Dokumentacja projektu z przedmiotu „Przetwarzanie danych w chmurach obliczeniowych”

Grafowa baza danych informacji o filmach – projekt typu ‘Proof of concept’

Jakub Strugała
AGH
Wydział Fizyki i Informatyki Stosowanej
Informatyka Stosowana
15.12.2021

Gotowa aplikacja została umieszczona na platformie chmurowej *Heroku* i dostępna jest pod następującym linkiem:

https://movie-searcher-application.herokuapp.com

Kod źródłowy można znaleźć na hostingowym serwisie *GItHub* pod linkiem:

https://github.com/JJ807/MovieSearcherApplication

1. Stack technologiczny

Do powstania aplikacji użyto następujących rozwiązań technologicznych:

- Neo4j – system zarządzania grafową bazą danych
- AuraDBFree – serwis chmurowy dla grafowych baz Neo4j, na którym umieszczona została baza danych wykorzystywana w projekcie
- Spring Boot – framework do języka Java, za pomocą którego napisana została część back-end'owa projektu
- Angular – framework wykorzystujący język TypeScript; posłużył do zrealizowania części front-end'owej aplikacji
- Git – system kontroli wersji
- Github – serwis hostingowy, gdzie znajduje się kod źródłowy projektu
- Heroku – platforma chmurowa, na której postawiona została ogólnie dostępna aplikacja

2. Koncepcja i założenia

Celem projektu było stworzenie grafowej bazy danych informacji o filmach w oparciu o system zarządzania bazą danych Neo4j oraz interfejsu dostępu do bazy.

Aplikacja powinna wykorzystywać Neo4j do stworzenia sieci połączeń pomiędzy osobami związanymi z kinematografią a samymi filmami.
W ten sposób, za pomocą zapytań do bazy w języku Cypher, możliwym będzie otrzymanie informacji o relacjach osób z filmami. Następnie, informacje te są mapowane na odpowiednie klasy DTO, które znajdują się w folderze *„movies"* [link].

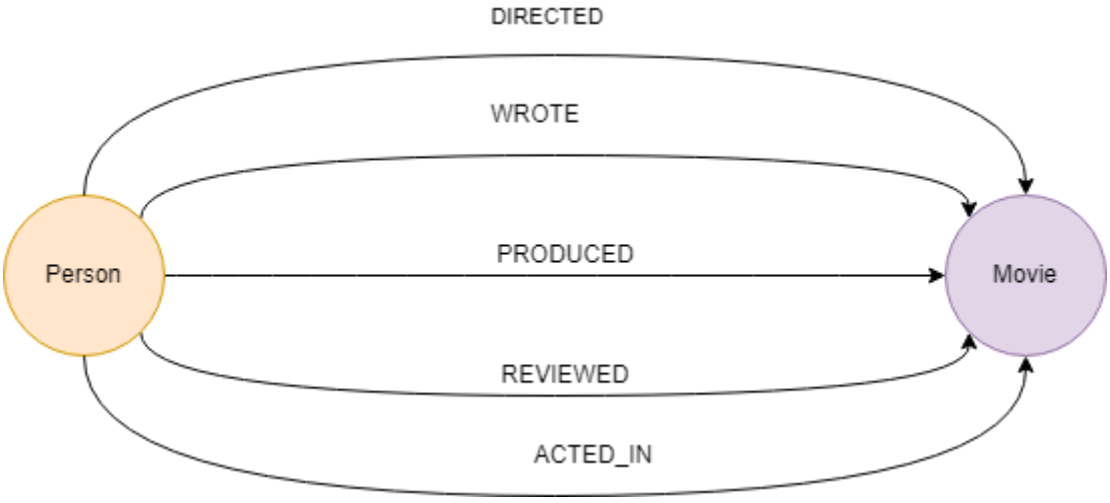
Kolejno, wspomniane obiekty DTO mają swoje odpowiedniki po stronie front-end'owej aplikacji w postaci interfejsów w klasie *HomeComponent* [link], gdzie zostają odpowiednio stylowane i wyświetlane.

3. Baza danych i relacje węzłów

Baza danych została utworzona za pomocą query przedstawionego w pliku *„createDatabase.cypher"* dostępnego tutaj:

https://github.com/JJ807/MovieSearcherApplication/blob/master/createDatabase.cypher

Prosta relacja pomiędzy węzłami reprezentującymi filmy oraz aktorów przedstawia się następująco:

