



MAC - PROJET

Implémentation d'un réseau social axé sur la cinématographie



JANUARY 19, 2020

FRÉDÉRIC KORRADI, SIMON JOBIN, SIMON FLUCKIGER

Introduction

- Ce projet a pour but la réalisation d'une application logicielle réseau social client-serveur permettant l'acquisition d'informations relatives à des films et le partage d'informations entre amis

Descriptif du projet et points clés

- Client telegram interactif
- Back-end C# .NET Core 3.1
- Utilisation d'une API public pour récupération des informations relatives aux films, puis stockage des résultats des requêtes dans une cache par le serveur, lorsqu'un utilisateur effectue une requête identique à un autre utilisateur, le serveur ira chercher le résultat dans sa cache.
 - o Cache: Base de données document MongoDB, le stockage des données récupérées dans l'api en json permet le stockage direct sous forme de « document » json dans MongoDB
 - Permet une évolutivité des champs contenus dans les documents
 - Permet une amélioration des performances non-négligeable
 - o API: TMDb (<https://www.themoviedb.org/documentation/api>), récupération des films sous forme d'un tableau de films en json
- Gestion des données du réseau social au travers d'une base de données graphe Neo4J, chaque utilisateur a des relations avec des films
 - o L'utilisation d'une base de données graphe facilite la gestion des liens entre les utilisateurs et les films et permet l'utilisation d'un algorithme de parcourt de graphe simple pour définir les relations de proches en proches de manière optimisée
- Implémentation d'un système de suggestions de films, celui-ci se base sur les notations relatives des films effectuées par chacun des *amis* (proches ou éloignées) d'un utilisateur X
 - o Définition d'une pondération logarithmique des notations d'un film en fonction de la profondeur d'amitié des amis de l'utilisateur ayant notés le film
 - o Chaque film se voit affecté un score s utilisé pour la génération d'un classement de films personnalisé. La pondération effectuée de manière logarithmique nous permet de diminuer significativement (i.e. de manière logarithmique) l'importance des votes des utilisateurs en fonction de leur *degré* d'amitié (i.e. le degré d'amitié entre deux personnes est défini par $k = \text{nombre de relations d'amitié minimum entre deux personnes (plus court chemin)} + 1$). La démonstration des résultats de l'implémentation de l'algorithme est proposé plus loin dans le document.

$$S = \frac{\sum_{k=0}^{depth} 2^{depth-k} \cdot \bar{x}_k}{\sum_{k=0}^{depth} 2^{depth-k} \cdot \{1 \text{ si au moins une note au nv } k, 0 \text{ sinon}\}} \quad | \quad x_k = \text{moyenne des notes d'un film pour les amis de profondeur } k \text{ de l'utilisateur}$$

- Architecture propre, tests unitaires

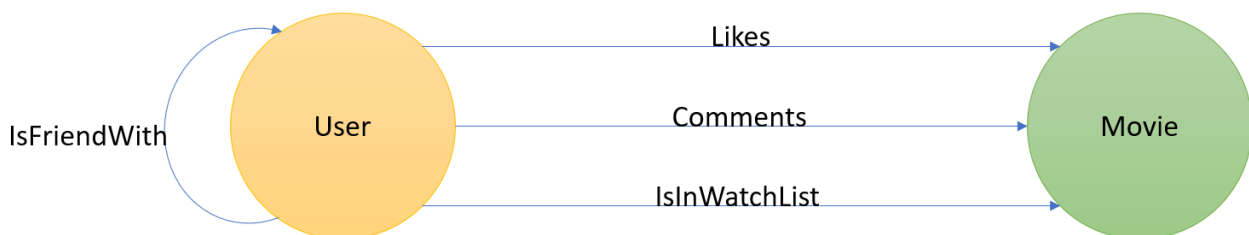
Fonctionnalités implémentées

- Recherche d'un film
 - o Par nom
 - o Par année
 - o Par genre
- Ajout d'un ami
- Ajout d'un film à la *watchlist*

- Récupération d'un ami
- Commentaire d'un film
- Visualisation des commentaires d'un film
- Notation d'un film
- Visualisation des notes d'un film (attribuées par des utilisateurs du réseau social)
- Visualisation de la *watchlist* d'un ami
- Affichage d'une suggestion de films par utilisateur
 - o Affichage d'un classement des films ayant eu la meilleure moyenne pondérée en fonction des notations des amis de l'utilisateur
- Utilisation de la cache si requête déjà exécutée

