

21.12.2020

Przetwarzanie Danych w Chmurach Obliczeniowych


Dokumentacja projektu


Celem projektu było stworzenie aplikacji wykorzystującej grafową bazę danych. Stworzyłem aplikację webową w React.js z serwerem w node.js. Serwer posiada REST'owe API i realizuje żądania wysyłane przez klienta. Do komunikacji z bazą danych wykorzystuje powszechnie dostępny sterownik dla grafowej bazy danych. Aplikacja klienta nie jest znacząco rozbudowana w efekty wizualne, ani w piękny UI, niemniej jednak w pełni realizuje założoną funkcjonalność. Założenia projektu są typu proof-of-concept, tzn. aplikacja ma zaprezentować możliwości, typowe wykorzystania i zalety grafowej bazy danych w konkretnych zastosowaniach. Jako temat przewodni moje pracy wybrałem aplikację społecznościową typu facebook lub twitter.


Aplikacja oferuje proces rejestracji użytkownika, wymagając podania kilku podstawowych informacji:


- imienia
- nazwiska
- daty urodzenia (YYYY-MM-DD)
- adresu email
- hasła dostępu


Po udanej rejestracji, następuje przekierowanie na stronę logowania. Poprawne logowanie skutkuje przekierowaniem na główną stronę aplikacji, tzw. Dashboard, gdzie wyświetlane są posty wszystkich użytkowników, których zalogowany użytkownik obserwuje, komentarze innych użytkowników do postów oraz liczba polubień.

 Piotr Nowak

 Logout


 Dashboard

 Wall

 Explore

Hubert Mazur

2020-12-21 12:7:50

 5

Graph databases are really cool!

Type

COMMENT


Lidia Ostawa

2020-12-21 12:18:54

Indeed

Hubert Mazur

2020-12-20 15:6:13

 4

First post for testing purposes

Type

COMMENT

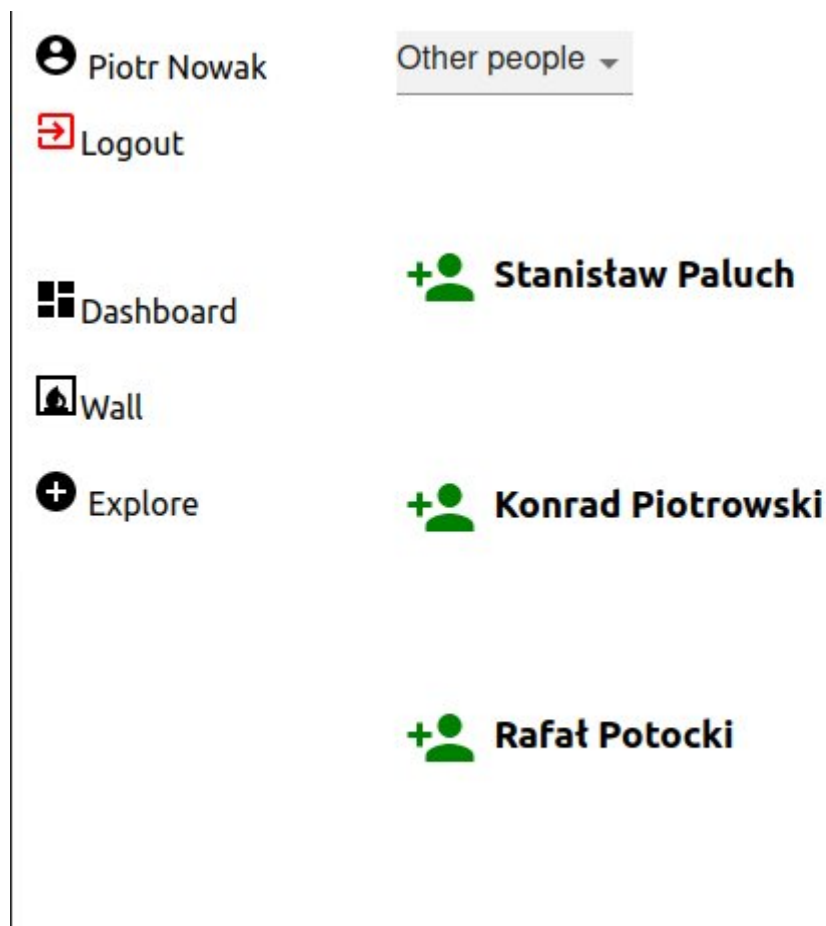
Hubert Mazur

Post użytkownika zawiera informacje o jego dodaniu, osobie, która go dodała i dotyczących go komentarzach. Możliwe jest dodanie nowego komentarza poprzez wpisanie jego treści w polu **Type** oraz kliknięciu przycisku **COMMENT**. Komentarze pod postem sortowane są w kolejności od najnowszych. Dodatkowo w poście wyświetlana jest liczba polubień w postaci serduszka i towarzyszącej mu liczby polubień. Dla użytkownika kolor serca może być czarny lub czerwony w zależności, czy użytkownik polubił już dany post. Klikając w ikonę można polubić lub usunąć polubienie postu, co objawi się zmianą koloru.

