

Architektura aplikacji opartej o różne klasy baz danych na przykładzie serwisu internetowego o filmach

(Application architecture based on different database models on an example
of website about movies)

Kamil Matuszewski

Praca inżynierska

Promotor: dr Paweł Rajba

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

7 lutego 2018

Streszczenie

Podczas projektowania aplikacji internetowej należy wybrać model bazy danych, który odpowiadałby naszym zapotrzebowaniom. Każdy model ma swoje wady i zalety, więc w zależności od funkcjonalności aplikacji, pewne modele będą się w niej lepiej sprawdzały niż inne. Problemem jest jednak sytuacja, w której nie jesteśmy w stanie znaleźć modelu, spełniającego wszystkie nasze zapotrzebowania. Zamiast starać się wybrać system zarządzania bazą danych, który efektywnie realizuje wybrane funkcjonalności, można zaprojektować aplikację, która korzysta nie z jednego ale z kilku modeli. Jest to jednak bardziej skomplikowane. W tej pracy staram się poradzić sobie z tymi komplikacjami i zaprojektować oraz zaimplementować właśnie taką aplikację.

When designing a web application, it is required to choose a database model that meets the application's needs. Each model has its advantages and disadvantages, so depending on the functionalities of the application, some models work better than others. The problem is, however, in a situation where we are unable to find a model that meets all of our needs. Instead of trying to choose a database management system that effectively implements only a part of functionalities, we could design an application that uses not one, but several models. However, it is more complicated. In this work I try to deal with these problems and design and implement an application that uses various database models.

Spis treści

1. Wprowadzenie	9
1.0.1. Przegląd wymagań	10
2. Modele baz danych	13
2.1. Przegląd modeli baz danych	13
2.1.1. Model relacyjny	13
2.1.2. Model grafowy	14
2.1.3. Model tekstowy	15
2.2. Problemy stosowania wielu modeli baz danych	15
2.3. Modele zastosowane w rozwiązaniu	16
2.3.1. MySQL	16
2.3.2. Neo4J	17
2.3.3. MongoDB	20
2.4. Podsumowanie	21
3. Architektura aplikacji	25
3.1. Warstwa widoku	25
3.2. Warstwa trasowania	27
3.3. Warstwa kontrolera	27
3.3.1. Kontrolery Auth	27
3.3.2. GenreController	27
3.3.3. RecommendationController	27
3.3.4. ListController	28

3.3.5. LogController	28
3.3.6. MovieController	28
3.3.7. PersonController	28
3.3.8. ReviewController	28
3.3.9. SearchController	29
3.3.10. UserController	29
3.4. Warstwa modelu	29
3.4.1. EditHistory	29
3.4.2. Genre	29
3.4.3. Movie	30
3.4.4. Person	30
3.4.5. Review	30
3.4.6. SearchStats	30
3.4.7. User	30
3.5. Warstwa dostępu do danych	31
3.5.1. Eloquent	31
3.5.2. NeoEloquent	31
3.5.3. Moloquent	32
4. Funkcjonalności	33
4.1. Pasek nawigacyjny	33
4.2. Strona logowania	34
4.3. Strona rejestracji	34
4.4. Strona domowa	34
4.5. Listy	35
4.5.1. Lista filmów	35
4.5.2. Lista osób	36
4.5.3. Lista użytkowników	36
4.6. Wyszukiwarka	37
4.7. Strona filmu	37

4.7.1. Strona edycji filmu	38
4.7.2. Strona recenzji	39
4.7.3. Strona dodawania recenzji	40
4.8. Strona osoby	40
4.8.1. Strona edycji osoby	41
4.9. Profil użytkownika	41
4.9.1. Strona edycji użytkownika	42
4.10. Strona dodawania	43
4.11. Strona logów	43
5. Instalacja aplikacji	45
5.1. Wymagania	45
5.2. Instalacja i uruchamianie	45
Bibliografia	47

Rozdział 1.

Wprowadzenie

Jedną z pierwszych rzeczy o których trzeba pomyśleć podczas tworzenia aplikacji przechowującej jakieś dane, jest wybranie odpowiedniej bazy danych. Na rynku znajdujemy wiele rozwiązań. Każde z nich ma swoje wady i zalety. Decydując się na jakiś model próbujemy maksymalnie wykorzystać jego zalety, minimalizując straty związane z jego wadami. Skoro nasza aplikacja nie musi czegoś robić, albo robi to bardzo rzadko, nie musimy przejmować się tym, że wybrany przez nas model nie radzi sobie najlepiej z takimi zapytaniem. Co jednak jeśli na rynku nie ma opcji pokrywającej wszystkie nasze potrzeby? Musimy decydować się na jakiś kompromis. Porzucić efektywność pewnych funkcjonalności na rzecz innych. Zdecydować na przykład, które z nich są ważniejsze i na tej podstawie wybrać odpowiedni model. A może istnieje inne rozwiązanie? Rozwiązanie, które nie wymaga od nas porzucania żadnych funkcjonalności?

Skoro każde rozwiązanie ma swoje wady i zalety, to czy nie rozsądnie byłoby, zamiast wybierać jedno z nich, wybrać kilka wzajemnie się uzupełniających? Skoro jeden model nie radzi sobie z czymś, co staje się siłą drugiego, to łącząc te podejścia można by było stworzyć aplikację, która wybiera zawsze najefektywniejszą drogę, która wykonuje zapytania tam, gdzie będzie najwygodniej i najefektywniej. Można stworzyć aplikację korzystającą nie z jednego ale z kilku modeli baz danych. Projekt takiej aplikacji jest jednak bardziej skomplikowany, niż w przypadku aplikacji korzystającej tylko z jednego modelu, ale być może dzięki temu otrzymamy produkt lepiej dostosowany do potrzeb naszej aplikacji.

Aplikacja powstała jako część pracy została stworzona przy pomocy frameworku Laravel, bazującym na modelu MVC (Model-View-Controller). Napisana została w języku PHP. Jest to prosty serwis, będący bazą filmów. Każdy użytkownik może polubić film bądź osoby z nim związane (aktorów, reżyserów, scenarzystów). Na podstawie polubień dla każdego użytkownika powstaje lista rekomendacji, dzięki której może on znaleźć nowy film dla siebie. Aplikacja ta jest przykładem, jak można sobie poradzić z obsługą kilku baz danych.

1.0.1. Przegląd wymagań

W aplikacji chcielibyśmy mieć cztery poziomy uprawnień: **gość**, **użytkownik zalogowany**, **moderator** oraz **administrator**. Gościem nazywamy dowolną osobę odwiedzającą serwis. Użytkownikiem zalogowanym jest osoba odwiedzająca serwis, która zarejestrowała swój profil dzięki czemu może współtworzyć ranking oraz pozycję filmów. Moderator to osoba, która ma uprawnienia do zarządzania bazą osób czy filmów, z kolei administrator ma możliwość zarządzania użytkownikami i ich uprawnieniami.

Gość ma prawo do:

- Przeglądania list filmów, użytkowników i osób
- Wyszukiwania filmów, użytkowników i osób
- Przeglądania profili poszczególnych użytkowników oraz stron poszczególnych filmów i osób

Użytkownik zalogowany ma prawo do:

- Przeglądania strony z rekomendacjami
- Usuwania rekomendacji i przywracania rekomendacji usuniętych
- Oceniania, polubienia i pisania recenzji do filmu (każdy użytkownik może napisać maksymalnie jedną recenzję do każdego z filmów)
- Obserwowania użytkowników
- Edycji swojego profilu
- Obserwowania osób
- Zaznaczenia/odznaczenia opcji wyświetlania obejrzanych filmów podczas wyszukiwania

Moderator ma prawo do:

- Edytowania informacji o filmach i osobach
- Dodawania i usuwania filmów oraz osób
- Dodawania i usuwania gatunków filmów

Administrator ma prawo do:

- Edycji informacji o użytkownikach i nadawania im uprawnień

- Usuwania użytkowników
- Przeglądania strony logów: historii edycji, historii wyszukiwania
- Usuwania logów starszych niż 30 dni (możliwość konfiguracji liczby dni)

Rozdział 2.

Modele baz danych

Projekt aplikacji, która używać będzie kilku modeli baz danych, wymaga z początku wybrania owych modeli. Należy więc zastanowić się nad kilkoma kwestiami. Oczywiście, należy wybrać modele a do każdego modelu odpowiedni system zarządzania, najlepiej taki, który wspiera język, w którym aplikacja ma być napisana. Należy też rozsądnie rozdzielić funkcjonalności pomiędzy wszystkie modele.

2.1. Przegląd modeli baz danych

2.1.1. Model relacyjny

Model relacyjny jest prawdopodobnie obecnie najbardziej popularnym modelem. Ideą jest przechowywanie danych w postaci tabel. Kolumny w takiej tabeli odpowiadają odpowiednim atrybutom, natomiast wiersze odpowiadają pojedynczemu obiektowi określonego typu. Dla przykładu, w tabeli osób, atrybutami (kolumnami) będą *imię*, *nazwisko*, *rok urodzenia*, natomiast wierszem (bądź krotką) będzie *Jan, Kowalski, 01-01-2000*. W modelu relacyjnym, każda krotka może posiadać unikalny **klucz główny**. ”Możliwe jest także dodawanie **klucza obcego**, czyli unikalnej wartości dzięki której możliwe jest odwołanie się do konkretnego wiersza w tabeli. Klucz obcy umożliwia między innymi łączenie tabel [1].