Model de données

Graphe (Neo4J)



Document (MongoDB)

movie document

```

{
  "popularity": 493.891,
  "vote_count": 1501,
  "video": false,
  "poster_path": "/xBHvZcjRiWyobQ9kxBh06B2dtRI.jpg",
  "id": 419704,
  "adult": false,
  "backdrop_path": "/p3TCqUDoVsrIm8fHK9KOTfWnDjZ.jpg",
  "original_language": "en",
  "original_title": "Ad Astra",
  "genre_ids": [
    12,
    ...
  ],
  "title": "Ad Astra",
  "vote_average": 6,
  "overview": "The near future, ...",
  "release_date": "2019-09-17"
}
  
```

Requêtes avancées

- Notre système comportant trois sources de données (i.e. Neo4JS, MongoDB, tmdbAPI), chacun de ces systèmes contient des requêtes qui lui sont propres.
- Neo4JS :
 - o Définition d'une relation désignant le souhait d'un utilisateur de regarder un film :

```

public void UserWillwatchMovie(string username, string originalTitle)
{
    using (var session = _driver.Session())
    {
        session.Run("MATCH (user:User {username : '" + username
            + "'}), (movie:Movie {originalTitle : '"
            + originalTitle + "'}) MERGE (user)-[:TO_WATCH]->(movie)");
    }
}

```

- Récupération d'une liste de suggestions de films résultant de la moyenne pondérée de la notation des films par les *amis* d'un utilisateur tel que cité dans le descriptif du projet

```

internal void GetSuggestedMovies(long tId, int depth, Dictionary<string, RatedMovie> moviesRatings)
{
    if (depth > 0)
    {
        using (var session = _driver.Session())
        {
            // look for all neighbor friends
            var friends = session.Run("MATCH (a)-[:IS_FRIEND]->(b) WHERE a.tId = "
                + tId + " RETURN b.tId");

            foreach (var friend in friends)
            {
                // complete the map movie - rating
                var movies = session.Run("MATCH (u)-[:RATES]->(m) WHERE u.tId = "
                    + (long)friend["b.tId"]
                    + " RETURN m.originalTitle, r.rating");

                GetSuggestedMovies((long)friend["b.tId"], depth - 1, moviesRatings);

                if (movies != null)
                {
                    foreach (var movie in movies)
                    {
                        if (!moviesRatings.ContainsKey((string) movie["m.originalTitle"]))
                        {
                            moviesRatings.Add((string) movie["m.originalTitle"], new RatedMovie(depth));
                        }

                        // Add the rating to the sum
                        moviesRatings[(string) movie["m.originalTitle"]].RatingsAtDepthLevel[depth - 1, 0]
                            += Convert.ToDouble(movie["r.rating"]);

                        // increment nbRating at depth level for the movie
                        moviesRatings[(string)movie["m.originalTitle"]].RatingsAtDepthLevel[depth - 1, 1] += 1;
                    }
                }
            }
        }
    }
}

```

- TmdbAPI

- Création d'une méthode C# d'abstraction d'un *query builder*, permet d'utiliser les opérateurs *AND* et *OR* lors du passage des paramètres d'une requête. Cette méthode permet une recherche utilisant le *Modèle Booléen* à plusieurs paramètres à l'aide des deux opérateurs précités, celle-ci une fois construite effectuera un appel dans l'api.

```
//multiples values can be used in the filter, use operators and others values to build the filter query
4 references | 1/1 passing | Simon Flückiger, 17 days ago | 1 author, 2 changes | 0 exceptions
public string GetMoviesByFilter(Filter filter, string value, Operator? op = null, string value2 = null, Operator? op2 = null, string value3 = null,
    Operator? op3 = null, string value4 = null, Operator? op4 = null, string value5 = null, Operator? op5 = null, string value6 = null)
{
    return ExecuteRequest("/discover/movie", "&" + FilterToString(filter) + "=" + value +
        (op is null ? "" : (op == Operator.AND) ? "," : "|") + value2 + (op2 is null ? "" : (op2 == Operator.AND) ? "," : "|") + value3 +
        (op3 is null ? "" : (op3 == Operator.AND) ? "," : "|") + value4 + (op4 is null ? "" : (op4 == Operator.AND) ? "," : "|") + value5 +
        (op5 is null ? "" : (op5 == Operator.AND) ? "," : "|") + value6);
}

//getRequest example: /movie/123
//additionalParams example: &language=en-US&query=harry potter
3 references | Simon Flückiger, 21 days ago | 1 author, 1 change | 0 exceptions
private string ExecuteRequest(string getRequest, string additionalParams = "")
{
    string html = string.Empty;
    string url = _apiUrl + getRequest + "?api_key=" + _apiKey + additionalParams;

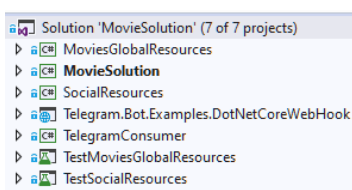
    HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
    request.AutomaticDecompression = DecompressionMethods.GZip;

    using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())
    using (Stream stream = response.GetResponseStream())
    using (StreamReader reader = new StreamReader(stream))
    {
        html = reader.ReadToEnd();
    }
    return html;
}
```

