

WYDZIAŁ FIZYKI I INFORMATYKI STOSOWANEJ



DOKUMENTACJA PROJEKTU  
**Grafowa baza danych POC**

Aleksandra Poręba  
nr. indeksu 290514

11 grudnia 2019

# Spis treści

<b>1</b>	<b>Opis działania aplikacji</b>	<b>3</b>
1.1	Dodawanie . . . . .	3
1.2	Modyfikacja . . . . .	4
1.3	Usuwanie . . . . .	4
1.4	Wyszukiwanie . . . . .	4
<b>2</b>	<b>Budowa aplikacji</b>	<b>4</b>
2.1	Modele . . . . .	4
2.2	Widoki . . . . .	6
<b>3</b>	<b>Typy danych</b>	<b>6</b>
<b>4</b>	<b>Wykorzystane technologie</b>	<b>7</b>
<b>5</b>	<b>Bibliografia</b>	<b>7</b>

# 1 Opis działania aplikacji

Celem aplikacji było zademonstrowanie użycia grafowej bazy danych.

Aplikacja udostępnia możliwość wyszukiwania najszybszego połączenia pomiędzy przystankami. Użytkownik może dodawać własne przystanki i relacje między nimi, usuwać je, a także modyfikować.

Na stronie głównej, przedstawionej na rysunku 1, znajduje się lista wszystkich dostępnych przystanków w aplikacji.



Rysunek 1: Strona główna aplikacji

Przemieszczanie po stronie odbywa się za pomocą paska nawigacji.

## 1.1 Dodawanie

W aplikacji możemy dodawać zarówno przystanki (w zakładce "Dodaj przystanek") jak i połączenia (w zakładce "Dodaj połączenie"). Odpowiednie informacje należy podać w formularzu, a następnie potwierdzić przyciskiem "Dodaj".

Jeśli użytkownik chce dodać przystanek w formularzu musi podać jego nazwę oraz adres.

Aby utworzyć nowe połączenie pomiędzy punktami, należy wybrać dwa odpowiednie przystanki z listy, podać numer autobusu (może być też tramwaju), którym możemy odbyć tą trasę oraz szacowany czas przejazdu w minutach.

Aplikacja jest zabezpieczona przed dodaniem nieprawidłowych danych, takich jak puste wartości, za krótka nazwa ulicy, czy wybór dwóch takich samych przystanków.

## 1.2 Modyfikacja

W zakładce "Modyfikuj" mamy możliwość zmienienia danych dotyczących przystanku. Po wybraniu nazwy punktu, którego chcemy edytować, zostaje utworzony formularz do zmiany danych z aktualnymi wartościami, pobranymi z bazy.

## 1.3 Usuwanie

Aplikacja udostępnia możliwość usunięcia przystanków razem z wszystkim połączeniami do nich, oraz pojedynczych połączeń. Aby tego dokonać należy wybrać interesujące użytkownika nazwy z listy.

## 1.4 Wyszukiwanie

W zakładce "Wyszukaj połączenia" użytkownik może wyszukać najszybszą trasę pomiędzy dwoma przystankami.

Jako rezultat zostaje wyświetlona tabela z kolejnymi przystankami, numerem autobusu (tramwaju), którym możemy dojechać na dany przystanek, oraz ile czasu podróży już minęło, od jej rozpoczęcia. Przykładowa tabela przedstawiona została na rysunku 2.

# 2 Budowa aplikacji

Projekt składa się z `models.py`, który zawiera model przystanku, komunikujący się z bazą danych, `views.py` generujący widoki aplikacji oraz `templates` z szablonami widoków.

## 2.1 Modele

Aplikacja zawiera jeden model danych - model przystanku, który reprezentuje węzeł w bazie. Identyfikowany jest on za pomocą nazwy. Aby wykonać operacje na węzłach wysyłane są zapytania języka Cypher, generowane na

Strona główna Wyszukaj połączenia Dodaj przystanek Dodaj połączenie Modyfikuj Usuń przystanek Usuń połączenie		
<b>Najkrótsze połączenie:</b>		
Zajmie ono 17 minut.		
Przystanek	Numer autobusu	Czas podróży w minutach
Plac Inwalidów	-	0
Urzędnicza	4	2
Biprostal	4	5
Kawiory	194	10
Miasteczko Studenckie	208	17

Rysunek 2: Przykładowy rezultat wyszukiwania. Aby dojechać z przystanku "Plac Inwalidów" do "Miasteczko Studenckie" należy pojechać autobusem (tramwajem) numer 4 przez przystanek "Urzędnicza" do przystanku "Biprostal". Zajmie to 5 minut. Następnie należy przesiąść się na autobus numer 194 i dojechać nim na "Kawiory". Ostatnim etapem trasy będzie podróż autobusem 208 do przystanku docelowego. Cała podróż zajmie 17 minut.