Obiekty w modelu relacyjnym mają stały, określony **schemat**. Każdy obiekt należy do jakiejś tabeli, a każda tabela ma z góry określone atrybuty. Każdy z atrybutów ma także określony typ. Takie rozwiązanie wspiera sprawdzanie integralności danych podczas ich dodawania - na przykład sprawdzanie, czy wprowadzona wartość rzeczywiście jest datą. Możemy też określać szczególne wymagania - na przykład, że dana wartość musi zostać wprowadzona lub musi być unikalna. Dzięki temu możliwe jest skuteczne analizowanie takich danych - wyświetlanie potrzebnych kolumn, łączenie tabel a także sortowanie i grupowanie [2].

W takim modelu baz danych musimy poradzić sobie jeszcze z relacjami. Two-

rzenie relacji **jeden-do-jednego** oraz **wiele-do-jednego** zwykle realizowane jest poprzez dodanie klucza obcego, wskazującego na odpowiednie wiersze. Stworzenie relacji **wiele-do-wielu** jest nieco bardziej skomplikowane - należy utworzyć dodatkową tabelę, zawierającą informacje o powiązaniach pomiędzy obiektami. Dla przykładu, stworzenie tabeli „zna” wraz z kolumnami „użytkownik1” oraz „użytkownik2”. Każdy wiersz takiej tablicy zawiera więc dwa atrybuty, będące kluczami obcymi z tabeli „użytkownik”.

2.1.2. Model grafowy

Podczas gdy model relacyjny przechowuje dane w postaci tabel, model grafowy przychodzi z nieco innym pomysłem. Zamiast reprezentować obiekt jako wiersz tabeli, możemy o nim pomyśleć jak o **wierzchołku**. Powiązania pomiędzy wierzchołkami można zaprezentować za pomocą **krawędzi**. Zapisując dane w ten sposób powstaje nam graf zależności. Taki graf może być zarówno skierowany jak i nieskierowany, a wierzchołki wcale nie muszą mówić o obiektach jednego typu. Przykładowo można więc pomyśleć o zbiorze miast. W każdym mieście mieszkają jacyś ludzie, znajdują się tam także jakieś sklepy. Ludzie mogą się oczywiście znać, być może mogą być ze sobą spokrewnieni. Mogą również lubić robić zakupy w pewnych sklepach. Sklepy natomiast mogą mieć różne produkty różnych rodzajów. Taką grupę danych dość naturalnie zaprezentować można właśnie w postaci grafu.

Idea grafowych baz danych polega na przechowywaniu w wierzchołkach informacji o sąsiadach, to znaczy wierzchołkach połączonych z nim krawędzią. W takim przypadku wyszukiwanie na przykład informacji o sklepach, w których dana osoba lubi robić zakupy, zdaje się być stosunkowo szybkie. Nie potrzebujemy, jak w przypadku baz relacyjnych, kluczy, które ułatwią nam wyszukiwanie. Bazy grafowe dość dobrze radzą sobie z powiązanymi ze sobą danymi. W rzeczywistości, w większości przypadków, nasze zapytania ograniczają się jedynie do pewnej porcji grafu. Graf staje się większy, ale lokalne sąsiedztwo wierzchołków rośnie dość wolno. Ciężko w końcu o sklep, w którym wszyscy lubiliby robić zakupy. Wpływ na to ma nie tylko jakość produktów i ceny, ale także lokalizacja sklepu - w końcu nikt nie będzie jeździł do sklepu oddalonego o kilkaset kilometrów, niezależnie od tego jak dobre jakościowo i cenowo produkty będzie można tam znaleźć. Fakt ten znajduje zastosowanie w praktyce - nie musimy się aż tak martwić o to, jak duża jest nasza baza oraz jak bardzo się powiększa, bo interesujące nas fragmenty aż tak bardzo się nie rozrastają - rośnie więc szybkość zapytań.

Łatwość poruszania się po takim grafie daje nam więc między innymi możliwość szybkiego wyszukiwania powiązań pomiędzy obiektami, co przydaje się na przykład podczas tworzenia silnika rekomendacji.

2.1.3. Model tekstowy

Jedną z głównych cech relacyjnych baz danych jest narzucony z góry schemat, który wymusza na nas ustalenie, jak będą wyglądać wrzucane do bazy dane, już na etapie projektowania aplikacji. Czasem, na początku tworzenia aplikacji, nie wiemy, jaka będzie dokładna struktura obiektów, które chcielibyśmy zapisywać w bazie. Może się również zdarzyć, że podczas rozwijania naszej aplikacji znajdzie potrzeba zapisywania dodatkowych informacji. W bazie relacyjnej tabele muszą od samego początku mieć określoną strukturę więc dodanie kolejnego atrybutu jest często kłopotliwe. Rozwiązaniem tych problemów jest podejście tekstowe - zamiast przechowywać dane w tabelach czy choćby grafach, trzymamy je jako odpowiednio sformatowane obiekty, na przykład przy pomocy **JSON** czy **XML**.

Bazy tekstowe pozwalają więc na łatwe przechowywanie danych, bez potrzeby martwienia się o wypełnianie wszystkich atrybutów, czy obawy o to, że w przyszłości będziemy potrzebować czegoś więcej. Być może utrudniamy w ten sposób wyszukiwanie, jednak jednocześnie upraszczamy logikę aplikacji. Dokumenty są niezależnymi jednostkami, nie tracimy więc na wydajności zapisów i odczytów [6].

2.2. Problemy stosowania wielu modeli baz danych

Projektowanie aplikacji używającej wielu modeli baz danych boryka się z pewnymi problemami, których nie mieliśmy podczas projektowania aplikacji używającej jedynie jednego modelu. Z uwagi na powiązania danego obiektu, będziemy go potrzebować w różnych kontekstach. Problem ten można rozwiązać na dwa sposoby.

- *Podzielenie danych*, dzięki któremu możemy uniknąć redundancji. Dane które pragniemy przechowywać dzielimy na rozłączne części, w zależności od naszych potrzeb, a następnie konsolidujemy obiekty na poziomie naszej aplikacji. Część danych będziemy odczytywać z jednej, część z drugiej a część z trzeciej bazy. Zapytania będą więc kierowane do bazy, w której dane informacje się znajdują. Możemy na przykład informacje o użytkownikach przechowywać w bazie MySQL, informacje o filmach oraz o osobach przechowywać w Neo4j, natomiast wszystkie logi takie jak historia edycji - w MongoDB. Główną zaletą takiego rozwiązania jest właśnie brak redundancji. Powstaje też pewien problem. Brak informacji o użytkownikach w bazie grafowej, oraz brak informacji o filmach w bazie relacyjnej, uniemożliwia nam zaznaczenie, które filmy dany użytkownik oglądał i jak je ocenił, co w efekcie sprawia, że nie jesteśmy w stanie stworzyć sensownego silnika rekomendacji. Problem jest bardziej złożony - analizując strukturę aplikacji można dojść do wniosku, że w tym przypadku ciężko o sensowny podział. To rozwiązanie jest sensowne, jeśli w naszej aplikacji są dwie osobne instancje niepowiązanych ze sobą danych, z których część łatwiej wykonywać w jednym modelu a część w drugim.

- *Powielanie obiektów* może być rozwiązaniem problemu pojawiającego się w poprzednim scenariuszu. Kiedy wszystkie dane w naszej aplikacji są ze sobą silnie powiązane, informacje o istnieniu wszystkich obiektów muszą być obecne przynajmniej w jednej z baz. Podczas tworzenia obiektu część informacji o nim przechowujemy w jednej bazie, inną część w drugiej, a jeszcze inną w trzeciej. Wszystkie trzy dane łączymy na przykład za pomocą unikalnego klucza, który pozwoli nam na odwoływanie się do informacji o danym obiekcie znajdującym się w innej bazie. W ten sposób trzymamy nadmiarowe informacje, a raczej powielamy informacje o istnieniu obiektu. Zapytania także stają się bardziej skomplikowane - musimy myśleć nie tylko o tym, jak wyszukać obiekt, ale też skąd pobrać które informacje. Sprawia to, że często musimy się odwoływać do tego samego obiektu kilkakrotnie - żeby pobrać wszystkie możliwe informacje o obiekcie należy odwołać się do każdej z baz, w których te informacje się znajdują. Sprawia to, że sam projekt staje się trudniejszy - musimy zastanowić się, która informacja przechowywać w której z baz, tak, żeby zminimalizować potrzebę odwoływania się do wszystkich.

2.3. Modele zastosowane w rozwiązaniu

W mojej aplikacji niejako połączyłem oba podejścia. Dane podzieliłem pomiędzy bazę grafową a bazę relacyjną, duplikując obiekty, ale część danych, które nie wymagają połączeń z resztą, przechowuję w bazie tekstowej. Poniżej znajdują się dokładniejsze opisy poszczególnych baz, oraz tego, jak wyglądają poszczególne zapytania.

2.3.1. MySQL

MySQL jest jednym z najpopularniejszych systemów zarządzania, opartym na relacyjnych bazach danych. Wspiera on język **SQL**, umożliwiając tworzenie dość prostych i intuicyjnych zapytań. Bazy tej będziemy chcieli użyć do wykonywania zapytań o grupę obiektów. Pobierać z niej będziemy listę wszystkich obiektów z określonej tabeli, oraz informacji o określonym obiekcie. Nie będziemy jednak przechowywać w niej informacji o relacjach, to znaczy nie będziemy tworzyć dodatkowych tabel dotyczących związków pomiędzy poszczególnymi obiektami.