Kolejną sekcją jest **wall**, który interfejsem nie różni się wcale od strony głównej (**dashboard**), z tą różnicą, że wyświetlane są tylko posty zalogowanego użytkownika, oraz istnieje możliwość usunięcia postu. Pozostałe reguły zostają niezmienione.

Ostatnia sekcja mojej aplikacji umożliwia przegląd innych użytkowników pod różnym kątem. Zmiana parametru wyboru umożliwia odmienny sposób selekcji użytkowników, wśród nich są:

- **People You might know** - pokazuje użytkowników, których możemy znać, na zasadzie wspólnych znajomych, których liczba jest również wyświetlana (jest to najciekawsza funkcjonalność w kontekście grafowej bazy danych, toteż informacja o tym trybie wyszukiwania w bazie znajduje się w dalszej części dokumentacji)
- **Other People** - wszyscy użytkownicy aplikacji, których obecnie się nie obserwuje
- **People You follow** - użytkownicy aktualnie obserwowani
- **People that are following You** - użytkownicy, którzy nas obserwują



Dodanie użytkownika do obserwowanych następuje poprzez kliknięcie zielonej ikony osoby. W przypadku, gdy chcemy przestać obserwować, wówczas ikona jest przekreślona i ma kolor czerwony.

Projekt zakłada wykorzystanie grafowej bazy danych jako składnicy danych. Wykorzystana została baza **Neo4j**, zdecydowanie będąca liderem wśród grafowych baz danych. Użycie takiego modelu danych w mojej aplikacji znacznie ułatwia realizację wielu funkcjonalności mojej aplikacji. Moja aplikacja w dużej mierze opiera się o relacje, np.:

- użytkownik polubił post
- użytkownik obserwuje innego użytkownika
- użytkownik dodał komentarz do postu

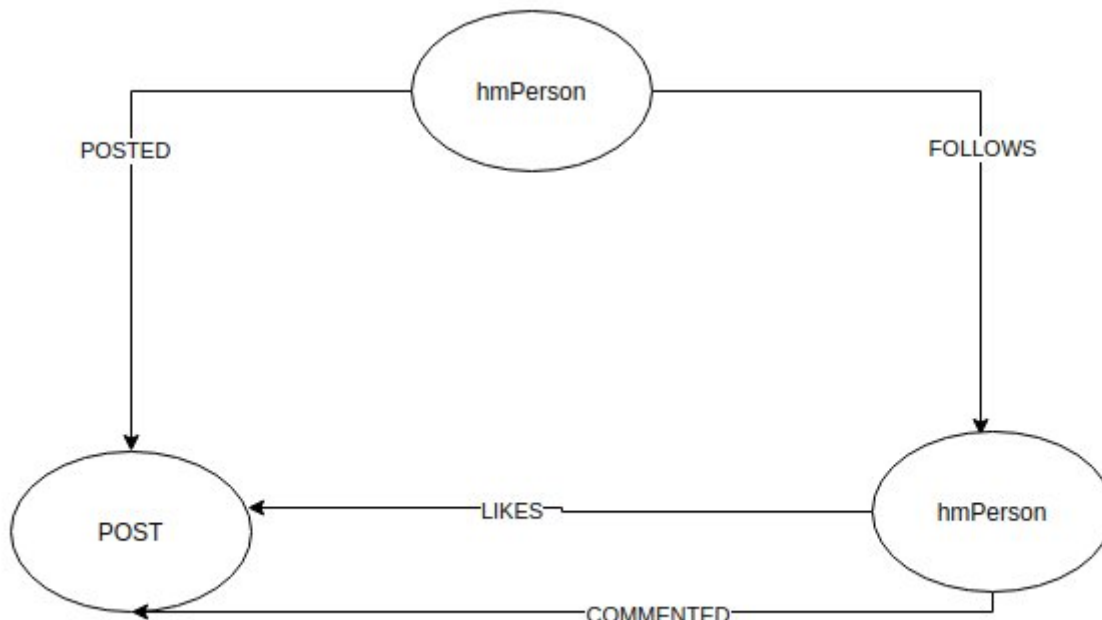
Analizując powyższe zasady, nie sposób nie zgodzić się, że grafowa baza danych idealnie wpasowuje się w potrzeby. Oczywiście, dla samego przechowywania danych, baza SQL'owa również sobie poradzi, jednak analiza i przetwarzanie danych byłoby dużo trudniejsze lub niemal karkołomne.

