatności dla systemu i klienta.

```
2  [ParkingId: 108      Score: 13
3  , ParkingId: 103      Score: 12
4  , ParkingId: 107      Score: 11
5  , ParkingId: 349      Score: 11
6  , ParkingId: 193      Score: 4
7  , ParkingId: 223      Score: 4
8  , ParkingId: 280      Score: 4
9  , ParkingId: 346      Score: 4
10 , ParkingId: 109      Score: 3
11 , ParkingId: 117      Score: 3
12 , ParkingId: 127      Score: 3
13 , ParkingId: 129      Score: 3
14 , ParkingId: 142      Score: 3
15 , ParkingId: 190      Score: 3
16 , ParkingId: 192      Score: 3
17 , ParkingId: 206      Score: 3
18 , ParkingId: 230      Score: 3
19 , ParkingId: 238      Score: 3
```

Parkingi na górze listy są najbardziej odpowiednie dla klienta

	parking_lot_id [PK] bigint	free_spaces integer	is_for_handicapped boolean	is_guarded boolean	is_paid boolean	spot_size integer	zone_id bigint
1	103	30	true	true	true	2	36
2	107	53	true	true	false	1	36
3	108	46	false	true	true	0	36
4	193	1	false	true	true	4	59
5	223	1	false	true	true	1	40
6	280	7	false	true	true	1	94
7	349	28	false	false	false	4	36

13. Wnioski końcowe

Podsumowując powyższe przykłady, zakładamy, że nasz Solver działa poprawnie. Wskazuje nam strefę w otoczeniu klienta, która ma najwyższy priorytet, a następnie wybiera odpowiedni parking wewnątrz tej strefy, tak aby był jak najbliższy ideału.