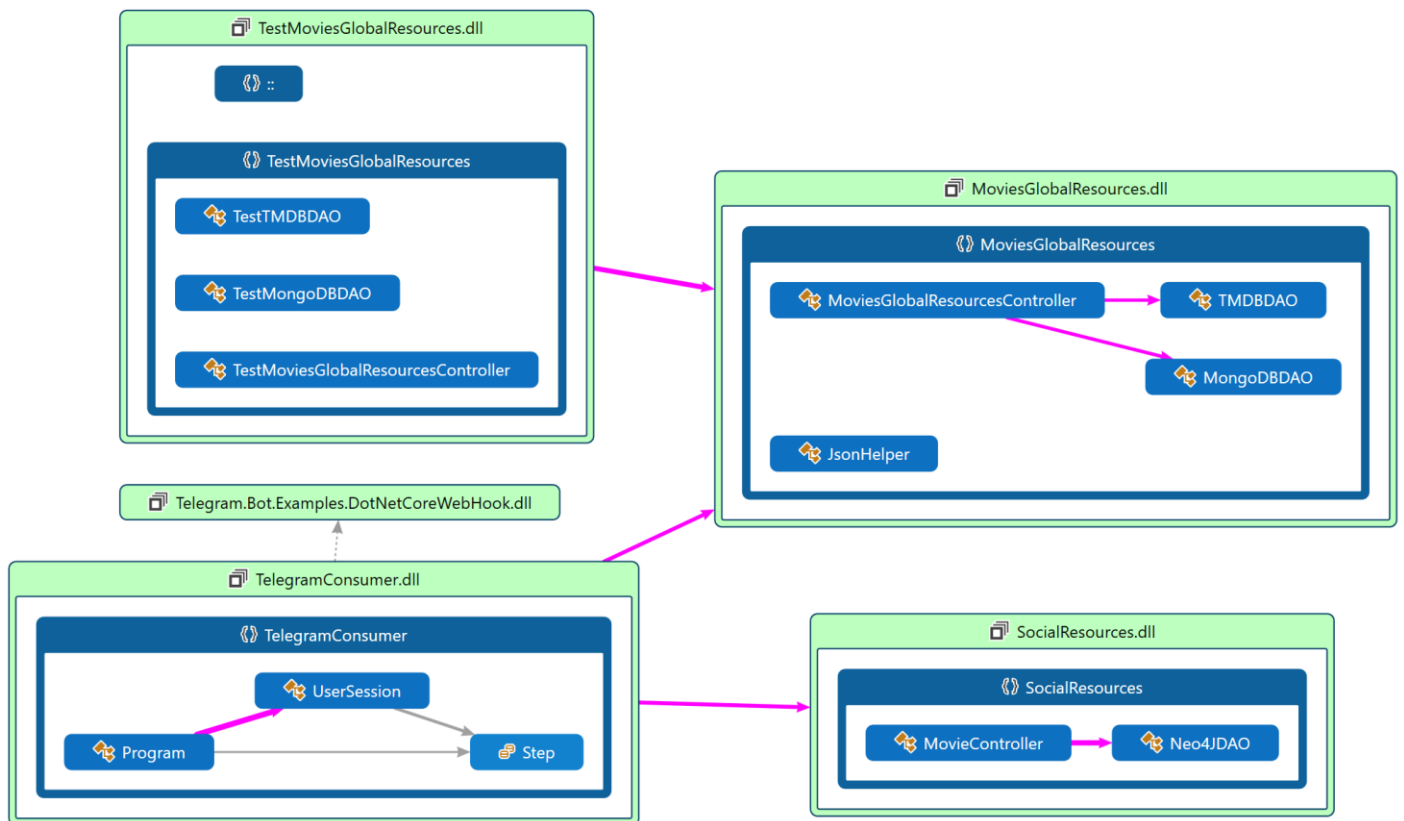
- MongoDB
 - o Utilisation des librairies *MongoDB.Driver* et *MongoDB.Bson* , les objets *Bson* représentent un document MongoDB formaté en json dans une entité C# . L'exemple ci-dessous est la sélection des films par année

```
internal List<BsonDocument> GetMoviesByYear(int year)
{
    var builder = Builders<BsonDocument>.Filter;
    var filter = builder.Gte("release_date", year + "-01-01") & builder.Lte("release_date", year + "-12-31");
    return PrepareResult(_movieCollection.Find(filter).ToList());
}
```