podstawie danych przesłanych od widoku z formularzy. Przykładowe zapytania:

- dodawanie przystanku  
`CREATE (p:Przystanek {nazwa:'Kawiory', ulica: 'Nawojki', numer:'1'})`
- modyfikacja danych przystanku  
`MATCH (p:Przystanek {nazwa: 'Kawiory'}) SET n.ulica = 'Nawojki', n.numer = '2' RETURN n`
- usuwanie przystanku  
`MATCH (p:Przystanek {nazwa: 'Kawiory'}) DETACH DELETE p`

Model przystanku operuje też na połączeniach między węzłami, na przykład:

- dodawanie relacji  
`MATCH (p1:Przystanek {nazwa: 'AGH'}), (p2:Przystanek {nazwa:`

```
'Kawiory'}) CREATE (p1) - [r:POLACZONY{ numer: '194',  
czas: 3}]->(p2) RETURN r
```

- usuwanie relacji

```
MATCH (p1:Przystanek{nazwa: 'AGH'})-[r:POLACZONY]-  
(p2:Przystanek {nazwa:'Kawiory'}) DELETE r
```

- sprawdzanie, czy relacja istnieje

```
MATCH (s:Przystanek{nazwa: 'AGH'}), (e:Przystanek {nazwa:  
'Kawiory'}) RETURN EXISTS ((s)-[:POLACZONY]-(e))
```

Wyszukiwanie najkrótszej ścieżki odbywa się za pomocą zapytania wykorzystującego algorytm Neo4j `algo.shortestPath`:

```
MATCH (s:Przystanek {nazwa:'AGH'}), (e:Przystanek {nazwa:'Kawiory'})  
CALL algo.shortestPath.stream(start, end, 'czas')  
YIELD nodeId, cost  
RETURN algo.asNode(nodeId).nazwa AS name, cost
```

## 2.2 Widoki

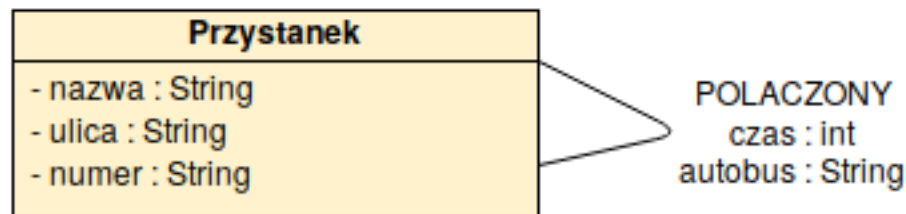
W pliku `views.py` znajdują się funkcje odpowiadające na żądania przekazywane przez obiekt `HttpRequest`. Widoki generują odpowiednie szablony. Formularze przekazywane są do widoków na pomocą żądania typu `POST`. Są one najpierw walidowane, a następnie wywoływana jest odpowiednia funkcja modelu. Gdy model zwróci odpowiedź, widok musi odpowiednio na nią zareagować - sprawdzić, czy operacja powiodła się i przeparsować wynik, aby można było go umieścić w szablonie.

## 3 Typy danych

Przystanek w bazie danych jest opisywany poprzez węzeł zawierający trzy atrybuty: nazwę przystanku oraz adres: ulicę i numer. Wszystkie z nich przechowywane są jako łańcuch znaków. Oprócz atrybutów Neo4j dodaje do węzłów identyfikator.

Przystanki mogą być ze sobą połączone relacją o etykiecie "POLACZONY". Zawiera ona dwa atrybuty - czas przejazdu podany w minutach oraz numer

autobusu (lub tramwaju), którym możemy pokonać daną trasę. Czas przechowywany jest jako liczba całkowita, numer autobusu jako łańcuch znaków (czasem pojawiają się litery przy numerach autobusów, aby określić jego specjalny typ, np. nocny)



Rysunek 3: Model UML

## 4 Wykorzystane technologie

Aplikacja została zrealizowana za pomocą mikroframeworka Flask w języku Python 3. Dostęp do bazy danych jest realizowany przez bibliotekę neo4jrest-client, wysyłającą zapytania do serwera REST bazy Neo4j. Połączenie jest realizowane przez HTTP REST. Baza danych jest dostarczana przez graphenedb, która jest bazą Neo4j w chmurze, hostowana na Heroku.

Do oprawy graficznej strony został użyty framework Bootstrap4. Szablony widoków uzupełniane są za pomocą Jinja.

Aplikacja znajduje się w środowisku chmurowym Heroku pod adresem: <https://mighty-mountain-55683.herokuapp.com/>.

## 5 Bibliografia

Using with Python and Neo4j Rest Client  
The Neo4j Cypher Manual v3.5

neo4j-rest-client's Documentation  
The Neo4j Graph Algorithms User Guide v3.5