W bazie znajdują się następujące tabele:

- Tabela użytkowników, **users**, wraz z kluczem głównym **id**, oraz unikalnymi wartościami **login** i **email**, przechowująca dane użytkowników, takie jak imię (**name**), nazwisko (**surname**), obrazek użytkownika (**avatar**), data urodzin (**birthday**), informacje o użytkowniku (**about**), lokalizację (**location**), płeć (**gender**), poziom uprawnień (**access**) oraz zaszyfrowane hasło (**password**).

Dodatkowo przechowywane będą informacje o dacie utworzenia (**created_at**) i modyfikacji (**updated_at**) oraz, na potrzeby aplikacji, specjalny token używany przy zaznaczeniu opcji „remember me” (więcej informacji w rozdziale 3).

- Tabela osób, **persons**, wraz z kluczem głównym **id**, przechowująca dane o osobach, takie jak imię (**name**), nazwisko (**surname**), zdjęcie (**photo**), data urodzin (**birthday**) czy biografia (**biography**). Dodatkowo przechowywane będą informacje o dacie utworzenia (**created_at**) oraz modyfikacji (**updated_at**).
- Tabela filmów, **movies**, wraz z kluczem głównym **id** przechowujące dane o filmach takie jak tytuł (**title**), data produkcji (**prod_date**), opis (**description**), zdjęcie (**photo**), ocena (**score**) i liczba ocen (**number_of_scores**). Dodatkowo przechowywane będą informacje o dacie utworzenia (**created_at**) oraz modyfikacji (**updated_at**).
- Tabela recenzji **reviews**, wraz z kluczem głównym **id**, przechowująca treści recenzji (**text**) oraz informacje o dacie utworzenia (**created_at**) oraz modyfikacji (**updated_at**).

Przykłady zapytań:

- Wybór wszystkich filmów:

```
| select * from movies;
```

- Wybór recenzji o konkretnym id:

```
| select * from reviews  
| where id=1;
```

- Aktualizacja określonego pola użytkownika (na przykład aktualizacja imienia):

```
| update users  
| set name="Alicja"  
| where login="Alicja11";
```

- Utworzenie nowej krotki osoby:

```
| insert into persons (name, surname)  
| values (Jan, Kowalski);
```

2.3.2. Neo4J

Neo4J jest stosunkowo nowym (pierwsza wersja ukazała się w 2010 [4]) systemem zarządzania, opartym na grafowych bazach danych. Wspiera on język **Cypher**,

wzorowany na SQL. Umożliwia on tworzenie czytelnych zapytań, bez potrzeby pisania jak chcemy dane informacje uzyskać [5].

Neo4J pozwala na tworzenie grafów z **etykietami**, to znaczy grupowania wierzchołków i krawędzi poprzez specjalną nazwę. Krawędzie mogą również zawierać dodatkowe informacje, takie jak data utworzenia relacji.

Typowe zapytania w języku Cypher są dość intuicyjne i proste do zrozumienia. Okrągłe nawiasy prezentują wierzchołki, myślniki tworzą krawędzie (skierowanie możemy zapisać za pomocą trójkątnego nawiasu). Dodatkowo pomiędzy krawędziami możemy umieścić kwadratowe nawiasy, by dostać się do danej relacji. Po dwukropku możemy określić etykietę wierzchołka bądź krawędzi. Możemy więc stworzyć zapytanie:

```
match (n:Osoba)-[l:JestOjcem]->(m:Osoba) return n;
```

Zwracające nam osoby będące czyimś ojcem.

Z bazy pobierać będziemy informacje o relacjach: wyszukiwać ścieżki, dla danego wierzchołka sprawdzać jakie wierzchołki są z nim w relacji.

Zbiór wierzchołków:

- Film, **Movie**, wraz z unikalnym identyfikatorem (**id**) oraz identyfikatorem filmu w bazie relacyjnej (**mid**).
- Gatunek, **Genre**, wraz z unikalną nazwą (**name**).
- Osoba, **Person**, wraz z unikalnym identyfikatorem (**id**) oraz identyfikatorem osoby w bazie relacyjnej (**pid**).
- Recenzja, **Review**, wraz z unikalnym identyfikatorem (**id**) oraz identyfikatorem recenzji w bazie relacyjnej (**rid**).
- Użytkownik, **User**, wraz z unikalnym identyfikatorem (**id**) oraz loginem użytkownika (**login**).

Główną siłą grafowej bazy danych będą powiązania pomiędzy obiektami, czyli krawędzie.

- Gatunek filmu: **(:Movie)-[:IS_GENRE]->(:Genre)**. Jest to relacja wiele do wielu: jeden film może być wielu gatunków, a każdy gatunek może być przypisany do wielu filmów.
- Recenzja filmu: **(:Movie)-[:HAS_REVIEW]->(:Review)**. Jest to relacja wiele do jednego: jeden film może mieć wiele recenzji, ale jedna recenzja może dotyczyć tylko jednego filmu.

- Lubi: **(:User)-[:LIKE]->(:Movie)**. Jest to relacja wiele do wielu: każdy użytkownik może lubić wiele filmów, a każdy film może być lubiany przez wiele użytkowników.
- Nie lubi: **(:User)-[:DOES_NOT_LIKE]->(:Movie)**. Jest to relacja wiele do wielu: jeden użytkownik może nie lubić wielu filmów, a każdy film może być nie lubiany przez wielu użytkowników.
- Oceniał: **(:User)-[:SCORED score]->(:Movie)**. Jest to relacja wiele do wielu: jeden użytkownik może ocenić wiele filmów, a każdy film może być oceniony przez wielu użytkowników. Dodatkowo, w krawędzi przechowujemy informację o tym, na jaką ocenę dany użytkownik ocenił film.
- Wyreżyserował: **(:Person)-[:DIRECTED]->(:Movie)**. Jest to relacja wiele do wielu: jedna osoba mogła wyreżyserować wiele filmów, a każdy film mógł mieć wielu reżyserów.
- Napisał scenariusz: **(:Person)-[:WROTE]->(:Movie)**. Jest to relacja wiele do wielu: jedna osoba mogła napisać scenariusze do wielu filmów, a każdy film mógł mieć wielu scenarzystów.
- Zagrał: **(:Person)-[:STAR role]->(:Movie)**. Jest to relacja wiele do wielu: jedna osoba mogła grać w wielu filmach, a każdy film mógł mieć wielu aktorów. Dodatkowo, w krawędzi przechowujemy informację o roli, którą w filmie odgrywała dana osoba.
- Jest fanem: **(:User)-[:IS_FAN]->(:Person)**. Jest to relacja wiele do wielu. Jeden użytkownik może mieć wielu idoli, a każda osoba może mieć wielu fanów.
- Jest obserwowany: **(:User)-[:FOLLOWED_BY]->(:User)**. Jest to relacja wiele do wielu. Jeden użytkownik może obserwować wielu użytkowników, a także może być obserwowany przez wielu użytkowników.
- Napisał recenzję: **(:User)-[:WROTE_REVIEW]->(:Review)**. Jest to relacja wiele do jednego. Jeden użytkownik mógł napisać wiele recenzji, ale jedna recenzja może mieć tylko jednego autora.

Przykłady zapytań:

- Lista obserwowanych użytkowników:

```
match (n:User) <-[:FOLLOWED_BY]-(m:User)
where n.login="Alicja11"
return m;
```

- Lista gatunków filmu:

```

match (n:Movie)–[IS_GENRE]–>(m:Genre)
where n.mid=1
return m;

```

- Utworzenie nowej osoby:

```

create (n:Person {pid=1})
return n;

```

- Ocenienie filmu:

```

match (n:User),(m:Movie)
where n.login="Alicja11" and m.mid=1
create (n)–[r:SCORE {score: 10}]–(m)
return r;

```

- Usunięcie polubienia filmu:

```

match (n:User)–[l:LIKE]–>(m:Movie)
where n.login="Alicja11" and m.mid=1
delete l;

```

- Zmiana oceny filmu:

```

match (n:User)–[l:SCORED]–>(m:Movie)
where n.login="Alicja11" and m.mid=1
set l.score=2
return l;

```

2.3.3. MongoDB

MongoDB jest systemem zarządzania tekstową bazą danych. Mongo przechowuje dokumenty jako **BSON** (binarny JSON). Pozwala to na łatwe zapisywanie i odczytywanie danych. Dane te nie muszą mieć schematu, dodatkowo dane te można w łatwy sposób skalować i dzielić pomiędzy różnymi serwerami [8].

MongoDB przechowuje dane jako **kolekcje**, czyli zbiory danych określonego typu.

W bazie przechowywać będziemy przede wszystkim historię edycji, wykorzystując wydajność MongoDB podczas zapisów, ale również i dane dotyczące wyszukiwania, które mogą zostać w przyszłości zanalizowane, dzięki czemu system wyszukiwania będzie mógł być ciągle ulepszany i poprawiany. Ulepszanie systemu wyszukiwania być może będzie wymagało dołożenia dodatkowych pól wyszukiwania, ale korzystając z faktu, że nie mamy żadnego schematu, nie będzie to problemem w przyszłości.

Baza MongoDB ma dwie kolekcje: **MongoEditHistory** przechowujące informacje o historii edycji odpowiednich pól, oraz **MongoSearchStats** przechowujące dane o wyszukiwaniu każdego użytkownika.