Architecture du code

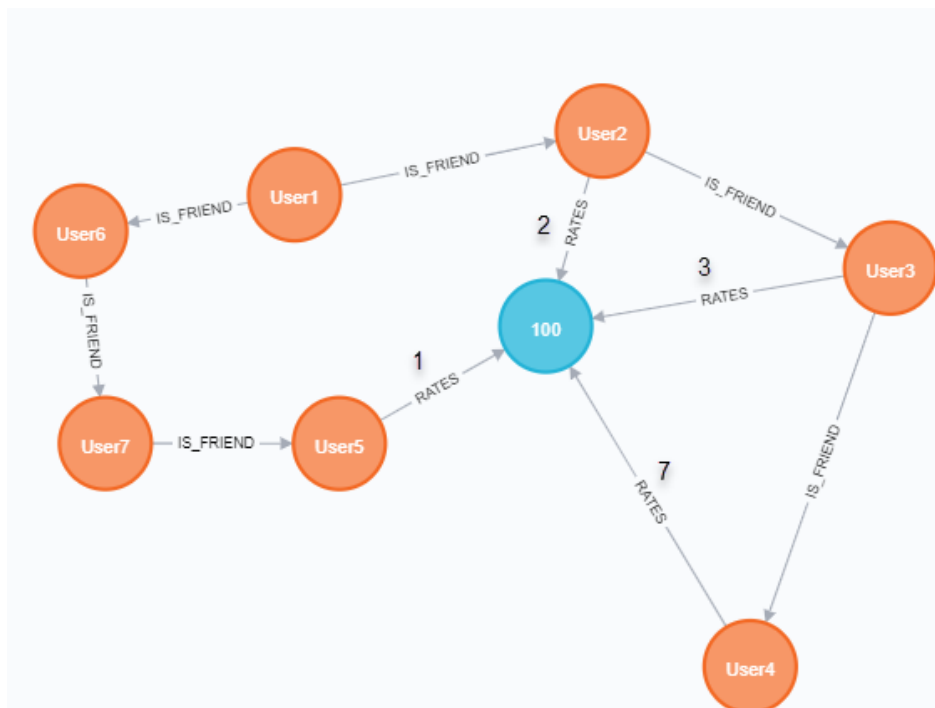


- MoviesGlobalResources : projet contenant la logique et le traitement des données de films globales (API TMDB + cache MongoDB)
- SocialResources : projet contenant la logique et le traitement des données du réseau social (Neo4J graphe DB)
- Telegram.Bot.Examples.DotNetCoreWebHook : web hook library pour *Telegram*
- TelegramConsumer : projet faisant office d'*agent* allant consommer la queue de messages du bot *Telegram* et routant les appels aux couches backend
- TestMoviesGlobalResources : projet de test unitaire pour le projet *MoviesGlobalResources*
- TestSocialResources : projet de test unitaire pour le projet *SocialResources*