Moja aplikacja opiera się o dwa typy węzłów: **hmPerson** oraz **hmPOST**. Każdy z nich przechowuje dodatkowe informacje. Węzły przedstawione zostały na poniższych rysunkach:

hmPerson	
name	string
lastName	string
born	string
email	email
password	string

hmPOST	
content	text
timestamp	timestamp

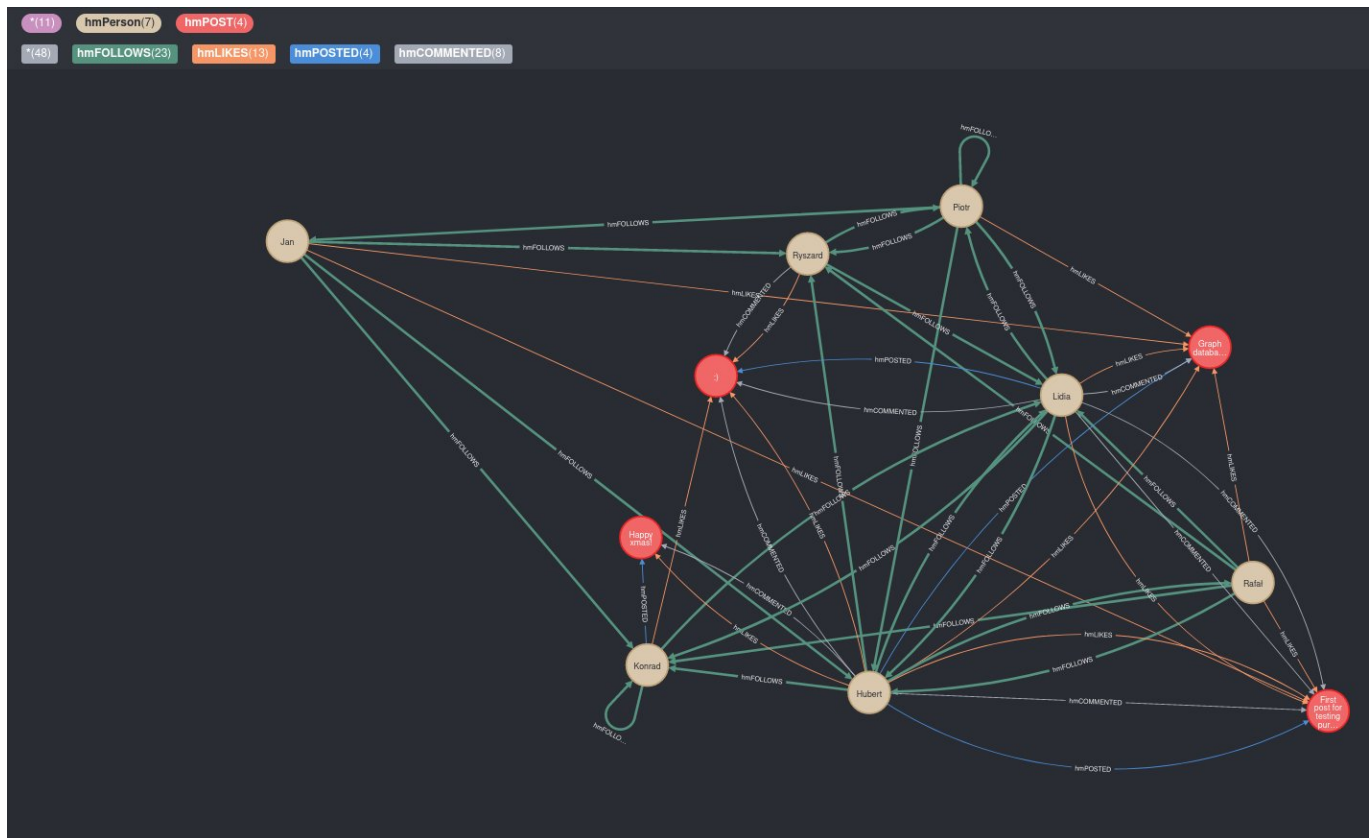
Węzły i ich własności przedstawione na powyższych rysunkach łączone są zależnościami. Węzły i wszystkie możliwe zależności pomiędzy nimi przedstawia rysunek poniżej:



Podsumowując, relacje możliwe pomiędzy poszczególnymi typami węzłów:

- (hmPerson)-[:FOLLOWS\]-(hmPerson) - relacja obserwowania
- (hmPerson)-[:POSTED | COMMENTED | LIKED\]-(hmPOST), osoba może opublikować, polubić lub skomentować post.