Przykłady zapytań:

- Wstawianie nowego elementu w historii wyszukiwania:

```
db.MongoSearchStats.InsertOne(
  {text: "Harry Potter", name: "movie", genre: "Fantasy"}
)
```

- Wstawianie nowego elementu w historii wyszukiwania, z więcej niż jednym gatunkiem:

```
db.MongoSearchStats.InsertOne(
  {text: "Harry Potter", name: "movie", genre: {"Fantasy", "Adventure", "Family"}}
)
```

- Wyświetlenie historii edycji tytułu filmu:

```
db.MongoEditHistory.find(
  {id: 1, name: "title"}
)
```

2.4. Podsumowanie

Projektując bazy w ten sposób, na nas spada ciężar wydobywania potrzebnych informacji oraz odpowiedniego zapisywania danych. Podczas tworzenia użytkownika, osoby, filmu czy recenzji, prócz wiersza umieszczanego w bazie relacyjnej, należy także stworzyć odpowiedni wierzchołek w bazie grafowej, przypisując mu jednocześnie odpowiedni klucz główny, by w przyszłości móc powiązać obiekty. Podobnie, podczas usuwania, należy pamiętać o usuwaniu danych z obu baz. Aktualizowanie danych jest dość proste: wszelkimi informacjami zajmuje się baza relacyjna, natomiast wszystkimi relacjami - baza grafowa.

Odczytywanie danych bywa jednak problematyczne. Z bazy grafowej wydobywamy informacje o relacjach, po czym odnajdujemy informacje w bazie relacyjnej. Dla przykładu, jeśli chcielibyśmy wydobyć wszystkich aktorów grających w danym filmie, musimy wykonać dwa zapytania. Na początku, z bazy grafowej wydobądźmy klucze główne aktorów grających w filmie:

```
match (m:Movie)-[l:Star]->(p:Person)
where m.mid=1
return p.pid;
```

Mając już listę kluczy głównych, możemy zająć się wydobyciem imion i nazwisk aktorów:

```
select name, surname from persons
where id=pid;
```

Wydaje się, że jest to niepotrzebna komplikacja, jednak spójrzmy na bardziej skomplikowany scenariusz. Na potrzeby silnika rekomendacji, chcielibyśmy znaleźć wszystkie filmy, w których grała jakaś osoba, która jest idolem pewnego użytkownika, którego obserwuje użytkownik „Alicja11”. Zapytanie w bazie SQL mogłoby wyglądać tak:

```
select title from movies
where id in (select played.mid from users
join follow on follow.follower=users.login
join fan on fan.login = follow.followed
join played p~on fan.pid = played.pid);
```

Podczas, gdy analogiczne zapytania w naszych bazach wyglądają tak:

```
match (n:User)<-[FOLLOWED_BY]-(:User)
-[IS_FAN]->(:Person)-[:Played]->(m.Movie)
where n.login="Alicja11"
return m.mid;
```

```
select title from movies
where id in (mid1,mid2,mid3,...);
```

Zapytania zdają się być krótsze i czytelniejsze, a przy okazji nie musimy się martwić o złożoność operacji **join**. Oczywiście, w przypadku bardziej skomplikowanych zapytań różnica ta się powiększa. Wyszukiwanie długich ścieżek jest znacznie prostsze w bazie grafowej, z czego możemy skorzystać.

Część zapytań nie będzie jednak wymagała od nas odwoływania się do obu baz. Dla przykładu, jeśli chcielibyśmy uzyskać wszystkie gatunki filmu o danym id, wystarczy nam odpytanie bazy grafowej:

```
match (m:Movie)-[:IS_GENRE]->(g:Genre)
where m.mid=1
return g.name;
```

Podobnie, odpytywanie o listę wszystkich filmów/osób/użytkowników wymaga odpytania jedynie bazy relacyjnej:

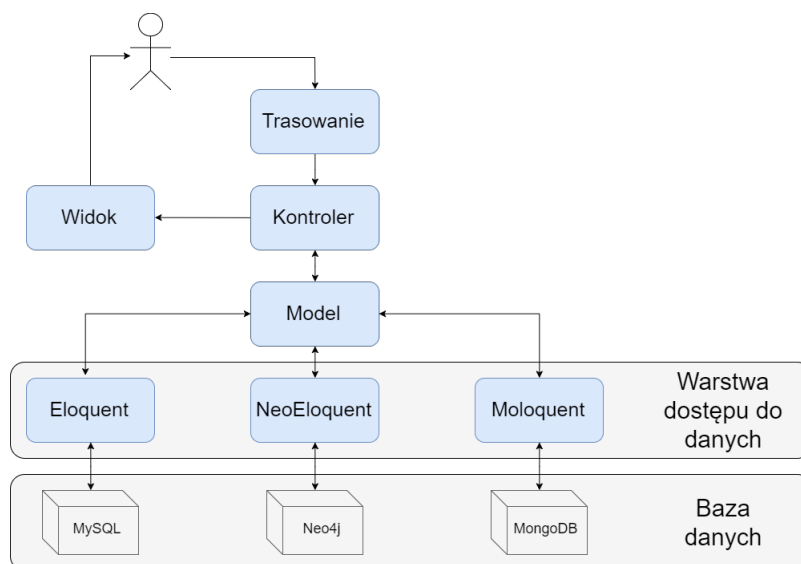
```
select title, photo, prod_date, score from movies;
```

Baza tekstowa zdaje się być prawie całkowicie niezależna: prócz kluczy głównych pochodzących z bazy relacyjnej nie potrzebujemy dodatkowych informacji w historii

edycji czy w statystykach wyszukiwania, tak długo, jak nie potrzebujemy dodatkowych informacji o tym, co tak naprawdę zostało zmienione.

Rozdział 3.

Architektura aplikacji

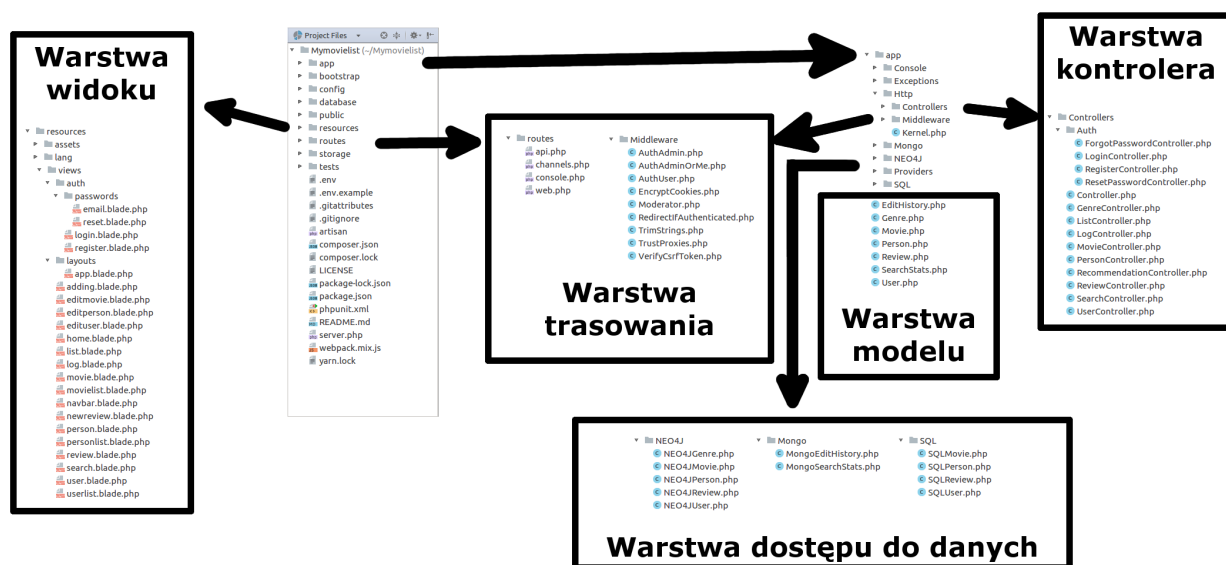


Rysunek 3.1: Architektura aplikacji

Aplikacja została napisana w frameworku Laravel, bazującym na wzorcu MVC. Użytkownik odbiera **widok**, a jego żądania poprzez **trasowanie** kierowane są do **kontrolera**, który odpowiada także za przygotowanie odpowiedzi wysyłanej do przeglądarki. Kontroler porozumiewa się z **modelem**, który przekazuje żądania do odpowiednich warstw **dostępu do danych**, które odpowiadają za komunikację z odpowiadającymi im bazami danych.

3.1. Warstwa widoku

Pliki warstwy widoku umieszczane są w folderze *resources/views*. Laravel wykorzystuje język szablonów **blade**, do dynamicznego generowania widoku. Umożliwia on nie tylko pisanie w języku **html** ale również w **PHP**, na przykład pętli, warunków czy odwołań do zmiennych. Zwyczajowo, część w języku PHP zaczyna się od



Rysunek 3.2: Struktura projektu

znaku @. W przypadku, w którym chcemy wyświetlić coś ze zmiennej, wystarczy napisać `{{ $nazwa_zmiennej }}`. Więcej informacji można znaleźć w [12].

Jedną z głównych korzyści które daje nam używanie szablonów blade, jest możliwość tworzenia szablonów strony, na których można później osadzać podstrony wypełnione odpowiednią treścią. Zmniejsza to konieczność duplikowania kodu. Strony internetowe mają zazwyczaj spójny interfejs - część elementów pozostaje taka sama na wszystkich stronach, zmienia się tylko „wnętrze” takiej strony. Dla przykładu, pasek nawigacyjny na górze strony pozostaje taki sam, niezależnie na której stronie go wyświetlamy.

Szablon główny aplikacji znajduje się w pliku `resources/views/layouts/app.blade.php`. Wskazujemy tam miejsca w których ma znaleźć się pasek nawigacyjny oraz zawartość strony, a także wyświetlamy błędy i wiadomości, co umożliwia komunikację z użytkownikami aplikacji.

