## 3. Opis proponowanego rozwiązania

### 3.1 Algorytm grafowy Floyda-Warshalla

Mapa miasta, składająca się przede wszystkich z ulic i skrzyżowań, została zaimplementowana w postaci grafu skierowanego, w którym ulice stanowią krawędzie, a skrzyżowania – wierzchołki grafu (przy czym dla ulic jednokierunkowych, ścieżka od jednego wierzchołka do drugiego jest możliwa do przejścia tylko zgodnie z kierunkiem ruchu na danej ulicy). Rys. 1. przedstawia przykładowy fragment mapy, na którym naniesiono strukturę grafu.



Rys. 1. Fragment mapy z naniesioną strukturą grafu

(Źródło: maps.google.com, opracowanie własne)

Z tego względu, należało podjąć decyzję polegającą na wybraniu algorytmu, który jak najlepiej poradzi sobie z wyznaczaniem najkrótszych ścieżek pomiędzy dwoma dowolnymi wierzchołkami.

Charakterystyczną cechą implementowanego systemu jest bardzo mała zmienność – ulice nie zmieniają przecież swojej długości, a nowe połączenia drogowe nie pojawiają się ani nie znikają zbyt często. Dzięki temu nie ma potrzeby każdorazowego wyznaczania ścieżki pomiędzy dwoma takimi samymi punktami na mapie, jeżeli nie zaszły żadne zmiany. Można więc przyjąć, że trasy łączące dowolne dwa skrzyżowania wystarczy wyznaczyć tylko raz i ewentualnie przeliczać je na nowo dopiero w momencie, w którym zaszła zmiana.

Wiedząc o powyższych cechach systemu, wybrano algorytm Floyda-Warshalla. Pozwala on na wyznaczenie ścieżek pomiędzy wszystkimi parami wierzchołków grafu (o ile takie połączenie istnieje). Zapamiętanie wyznaczonych ścieżek pozwala na natychmiastowe podanie drogi łączącej dwa wierzchołki. Algorytm ten został szczegółowo zaprezentowany w książce „Grafy i rekurencji” J. Mareckiego [3], poniżej zostanie jedynie pokrótce przedstawiony w postaci pseudo-kodu.

Punktem startowym algorytmu jest macierz H, dla której na przecięciu wiersza i oraz kolumny j znajduje się wartość k oznaczająca długość krawędzi łączącej wierzchołki i oraz j. Jeżeli taka krawędź nie istnieje, to wartość w tym miejscu wynosi ∞. Algorytm wyznacza w kolejnych krokach macierze D(i) dla i = 0..n, zawierające minimalne odległości pomiędzy wierzchołkami grafu:

n:=Rozmiar(H)

D(0) := H

for k := 1 to n do

for i := 1 to n do

for j := 1 to n do

d(k)ij := min {d(k-1)ij, d(k-1)ik + d(k-1)kj}

return D(n)

Listing 1. Algortym Floyda-Warshalla w postaci pseudokodu [3]

Jednakże, oprócz poznania najkrótszej odległości pomiędzy dwoma punktami, niezbędne jest także pamiętanie poszczególnych wierzchołków na ścieżce. W tym celu tworzona jest macierz poprzedników S, która początkowo, na przecięciu wiersza i oraz kolumny j przyjmuje wartość i, jeżeli istnieje krawędź łącząca wierzchołki i oraz j lub nil, jeżeli taka krawędź nie istnieje. Następnie, jeżeli funkcja min powyższego algorytmu zwróci drugą wartość, to na przecięciu wiersza i oraz kolumny j zostanie wstawiona wartość k. Gdy algorytm zakończy swoje działanie, ścieżka pomiędzy wierzchołkami a i b będzie mogła zostać odtworzona za pomocą poniższego sposobu:

if sab != nil then

Pokaż(a,b)

print(b)

else nie istnieje ścieżka z a do b

Listing 3. Metoda odtworzenia ścieżki z punktu a do b w formie pseudokodu [3]

Funkcja rekurencyjna Pokaż(x,y) została zdefiniowana jak nżej:

Pokaż(i,j)

k := sij

if i = k then print(k)

else Pokaż(i,k)

Pokaż(k+1,j)

Listing 3. Funkcja Pokaż(x,y) w postaci pseudokodu [3]

Funkcja ta wypisuje kolejne wierzchołki znalezionej ścieżki. Cały algorytm można zmodyfikować w ten sposób, aby po wyznaczeniu macierzy D – odległości i macierzy S – poprzedników, zapamiętywać ściezki wyznaczone metodą jak na listingu 2. Dzięki temu, oprócz początkowego wyliczenia, całe działanie algorytmu ograniczy się jedynie do zwracania wcześniej wyznaczonych ścieżek. Taki sposób działania pozwala na natychmiastowe podawanie trasy przejazdu.