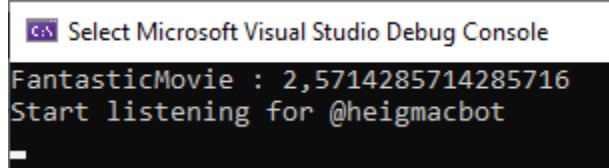


Démonstration de l'algorithme de suggestion de films

- L'appel de la méthode *GetSuggestedMovies()* (c.f. chapitre des requêtes avancées) permet l'obtention de la liste des films suggérés. Ci-dessous est la démonstration du fonctionnement du scoring du film "FantasticMovie (is = 100)" pour le "user1".
- Le graphe généré (par la méthode *Neo4JDAO.LoadInitialData(true)*) est le graphe suivant (en bleu les films, en orange les users):



- Le résultat de l'exécution de la méthode *GetSuggestedMovies()* est la suivante:



```

Select Microsoft Visual Studio Debug Console
FantasticMovie : 2,5714285714285716
Start listening for @heigmacbot

```

- Afin de démontrer le fonctionnement de l'algorithme, nous allons analyser le score du film "FantasticMovie (id = 100)" utilisant la formule mathématique présentée:

$$S = \frac{\sum_{k=0}^{depth} 2^{depth-k} \cdot \overline{x_k}}{\sum_{k=0}^{depth} 2^{depth-k} \cdot \{1 \text{ si au moins une note au nv } k, 0 \text{ sinon}\}} \mid x_k = \text{moyenne des notes d'un film pour les amis de profondeur } k \text{ de l'utilisateur}$$

- Le calcul du score du film "FantasticMovie (is = 100)" est explicité ci-dessous, la profondeur maximale utilisée (*depth* = 6)
 - Moyenne des notations des amis de degré 1 pour ce film: 2 (1 seul ami de degré 1)
 - Moy des amis de deg 2: 3 (1 seul ami de degré 2)
 - Moy des amis de deg 3: (7 + 1) / 2 = 4 (car il y a deux amis de degré 3)
 - Somme des moyennes pondérées: 2 * (2^6) + 3 * (2^5) + 4 * (2^4) = 128 + 96 + 64 = 288
 - Calcul final: 288 / (2^6 + 2^5 + 2^4) = 288 / 112 = **2.57**
- Le résultat de l'appel de *GetSuggestedMovies()* abouti également à la note de 2.57, notre algorithme respecte donc la fonction mathématique présentée et permet une pondération dégressive des notations en fonction du niveau d'amitié

Tests unitaires

- Les différentes fonctionnalités ont été testés dans notre projet par couche. La lecture du code des tests unitaires permet une visualisation des fonctionnalités implémentées

Test Explorer		
Test	Duration	Tra
TestMoviesGlobalResources (12)	3 sec	
TestMoviesGlobalResources (12)	3 sec	
TestMongoDBDAO (3)	657 ms	
TestFindMovies	602 ms	
TestGetMovie	33 ms	
TestInsertMovie	22 ms	
TestMoviesGlobalResourcesContr...	2 sec	
TestGetMovieGenres	246 ms	
TestGetMoviesByFilterCast	114 ms	
TestGetMoviesByFilterCrew	266 ms	
TestGetMoviesByFilterGenre	338 ms	
TestGetMoviesByFilterLanguage	353 ms	
TestGetMoviesByFilterYear	365 ms	
TestGetMoviesByName	328 ms	
TestTMDBDAO (2)	614 ms	
TestGetMoviesByFilter	415 ms	
TestGetMoviesByName	199 ms	
TestSocialResources (1)	4 ms	
TestSocialResources (1)	4 ms	
TestNeo4JDAO (1)	4 ms	

Annexe : Données techniques

API key

dca39aa4da3c154aa1c1b0d293e9ba5b

Example API Request

https://api.themoviedb.org/3/movie/550?api_key=dca39aa4da3c154aa1c1b0d293e9ba5b

API Read Access Token (v4 auth)

eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkY2EzOWFhNGRhM2MxNTRhYTJFjMWIwZDI5M2U5YmE1YiIsInN1YiI6IjVIMDczOGUwMjZkYWMxMDAxNDczMTBmMyIsInNjb3BlcyI6WyJhcGlfcmlhZCI6LCJ2ZXJzaW9uIjoxfQ.VyGq-6ZAJUxOifbooMGSmcoeyP7R4tG4hy02iRKbhSM

MongoDB

DB name: Movie

Collection name: MovieCollection