W poszczególnych plikach znajdują się wyglądy poszczególnych stron. Dodatkowo, a w aplikacji wykorzystuję bibliotekę **bootstrap** [13], zawierającej definicje wielu klas, które są używane podczas pisania poszczególnych widoków. Dzięki temu możliwe jest łatwe stworzenie strony dobrze wyglądającej na większości urządzeń, bez potrzeby ręcznego konfigurowania stylów **CSS**. Dodatkowo, do biblioteki dołączone są wtyczki języka **Javascript**, dzięki czemu wiele komponentów może działać bez konieczności pisania dodatkowego kodu.

3.2. Warstwa trasowania

Trasowanie odpowiada za przekierowywanie żądań użytkownika do odpowiednich kontrolerów. Całe trasowanie znajduje się w pliku *routes/web.php*. Dodatkowo warstwa ta odpowiada za autoryzację użytkowników, za pomocą **pośredników**. Mamy cztery rodzaje uprawnień: **niezalogowany**, **zalogowany**, **moderator** oraz **administrator**. Wszelkie definicje pośredników umieszczane są w folderze *app/Http/Middleware*, a dokładniej w metodzie **handle**, która sprawdza warunki i wyświetla stronę, jeśli są one spełnione, albo przekierowuje do innej strony w przeciwnym przypadku.

3.3. Warstwa kontrolera

Warstwa kontrolera ma dwie funkcje: po pierwsze wstawia do widoku odpowiednie informacje, a po drugie komunikuje się z modelem. Wszystkie kontrolery znajdują się w pliku *app/Http/Controllers*.

Każdy z kontrolerów odpowiada za inną podstronę. Poniżej znajdują się dokładniejsze opisy każdego kontrolera.

3.3.1. Kontrolery Auth

Kontrolery znajdujące się w folderze *app/Http/Controllers/Auth* służą uwierzytelnianiu użytkowników i zostały wygenerowane automatycznie przez technologię Laravel. Z tego powodu nie będziemy się nimi zajmować.

3.3.2. GenreController

Kontroler ten odpowiada za dodawanie oraz usuwanie gatunków.

3.3.3. RecommendationController

Kontroler ten odpowiada za rekomendacje. Kontroler pobiera odpowiednie dane z modelu i przekazuje gotowe listy do widoku. Możliwe jest usuwanie rekomendacji (zaznaczanie, że film się nie podoba), za które odpowiada funkcja **delete(\$mid)** przyjmująca jako parametr id filmu, a także resetowanie rekomendacji, to znaczy cofnięcie rekomendacji do stanu sprzed jakichkolwiek usunięć (odznaczenie, że film się nie podoba), za co odpowiada funkcja **reset(\$mid)**.

3.3.4. ListController

Kontroler ten odpowiada za operacje pomocnicze na listach. Obecnie, jedyną zaimplementowaną operacją jest możliwość posortowania listy po odpowiedniej kolumnie, w odpowiednim porządku (rosnącym bądź malejącym).

3.3.5. LogController

Kontroler ten odpowiada za stronę logów. Kolejne funkcje pobierają informacje o historiach edycji odpowiednich tabel oraz statystyk wyszukiwania. Funkcja **clear()**yczyści wszystkie bazy, usuwając z nich dane starsze niż 30 dni.

3.3.6. MovieController

Kontroler odpowiadający za operacje na filmach. Mamy tu metodę służącą wyświetlania odpowiednio posortowanej listy filmów (**list**), metodę pobierającą informacje o danym filmie (**movie(\$mid)**), metody służące do dodawania i usuwania filmów (**add** oraz **deleteMovie(\$mid)**), metody służące do edycji danych filmu, które przy okazji zapisują historię edycji, oraz metody służące do oceniania oraz polubienia danego filmu.

3.3.7. PersonController

Kontroler odpowiadający za operacje na osobach. Mamy tu metodę służącą wyświetlania odpowiednio posortowanej listy osób (**list**), metodę pobierającą informacje o danej osobie (**person(\$pid)**), metody służące do dodawania i usuwania osób (**add** oraz **deletePerson(\$mid)**), metody służące do edycji danych osób, które przy okazji zapisują historię edycji, oraz metody służące do polubienia (bądź usunięcia polubienia) danej osoby.

3.3.8. ReviewController

Kontroler ten odpowiada za stronę recenzji oraz stronę tworzenia nowej recenzji. Metoda **review(\$mid,\$rid)** przyjmuje jako parametry id filmu oraz id recenzji, i wyświetla odpowiednie informacje o recenzji. Metoda **newReview(\$mid)** odpowiada za przekazanie informacji do strony tworzenia nowej recenzji, z kolei metody **createReview(\$mid)** oraz **deleteReview(\$mid,\$rid)** odpowiadają kolejno za dodawanie i usuwanie recenzji. Dodatkowo zapisują informacje o poprzedniej wartości odpowiedniego pola, wysyłając dane do modelu odpowiadającego za historię edycji.

3.3.9. SearchController

Kontroler ten odpowiada za stronę wyszukiwania. Kolejne metody odpowiadają za wyszukiwanie, filtrowanie i sortowanie danych pobieranych z odpowiedniego modelu, po czym wysyłają je do widoku.

3.3.10. UserController

Kontroler odpowiadający za operacje na użytkownikach. Mamy tu metodę służącą wyświetlania odpowiednio posortowanej listy użytkowników (**list**), metodę pobierającą informacje o danym użytkowniku (**user(\$login)**), metody służące do dodawania i usuwania użytkowników (**add** oraz **deleteUser(\$login)**), metody służące do edycji danych użytkowników, które przy okazji zapisują historię edycji, oraz metody służące do zaznaczenia użytkowników jako obserwowanych.

3.4. Warstwa modelu

Warstwa modelu pośredniczy pomiędzy warstwą dostępu do danych a warstwą kontrolera. Przede wszystkim warstwa ta zbiera żądania i przekierowuje je do odpowiednich ORM, implementuje też wszystkie operacje wyszukiwania które wykonujemy na bazie danych. Dodatkowo, wprowadza ona warstwę abstrakcji, dzięki czemu wyższe warstwy nie są świadome używanych przez nich technologii baz danych, co wpływa na utrzymywalność rozwiązania. Pliki tej warstwy znajdują się bezpośrednio w folderze *app*.

3.4.1. EditHistory

Klasa odpowiadająca za historię edycji. Jako konstruktor klasa przyjmuje nazwę tabeli, następnie wszystkie zapytania przekierowuje do klasy **MongoEditHistory**.

3.4.2. Genre

Klasa odpowiadająca za gatunki. Jako konstruktor metoda przyjmuje albo samą nazwę gatunku, albo dodatkowo instancję klasy **NEO4JGenre** (jeśli nie została podana, konstruktor tworzy nową). Kolejne metody pozwalają na sprawdzanie istnienia gatunku o danej nazwie, tworzenie nowego gatunku oraz usuwanie istniejącego, a także zwracanie listy wszystkich gatunków i wszystkich filmów o danym gatunku.

3.4.3. Movie

Klasa odpowiadająca za filmy. Dane filmu przechowywane są w bazie relacyjnej, natomiast informacje o aktorach, scenarzystach, reżyserach, gwiazdach, istniejących recenzjach, polubieniach czy gatunkach przechowywane są w bazie grafowej. Za konstruktor przyjmuje id filmu, a także, jeśli jest taka potrzeba, instancje klas **NEO4JMovie** oraz **SQLMovie** (jeśli nie zostały podane tworzy nowe). Kolejne metody umożliwiają odczytywanie poszczególnych informacji, aktualizowanie, usuwanie i dodawanie filmów oraz polubień a także zmianę oceny na podstawie nowych ocen (bądź aktualizacji starych).

3.4.4. Person

Klasa odpowiadająca za osoby. Dane osób przechowywane są w bazie relacyjnej, natomiast informacje w jakich filmach zagrała, jakie wyreżyserowała, do jakich pisała scenariusz jak i informacje o fanach przechowywane są w bazie grafowej. Za konstruktor przyjmuje id osoby, a także, jeśli jest taka potrzeba, instancje klas **NEO4JPerson** oraz **SQLPerson** (jeśli nie zostały podane tworzy nowe). Kolejne metody umożliwiają odczytywanie informacji o osobach, aktualizowanie, usuwanie i dodawanie osób oraz polubień, a także zapisywanie nowych ról, dodawanie osoby jako reżysera czy scenarzystę czy pobieranie listy lub liczby wszystkich fanów.

3.4.5. Review

Klasa odpowiadająca za recenzje. Dane recenzji przechowywane są w bazie relacyjnej, natomiast informacje zarówno o autorze jak i o filmie do którego się odnoszą przechowywane są w bazie grafowej. Za konstruktor przyjmuje id recenzji, oraz dodatkowo może przyjmować instancje klas **NEO4JReview** oraz **SQLReview** (jeśli nie zostały podane tworzy nowe). Kolejne metody pozwalają na sprawdzanie istnienia recenzji o danym id, usuwanie oraz tworzenie recenzji oraz pozyskiwanie danych recenzji, czy informacji o autorze czy filmie do którego recenzja się odwołuje.

3.4.6. SearchStats

Klasa odpowiada za statystyki wyszukiwania. Wszystkie metody przekierowują zapytania do klasy **MongoSearchStats**.

3.4.7. User

Klasa odpowiadająca za użytkowników. Dane użytkowników przechowywane są w bazie relacyjnej, natomiast informacje o napisanych recenzjach, polubionych filmach i osobach, obserwowanych użytkownikach przechowywane są w bazie grafowej.