Na rysunku został zamieszczony model grafowej bazy danych reprezentujący użytkowników aplikacji, ich posty, polubienia postów i relacje między użytkownikami. Zielonym kolorem oznaczone są relacje obserwowania - relacja może być obustronna. Kolor niebieski to relacja dodania postu, tzn. (użytkownik)-[dodał]->(post). Relacja jest jednostronna. Kolor pomarańczowy określa relację polubienia, tzn. (użytkownik)-[lubi]->(post), przy czym ta relacja jest również jednostronna (użytkownik polubił post, ale post nie może polubić użytkownika, ani wejść z nim w żadną inną interakcję).



W poprzedniej części pracy zostało wspomniane o dużej zalecie grafowej bazy danych w przypadku analizy połączeń. Tym samym, w łatwy sposób można znaleźć listę wspólnych znajomych i zaporponować ją dla użytkownika. Ten przykład zapytania w bazie i jego analiza zostały przedstawione poniżej.

```
match (me:hmPerson)-[f:hmFOLLOWS]->(myfriends:hmPerson)-
[theirFollows:hmFOLLOWS]->(c:hmPerson) WHERE id(me) = $id AND NOT (me)-
[:hmFOLLOWS]->(c) RETURN c, COUNT(theirFollows) as tf ORDER BY tf DESC
```

Nazwa **hmPerson** jest identyfikatorem grupy węzłów, spośród których należy wyszukiwać. Powyższe zapytanie możnaby przetłumaczyć w następujący sposób: użytkownik obserwuje innych użytkowników, a ci użytkownicy jeszcze innych. Zwracana jest informacja o tych użytkownikach (znajomi znajomych) oraz liczba wspólnych znajomych (im więcej wspólnych znajomych, tym większa szansa, że znamy daną osobę). Zliczana jest ilość relacji obserwowania użytkowników przez znajomych. Powyższe rozwiązanie wydaje się być absolutnie naturalne i w niczym nie zaskakujące w kontekście grafów, jednak w bazie SQL, zrealizowanie tego byłoby nie lada wyczynem.

Ostatnim zapytaniem, które w pewien sposób jest szczególne i zasługuje by znaleźć swoje miejsce w niniejszej dokumentacji jest zapytanie o posty użytkowników, które następnie zostaną wyświetlone w panelu **Dashboard**. Jest ciekawe ze względu na swoją złożoność względem pozostałych zapytań.

```
MATCH (me:hmPerson)-[r:hmFOLLOWS]->(b:hmPerson)-[p:hmPOSTED]->(n:hmPOST)
WHERE id(me) = $id OPTIONAL MATCH (n)<-[com:hmCOMMENTED]-(c:hmPerson) CALL
{with n MATCH (n)<-[l:hmLIKES]-(l:hmPerson) RETURN COUNT(l) as likes}RETURN
b.name, b.lastName, n, id(n), collect({com:com,name: c.name,
lastName:c.lastName}) as comments, likes, EXISTS((me)-[:hmLIKES]->(n)) as
```

```
doILike ORDER BY n.timestamp DESC
```

Zapytanie znajduje wszystkie posty znajomych użytkownika, oraz wszystkie komentarze do tych postów. Dodatkowo zlicza wszystkie relacje polubień każdego postu, jak i sprawdza operatorem `EXISTS`, czy użytkownik wysyłający żądanie polubił już dany post. Kolejno w sekcji `RETURN` tworzona jest dla każdego postu kolekcja użytkowników, którzy skomentowali post, oraz treść komentarza i znacznik czasowy. Zastosowanie tego znacząco zmniejsza liczbę rekordów wynikowych, w innym przypadku, bez stosowania `collect` efekt byłby podobny do `LEFT JOIN` w bazie SQL. Komentarze nie byłyby zagregowane w zakresie jednego postu - każdy komentarz znajdowałby się w oddzielnym rekordzie.

Pozostałe zapytania konieczne do realizacji innych funkcjonalności są mniej rozbudowane i w dużym stopniu proste, dlatego nie będą tutaj przedstawione.

Aplikacja dostępna jest w serwisie heroku (linki):

Aplikacja klienta: <https://postus-fe.herokuapp.com>

Aplikacja serwera (API REST): <https://postus-be.herokuapp.com>

Kod źródłowy aplikacji:

Aplikacja klienta: <https://github.com/hubert-mazur/postUs-front>

Aplikacja serwera: <https://github.com/hubert-mazur/postUs>