Jednakże takie podejście również rodzi kilka problemów, zaczynając od wysokiej złożoności obliczeniowej algorytmu, która ze względu na potrójną pętlę for sięga rzędu O(n3), poprzez konieczność przechowywania macierzy o rozmiarze n x n na potrzeby wyznaczania odległości i ścieżki, aż do przymusu pełnego przeliczania algorytmu po dokonaniu jakichkolwiek zmian w postaci grafu. Na etapie realizacji niniejszej pracy stwierdzono jednak, że problemy te nie są aż tak istotną przeszkodą, aby rezygnować z tego algorytmu na rzecz innego, który miałby każdorazowo od nowa wyznaczać trasę.

### 3.2 Mechanizm zarządzania uprawnieniami użytkowników

Oprócz algorytmu grafowego, który jest „sercem” aplikacji, należało także zaimplementować kilka mechanizmów, które ułatwią użytkowanie programu. Jednym z nich jest kontrola uprawnień użytkowników, polegająca na ograniczeniu dostępu do poszczególnych funkcjonalności.

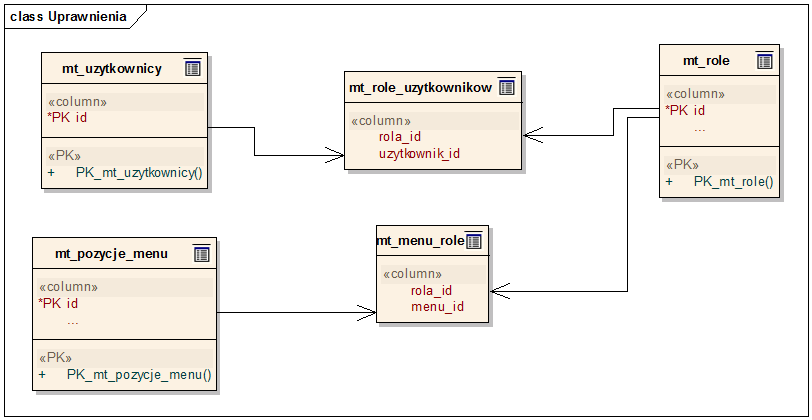
Podstawową formą ograniczenia dostępu jest konieczność zalogowania się, aby móc korzystać z aplikacji. Dzięki temu, nie będzie możliwości uzyskania nieatoryzowanego dostępu do żadnej z funkcjonalności systemu.

Oprócz tego, w zakresie zalogowanych użytkowników, można podzielić ich na grupy dysponujące różnym zestawem uprawnień. Można także pozostawić pewien ograniczony zakres funkcjonalności dostępny dla niezalogowanych użytkowników. Ten właśnie model został zaimplementowany w aplikacji będącej przedmiotem niniejszej pracy.

Użytkownik niezalogowany może korzystać jedynie z modułu klienta – aplikacji desktopowej służącej do zamówienia taksówki. Po zalogowaniu się, użytkownik będący klientem może uzyskać dostęp do bardziej szczegółowych opcji zamówienia, może także przeglądać dane swojego konta. Z kolei, jeżeli użytkownik jest taksówkarzem, zostaje mu udostępniony oddzielny panel zawierający niezbędne informacje.

Część aplikacji dostępna przez przeglądarkę internetową jest implementacją modułu dyspozytora. Zalogować się do niej mogą jedynie użytkownicy będący dyspozytorami bądź użytkownik – szef korporacji. Pozostali użytkownicy, pomimo posiadania konta w systemie, nie mogą uzyskać dostępu do tej części aplikacji. Również w tej części systemu zakres dostępnych do wykonania operacji zależy od uprawnień użytkownika.

Poszczególne uprawnienia zostały opisane w pojedynczej tabeli bazy danych. Kolejna tabela zawiera przyporządkowanie uprawnień do roli. „Rola” w tym kontekście jest zestawem uprawnień. Osobna tabela przechowuje przypisanie ról do użytkowników. Schemat powiązań tabel w bazie danych obrazujący powyższy opis został przedstawiony na rysunku 2.



Rys. 2. Schemat bazy danych dotyczący ról i uprawnień użytkowników

(opracowanie własne)

Przyjęto, że poszczególni użytkownicy, posiadający dane role, mają w ich zakresie takie same uprawnienia. Podczas realizacji opisywanego systemu nie pojawiła się konieczność przypisywania użytkownikom indywidualnych uprawnień do jakiejkolwiek części systemu.