Za konstruktor przyjmuje login użytkownika, a także, jeśli jest taka potrzeba, instancje klas **NEO4JUser** oraz **SQLUser** (jeśli nie zostały podane tworzy nowe). Kolejne metody umożliwiają odczytywanie informacji o użytkowniku, aktualizowanie, usuwanie i dodawanie użytkowników, polubień, obserwowań, idoli, ocenianie, recenzowanie oraz rekomendowanie filmów. Rekomendacje ustalane są na podstawie czterech zapytań: gatunków lubianych filmów, polubionych osób, filmów lubianych przez obserwowanych użytkowników, oraz filmów, w których grała jakaś osoba, będąca idolem któregoś z użytkowników, którego dany użytkownik obserwuje.

3.5. Warstwa dostępu do danych

Warstwa dostępu do danych odpowiada za komunikację z bazą danych. W większości klasy znajdujące się w tej warstwie implementują odpowiednie ORM, ale możliwe jest umieszczenie w nich zapytań surowych, jeśli zajdzie potrzeba wykonania konkretnego zapytania, którego nie można wykonać (w sposób prosty i efektywny) za pomocą ORM.

3.5.1. Eloquent

Eloquent to ORM dołączony do Laravela, który umożliwia łatwą pracę z relacyjną bazą danych. Po odpowiedniej konfiguracji mamy dostęp do metod, które ułatwiają nam odczytywanie i zapisywanie rekordów [9].

Pliki odpowiadające za komunikację z bazą relacyjną znajdują się w folderze *app/SQL*. Każda z klas nadpisuje metody klasy **Eloquent**, podając nazwę tabeli, rodzaj połączenia oraz pola, które można w danej tabeli wypełniać.

Klasa **SQLUser** przechowuje dodatkowe informacje, takie jak zaszyfrowane hasło oraz specjalny token, który umożliwia zapamiętanie użytkownika na określoną ilość czasu.

3.5.2. NeoEloquent

NeoEloquent to ORM, stworzony na podstawie ORM'a **Eloquent**, który umożliwia łatwą pracę z bazą danych **Neo4j**. Po odpowiedniej konfiguracji mamy dostęp do metod, które ułatwiają nam odczytywanie i zapisywanie rekordów [10].

Pliki odpowiadające za komunikację z bazą grafową znajdują się w folderze *app/NEO4J*. Każda z klas nadpisuje metody klasy **NeoEloquent** podając etykietę wierzchołka, rodzaj połączenia oraz pola, które można w wierzchołku wypełnić.

Dodatkowo, w każdej z klas zdefiniowane są relacje oraz ich nazwy. Dzięki temu możliwe będzie przechodzenie po krawędziach. Więcej informacji można znaleźć w repozytorium NeoEloquent [10].

W klasie **NEO4JUser**, w metodzie **myQuery(\$login)** został podany przykład tworzenia surowych zapytań, w przypadku potrzeby bezpośredniego sformułowania zapytania z pominięciem ORMa. Po wykonaniu zapytania zostaje ono przeparsowane i zwraca kolekcję filmów, w których grała jakaś osoba, której fanem jest jakiś użytkownik, obserwowany przez użytkownika o loginie podanym w parametrze.

3.5.3. Moloquent

Moloquent to ORM, stworzony na podstawie ORM'a **Eloquent**, który umożliwia łatwą pracę z bazą danych **MongoDB**. Po odpowiedniej konfiguracji mamy dostęp do metod, które ułatwiają nam odczytywanie i zapisywanie rekordów [11].

Pliki odpowiadające za komunikację z tekstową bazą danych znajdują się w folderze *app/Mongo*. Każda z klas nadpisuje metody klasy **Model** pochodzącej z plików Moloquent'a, definiując rodzaj połączenia. Ponieważ baza ta nie musi posiadać schematu, nie musimy definiować pól, które możemy wypełniać.

Rozdział 4.

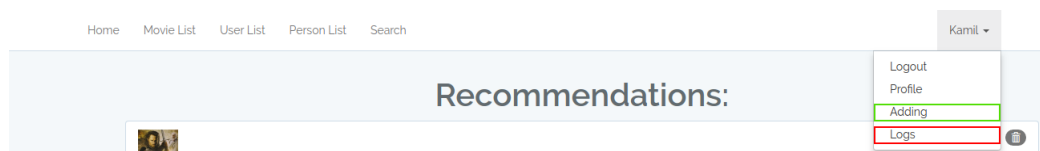
Funkcjonalności

Aplikacja jest serwisem internetowym. Użytkownik może poruszać się po kolejnych stronach przy pomocy odpowiednich odnośników, oraz panelu nawigacyjnego znajdującego się na górze strony. Podczas wyświetlania odpowiednich list istnieje możliwość posortowania ich według pewnej wartości, robi się to za pomocą przycisków znajdujących się obok nazw odpowiednich kolumn.

4.1. Pasek nawigacyjny



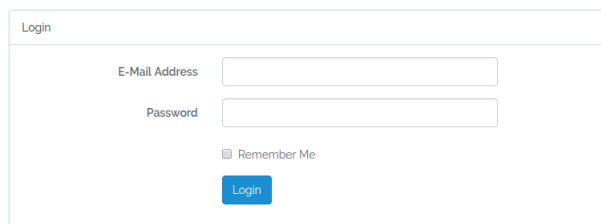
Rysunek 4.1: Pasek nawigacyjny gościa



Rysunek 4.2: Pasek nawigacyjny użytkownika

Pasek nawigacyjny służy do nawigowania po stronie. Po lewej stronie paska znajdują się poszczególne podstrony. W przypadku niezalogowanego użytkownika, w prawym rogu paska nawigacyjnego znajdują się odnośniki do strony logowania a także rejestracji. W wypadku zalogowanego użytkownika, po naciśnięciu loginu w prawym rogu paska, ukazują się dodatkowe opcje umożliwiające wylogowanie i dostęp do własnego profilu, a także, jeśli użytkownik ma do tego uprawnienia, zaznaczony na zielono odnośnik do strony dodawania (moderator) oraz zaznaczony na czerwono odnośnik do strony logów (administrator).

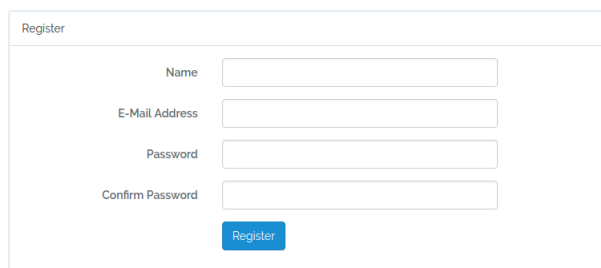
4.2. Strona logowania

A login form titled "Login" with a light blue border. It contains two input fields: "E-Mail Address" and "Password". Below the password field is a checkbox labeled "Remember Me". At the bottom is a blue button with the text "Login" in white.

Rysunek 4.3: Panel logowania

Strona logowania umożliwia niezalogowanemu użytkownikowi zalogowanie się. W tym celu należy podać adres email oraz hasło zarejestrowanego użytkownika. W przypadku niepowodzenia, zwracany jest odpowiedni komunikat. Możliwe jest także zaznaczenie opcji „zapamiętaj mnie” która sprawia, że użytkownik nie zostanie automatycznie wylogowany po określonym czasie.

4.3. Strona rejestracji

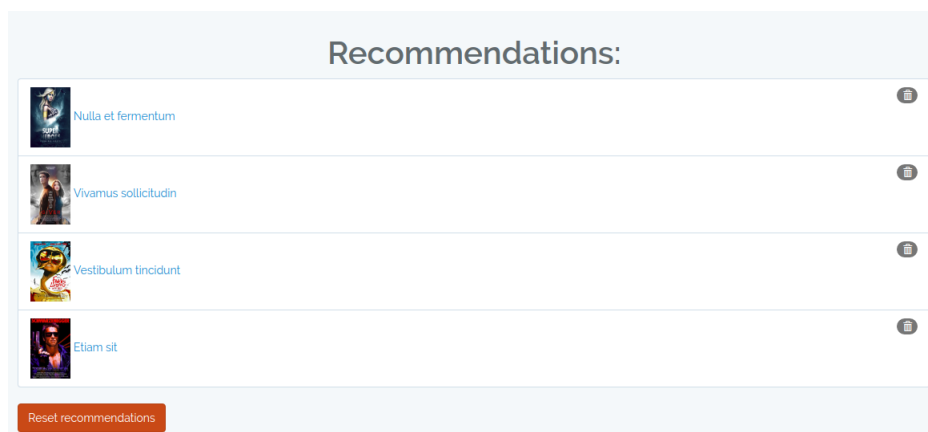
A registration form titled "Register" with a light blue border. It contains four input fields: "Name", "E-Mail Address", "Password", and "Confirm Password". At the bottom is a blue button with the text "Register" in white.

Rysunek 4.4: Panel rejestracji

Strona rejestracji umożliwia użytkownikowi zarejestrowanie się. By to zrobić należy podać nazwę użytkownika oraz email, które muszą być unikalne, a także wpisać i potwierdzić hasło. Pierwszy zarejestrowany użytkownik dostaje automatycznie uprawnienia administratora.

4.4. Strona domowa

Strona domowa jest stroną spersonalizowaną pod każdego użytkownika. Wyświetlane są na niej rekomendacje filmów, na podstawie wcześniejszej aktywności użytkownika. Możliwe jest usuwanie rekomendacji, za pomocą przycisku w prawym górnym rogu odpowiedniego wiersza, a także zrestartowanie rekomendacji, czyli przywrócenie wszystkich usuniętych rekomendacji, za pomocą przycisku pod wszystkimi rekomendacjami.