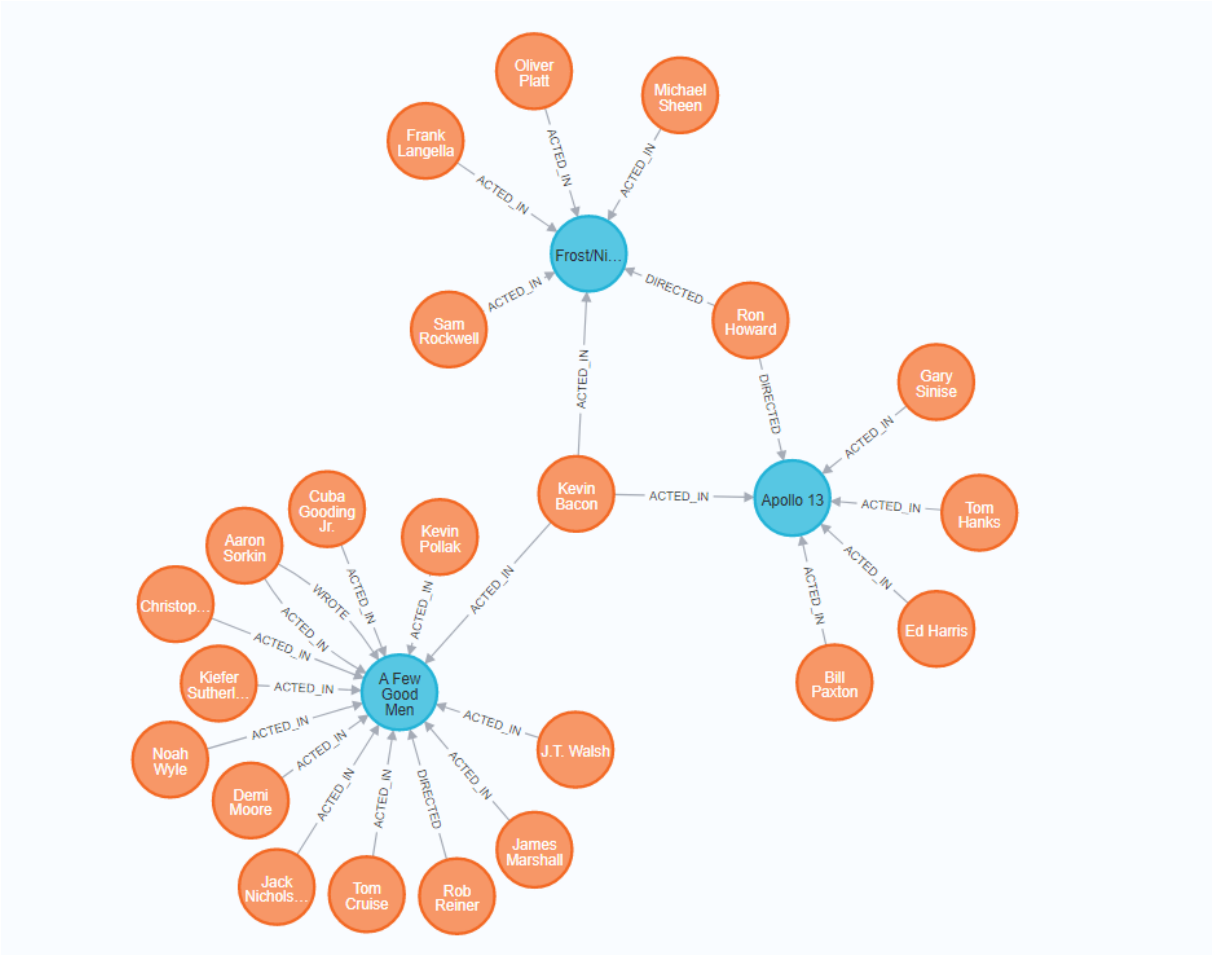


***Rysunek 1**: Relacja pomiędzy węzłem Person oraz węzłem Movie*

Poniżej znajduje się przykładowe zapytanie pokazujące wyraźnie tą relację

```
MATCH (bacon:Person {name:"Kevin Bacon"})-[*0..2]-(neighbourhood)
RETURN DISTINCT neighbourhood
```

oraz jego efekt widoczny w aplikacji Neo4j Browser:



Rysunek 2: *Graficzna reprezentacja bezpośredniego sąsiedztwa aktora Kevin'a Bacon'a – filmy i osoby*

Na załączonym rysunku widać, że zapytanie tworzy skierowany graf relacji aktorów i filmów, gdzie od każdej osoby wychodzi relacja mówiąca o roli jaką ta osoba odegrała w tworzeniu filmu. Od każdej osoby może prowadzić więcej niż jedna relacja, jak w przypadku Aaron'a Sorkin'a na powyższym rysunku.

4. Udostępnione punkty końcowe usługi sieciowej (endpointy)

W ramach zaimplementowanego rozwiązania, udostępniono 4 endpointy widoczne w pliku *„MovieController.java”* [link]. Natomiast serwis, który wykonuje faktyczne operacje znajduję się tutaj.

Pod wspomnianymi endpointami znajdują się funkcje wykonujące kolejno:

- […/movie/{title}]
  Request typu GET o detale filmu zwracający tytuł, który został dopasowany do podanego w ścieżce ciągu znaków oraz osoby z tym tytułem związane. Początkowo sprawdzane jest czy w faktycznym tytule filmu zawiera się podany ciąg.