Rysunek 4.5: Przykładowa strona domowa

4.5. Listy

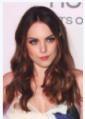


Na stronach z tej grupy wyświetlane są odpowiednie listy. Każdą z list można posortować po każdej z kolumn: służą do tego przyciski umieszczone obok nazwy kolumny.

4.5.1. Lista filmów

Title 		Production date 	Score 
	Class aptent taciti	2004-02-05	9.75
	Nulla et fermentum	2008-05-05	9.33
	Lorem ipsum	2003-01-01	8.85
	Phasellus a sociis	2010-01-12	8.75

Rysunek 4.6: Lista filmów

Strona zawierająca listę wszystkich filmów. Każdy z wierszy listy filmów, oprócz nazwy i obrazka filmu, zawiera także informacje o dacie produkcji, oraz oceny, obliczanej na podstawie ocen wszystkich użytkowników. Po naciśnięciu na tytuł interesującego nas filmu, zostajemy przekierowani do strony tego filmu.





Name 🚩🚩🚩		Surname 🚩🚩🚩	Number of Fans 🚩🚩🚩
 Elizabeth	Gillies		8
 Alton	Manning		2
 Dominic	Prestwood		3

Rysunek 4.7: Lista osób

4.5.2. Lista osób

Strona zawierająca listę wszystkich osób. Każdy z wierszy listy osób, oprócz zdjęcia, imienia i nazwiska osoby, zawiera także informację o liczbie fanów danej osoby. Po naciśnięciu na imię bądź nazwisko interesującej nas osoby, zostajemy przekierowani do strony tej osoby.

4.5.3. Lista użytkowników

Login 🚩🚩🚩	
 Arsenicro	
 AliceYoung	
 Whirthy	
 Puzzledbvtcode	

Rysunek 4.8: Lista użytkowników

Strona zawierająca listę wszystkich użytkowników. Każdy z wierszy listy użytkowników zawiera jej avatar oraz login. Po naciśnięciu na login interesującego nas użytkownika, zostajemy przekierowani na profil tego użytkownika.

Rysunek 4.9: Wyszukiwarka

4.6. Wyszukiwarka

Wyszukiwarka umożliwia nam wyszukiwanie filmów, osób bądź użytkowników. W przypadku filmów możliwe jest też wybranie gatunku filmu (bądź wybranie tylko gatunku filmu) oraz, w wypadku osób zalogowanych, zaznaczenie, czy pragniemy pokazywać także filmy które już obejrzelismy. Po kliknięciu przycisku „Search”, pod wyszukiwarką pojawia się lista obiektów, spełniających wybrane przez nas kryteria. W przypadku wyszukiwania bez kryteriów, dostajemy pełną listę osób/filmów/użytkowników.

4.7. Strona filmu

Rysunek 4.10: Strona filmu



Rysunek 4.11: Ikona usunięcia polubienia

Strona zawierająca informacje o filmie. Strona filmu podzielona jest na dwie części. Z lewej strony, pod obrazkiem filmu, prócz daty produkcji, dostajemy także

listę recenzji. Po naciśnięciu przycisku obok napisu „Reviews” zostajemy przekierowani do strony tworzenia nowej recenzji, chyba, że użytkownik nie jest zalogowany, bądź napisał już recenzję do danego filmu. Każdy kolejny rekord zawiera informację o tym, kto jest autorem recenzji, a po prawej stronie odnośnik do samej recenzji.




Z prawej strony, na samej górze, mamy informację o tytule filmu, a także ikony edycji oraz polubienia (bądź usunięcia polubienia). Po naciśnięciu przycisku edycji, jeśli mamy ku temu uprawnienia, zostajemy przekierowani do strony edycji.

Poniżej tytułu filmu znajdują się informacje na temat ogólnej oceny filmu, liczby użytkowników która oceniła ten film, oceny zalogowanego użytkownika (N/A w wypadku braku oceny bądź użytkownika niezalogowanego) oraz listę gatunków filmu oddzieloną przecinkami. Istnieje możliwość edycji oceny (o ile jesteśmy użytkownikiem zalogowanym) poprzez wybranie z listy ocen interesującej nas wartości, i naciśnięciu przycisku dyskietki znajdującej się poniżej.

Następną rzeczą jest opis filmu, a także lista reżyserów oraz scenarzystów, oddzielonych przecinkami. Po naciśnięciu na imię i nazwisko reżysera bądź scenarzysty, zostajemy przekierowani na odpowiadającą mu stronę osoby.

Niżej znajduje się spis wszystkich osób, które grały w filmie. Prócz zdjęcia osoby, mamy także informację o roli, oraz imię i nazwisko osoby. Po naciśnięciu imienia i nazwiska aktora, zostajemy przekierowani na odpowiadającą mu stronę osoby.

4.7.1. Strona edycji filmu

ID: 1  	
Title	<input type="text" value="Lorem ipsum"/>
Production date	<input type="text" value="2003-01-01"/>
Image	<input type="text" value="/img/Lorem ipsum.jpg"/>
Description	<div>Gandalf and Aragorn lead the 'World of Men' against Sauron's army to draw his gaze from Frodo and Sam as they approach Mount Doom with the One Ring.</div>
	

Rysunek 4.12: Strona edycji filmu

Strona służąca do edycji danych filmu a także do edycji obsady, reżyserów oraz scenarzystów. Na samej górze strony edycji filmu widnieje informacja o ID filmu. Ikony obok ID służą do cofnięcia się na stronę filmu oraz całkowite usunięcie filmu.

Kolejne wiersze umożliwiają edycje poszczególnych pól. Edycję należy zakończyć naciskając ikonkę dyskietki, znajdującą się bezpośrednio pod ostatnią z kolumn.

The screenshot shows a web interface for editing film information. At the top, there's a 'Genres' section with a light blue bar containing 'Action', 'Drama', and 'Fantasy', each with a blue 'x' icon, followed by an empty input field and a '+' icon. Below this, the 'Credits' section shows 'Directed by: Harold Richards' and 'Wrote by: Sofia Murphy', both with blue 'x' icons and empty input fields with '+' icons. The 'Cast' section is below, with a title 'Cast' in the center. It lists two cast members: 'Praesent' with a small profile picture and 'Sofia Murphy' with a blue 'x' icon, and 'Aliquam' with a small profile picture and 'Alex Thomas' with a blue 'x' icon. At the bottom, there are two input fields: 'Role:' and 'Person ID:', both with '+' icons.

Rysunek 4.13: Dalsze możliwości edycji

Poniżej informacji ogólnych znajdują się okna umożliwiające dodawanie gatunków, reżyserów, scenarzystów oraz aktorów. Każde pojedyncze dodanie należy potwierdzić naciskając ikonę plusa, znajdującą się bezpośrednio obok pola, które właśnie wypełniliśmy. Dodatkowo, obok każdego z gatunków/scenarzystów/reżyserów/aktorów znajduje się ikonka krzyżyka, która pozwala nam na usunięcie danego rekordu.

4.7.2. Strona recenzji

The screenshot shows a movie review page. On the left is a movie poster for 'The Lord of the Rings'. Below the poster, it says 'Production date: 2003-01-01' and has a 'Details' link. On the right, there's a review section. At the top, it shows the reviewer's avatar, name 'AliceYoung', a rating '9/10', and the date '2018-02-02 16:45:11'. Below this is a placeholder text: 'Cras ultricies nisl id euismod rutrum. Praesent sit amet neque non odio bibendum fermentum.' There's a trash icon to the right of the date.

Rysunek 4.14: Przykładowa recenzja

Strona zawierająca dane recenzji. Strona recenzji podzielona jest na dwie części. Po lewej stronie znajduje się obrazek filmu, pod którym mamy informację o dacie produkcji oraz odnośnik do strony filmu.

Po prawej stronie, na górze, mamy informację o autorze recenzji - avatar oraz login, będący odnośnikiem do profilu, ocenę filmu autora recenzji a także przycisk do usunięcia recenzji, dla administratorów i moderatorów. Pod tymi informacjami umieszczony jest sam tekst recenzji.

4.7.3. Strona dodawania recenzji



Rysunek 4.15: Dodawanie recenzji

Strona służąca do dodawania recenzji. Na górze strony mamy informację o tym, jakiego filmu recenzję piszemy. Obok tej informacji są ikony powrotu i zapisu. Pod nimi znajduje się pole tekstowe, w którym wpisuje się tekst recenzji. Po wpisaniu tekstu należy potwierdzić jej zapisanie przyciskiem.

4.8. Strona osoby



Rysunek 4.16: Strona osoby



Rysunek 4.17: Przycisk usunięcia obserwacji

Strona zawierająca informacje o osobie. Strona podzielona jest na dwie części. Po lewej stronie znajduje się zdjęcie osoby, pod którym mamy informację o jej dacie urodzin.

Po prawej stronie, obok imienia i nazwiska osoby, są przyciski edycji oraz obserwacji (lub usunięcia obserwacji), a pod nimi - biografia osoby. Następnie mamy

kolejne tabele, zawierające informacje o filmach, w których dana osoba wystąpiła (wraz z rolami które zagrała) a także filmy w których osoba była reżyserem oraz scenarzystą. Po naciśnięciu na tytuł filmu zostaniemy przekierowani na stronę filmu.

4.8.1. Strona edycji osoby

Name	Elizabeth
Surname	Gillies
Birthday	1993-07-23
Image	/img/Elizabeth.jpg
Biography	Elizabeth Gillies (born July 26, 1993) is an American actress, singer and songwriter. Elizabeth began acting professionally at the age of 12.

Rysunek 4.18: Strona edycji osoby

Strona służąca do edycji danych osoby. Na górze strony znajduje się informacja o ID filmu. Ikony obok ID służą kolejno do cofnięcia się na stronę osoby, oraz całkowitego usunięcia osoby.

Wiersze, niżej, umożliwiają nam edytowanie poszczególnych informacji. Wszelkie zmiany należy potwierdzić naciskając ikonę dyskietki, znajdującą się bezpośrednio pod ostatnią z kolumn.

4.9. Profil użytkownika

Reviews		
	Mauris imperdiet	2/10
	Phasellus a sagittis	9/10
	Cras interdum	4/10
	Vivamus iaculis	7/10
	Donec sed ornare	3/10

Rysunek 4.19: Nasz profil



Rysunek 4.20: Obserwacja i usunięcie obserwacji



Rysunek 4.21: Ikony płci

Strona profilu użytkownika podzielona jest na dwie części. Po lewej stronie znajduje się avatar użytkownika, pod którym są jej dane: imię, nazwisko oraz data urodzenia.

Po prawej stronie, mamy login użytkownika. Po jego lewej stronie znajduje się ikonka płci, a po prawej - ikony edycji oraz obserwacji (bądź usunięcia obserwacji). W przypadku przeglądania własnego profilu, ikona obserwacji nie jest widoczna.

Pod loginem użytkownika, znajduje się jego adres email, a pod nim - opis użytkownika (pole „about”).

Pod opisem znajduje się lista recenzji napisanych przez użytkownika. Lista ta zawiera tytuł filmu (będącym odnośnikiem do strony filmu), ocenę tego filmu oraz przycisk, będący odnośnikiem do strony samej recenzji.

4.9.1. Strona edycji użytkownika

AliceYoung	
Name	<input type="text" value="Alicja"/>
Surname	<input type="text" value="Young"/>
Birthday	<input type="text" value="1995-10-10"/>
Avatar	<input type="text" value="/img/AliceYoung.png"/>
About	<input type="text" value="Jestem Alicja!"/>
Gender	<input type="text" value="Female"/>

Rysunek 4.22: Strona edycji użytkownika

Access:	<input type="text" value="Moderator"/>
---------	----------------------------------------

Rysunek 4.23: Dodatkowe pole gdy zalogowany użytkownik jest administratorem

Strona służąca do edycji danych użytkownika. Na górze znajduje się login, obok którego umieszczone są ikony powrotu oraz usunięcia użytkownika.

Wiersze, znajdujące się poniżej, umożliwiają nam edytowanie poszczególnych informacji. W wypadku, gdy zalogowany użytkownik jest administratorem, dostaje on dostęp do edycji pola, które pozwala na zmianę uprawnień użytkownika. Wszelkie zmiany należy potwierdzić naciskając przycisk dyskietki, znajdującym się bezpośrednio pod ostatnim wierszem.

4.10. Strona dodawania

The image shows a web interface with four distinct sections for adding or deleting data. Each section has a title, a label for the input field, the input field itself, and a confirmation button.

- Add movie:** A single input field labeled 'Movie title:' with a 'Save' button below it.
- Add person:** Two input fields labeled 'Person name:' and 'Person surname:'. A 'Save' button is located below the second field.
- Add genre:** A single input field labeled 'Genre name:' with a 'Save' button below it.
- Delete genre:** A single input field labeled 'Genre name:' with a 'Delete' button below it.

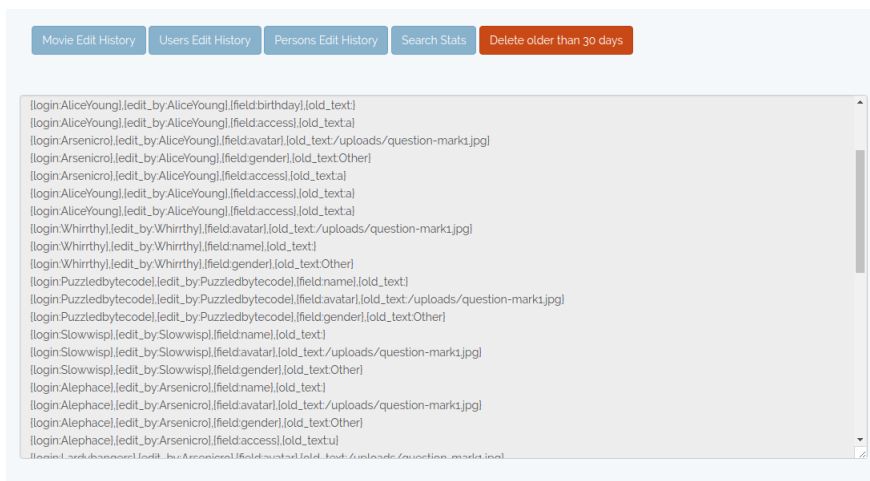
Rysunek 4.24: Strona dodawania

Strona dodawania umożliwia administratorom oraz moderatorom dodawanie nowych filmów, osób oraz gatunków, a także usuwanie gatunków. By dodać film, osobę lub gatunek należy wypełnić odpowiednie pola i potwierdzić przyciskiem „Save” znajdującym się bezpośrednio pod wypełnionym przez nas polem (bądź pod drugim wypełnionym przez nas polem, jeśli dodajemy osobę). Podobnie, by usunąć gatunek, należy wpisać jego nazwę i potwierdzić przyciskiem „Delete”.

4.11. Strona logów

Strona logów pozwala na wyświetlenie informacji o edycji poszczególnych pól a także o historii wyszukiwania wszystkich osób korzystających z aplikacji, a także na usunięcie wszystkich rekordów starszych niż 30 dni.

Po naciśnięciu odpowiedniego przycisku oznaczonego kolorem niebieskim, dostajemy informacje o edytowanych polach bądź wyszukiwaniach. Naciśnięcie czer-



Rysunek 4.25: Strona logów i przykładowe informacje

wonego przycisku sprawia, że wszystkie logi starsze niż 30 dni zostaną bezpowrotnie usunięte.

Rozdział 5.

Instalacja aplikacji

5.1. Wymagania

By aplikacja działała poprawnie, należy spełnić następujące wymagania:

- System Linux
- Zainstalowany **Laravel** [14] wraz z dostępem do poleceń *php artisan*
- Zainstalowany **Composer** [15] wraz z dostępem do poleceń *composer*
- Zainstalowany **NPM** [16] wraz z dostępem do poleceń *npm*
- Zainstalowany **MySQL** [17] wraz z pustą bazą danych
- Zainstalowany **Neo4j** [18] z pustą bazą danych
- Zainstalowany **MongoDB** [19] wraz z pustą bazą danych

5.2. Instalacja i uruchamianie

By aplikacja działała poprawnie, MySQL, MongoDB oraz Neo4j muszą być uruchomione.

W pliku *.env* znajduje się konfiguracja aplikacji. Ze względów bezpieczeństwa, plik ten nie jest załączony, załączony jest natomiast przykład, jako plik *.env.example*. Należy skopiować jego zawartość do pliku *.env* a następnie wypełnić odpowiednie dane dostępu do baz, zgodnie z przykładem. Parametr **DELETE_OLDER_THAN_DAYS** pozwala na zmodyfikowanie czasu, po którym logi zostają usunięte, po naciśnięciu odpowiedniego przycisku.

Każda aplikacja do poprawnego działania potrzebuje klucza, który można wygenerować poleceniem *php artisan key:generate*.

Po poprawnym skonfigurowaniu aplikacji, należy pobrać odpowiednie zależności. Można to zrobić za pomocą poleceń *npm run dev* oraz *composer install*.

W tym momencie aplikacja powinna być gotowa do działania. Żeby uruchomić serwer należy wpisać polecenie *php artisan serv*. Od teraz dostęp do aplikacji możemy uzyskać poprzez przeglądarkę internetową, wpisując odpowiedni adres **ip**, podany w pliku konfiguracyjnym.

Pierwsze zarejestrowane konto automatycznie otrzymuje uprawnienia administratora. W razie problemów, należy w relacyjnej bazie danych, ręcznie zmienić uprawnienia użytkownika (**a** oznacza administratora, **m** oznacza moderatora a **u** oznacza użytkownika).

Bibliografia

- [1] https://www.service-architecture.com/articles/database/relational_model_concepts.html
- [2] <https://www.techwalla.com/articles/what-are-the-advantages-of-a-relational-database-model> 2015
- [3] Graph Databases, 2nd Edition - New Opportunities for Connected Data, Ian Robinson, Jim Webber, Emil Eifrem, June 2015
- [4] <https://neo4j.com/blog/the-top-10-ways-to-get-to-know-neo4j/>
- [5] <https://neo4j.com/developer/cypher-query-language/>
- [6] <https://www.mongodb.com/document-databases>
- [7] <http://bsonspec.org/>
- [8] <https://www.mongodb.com/mongodb-architecture>
- [9] <https://laravel.com/docs/5.5/eloquent>
- [10] <https://github.com/Vinelab/NeoEloquent>
- [11] <https://github.com/jenssegers/laravel-mongodb>
- [12] <https://laravel.com/docs/5.5/blade>
- [13] <https://getbootstrap.com/>
- [14] <https://laravel.com/docs/5.5>
- [15] <https://getcomposer.org/>
- [16] <https://www.npmjs.com/>
- [17] <https://www.mysql.com/>
- [18] <https://neo4j.com/>
- [19] <https://www.mongodb.com/>