  Zapytanie do bazy wykonywane w ramach tego endpoint’a:

```
MATCH (movie:Movie {title: $title})
OPTIONAL MATCH (person:Person)-[r]->(movie)
WHERE TOLOWER(movie.title) CONTAINS TOLOWER($title)
WITH movie, COLLECT({ name: person.name, job: REPLACE(TOLOWER(TY
PE(r)), '_in', ''), role: HEAD(r.roles) }) as cast
RETURN movie { .title, cast: cast }
```

- […/movie/{title}/like]
  Request typu POST inkrementujący liczbę polubień filmu, który został dopasowany do zmiennej *title* w ścieżce. W tym wypadku nie istnieje potrzeba sprawdzania czy ciąg znaków zawiera się w tytule filmu, gdyż przekazujemy dokładny tytuł filmu pobrany dla każdego filmu z poprzedniego zapytania.

  Zapytanie do bazy wykonywane w ramach tego endpoint’a:

```
MATCH (m:Movie {title: $title})
WITH m, coalesce(m.likes, 0) AS currentVotes
SET m.likes = currentVotes + 1;
```

- […/searchMovie]
  Request typu GET zwracający podstawowe informacje o filmie (tytuł, podtytuł, datę opublikowania oraz liczbę polubień).

Zapytanie do bazy wykonywane w ramach tego endpoint'a:

```
MATCH (movie:Movie) WHERE TOLOWER(movie.title) CONTAINS TOLOWER ($title) RETURN movie ORDER BY movie.title
```

- […/searchPerson]
  Request typu GET zwracający filmy, w których aktor/ka o podanym imieniu i nazwisku grał/a. Zapytanie do bazy sprawdza najpierw czy istnieje aktor o imieniu i nazwisku, które zawierałoby podany ciąg znaków.

  Zapytanie do bazy wykonywane w ramach tego endpoint'a:

```
MATCH (person:Person WHERE TOLOWER(person.name) = TOLOWER($personName))-[:ACTED_IN]->(personMovies) RETURN person,personMovies
```

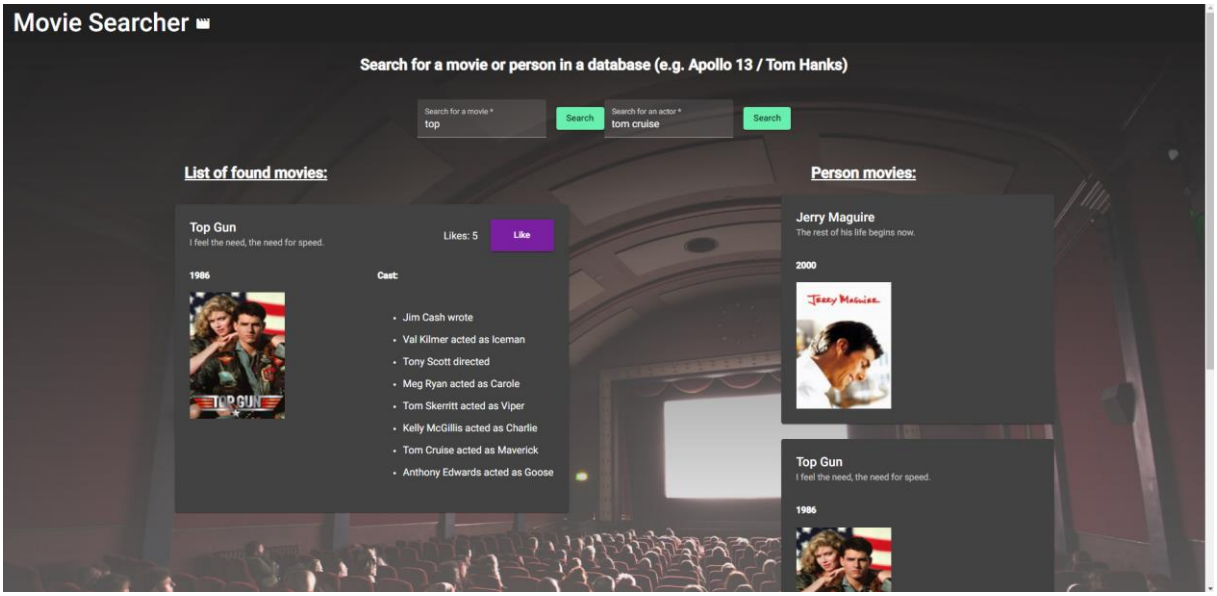
5. Dokumentacja użytkownika

W ramach interfejsu użytkownika udostępnione zostały 3 podstawowe funkcjonalności.

Użytkownik odwiedzający stronę może:

- wyszukiwać filmy na podstawie tytułu,
- wyszukiwać aktorów na podstawie ich imienia i/lub nazwiska
- ma możliwość polubienia danego filmu, poprzez kliknięcie przycisku „Like”, który inkrementuje liczbę polubień dla tego filmu a następnie aktualizuje widok

Warstwa wizualna aplikacji:



Rysunek 3: *Warstwa wizualna aplikacji*