

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.
  + 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
  + 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
  + 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
  + 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
  + 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 = nombre de relations d’amitié minimum entre deux personnes (plus court chemin) + 1*)

|

* Architecture propre, tests unitaires

# Fonctionnalités implémentées

* Recherche d’un film
  + Par nom
  + Par année
  + 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
  + 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)

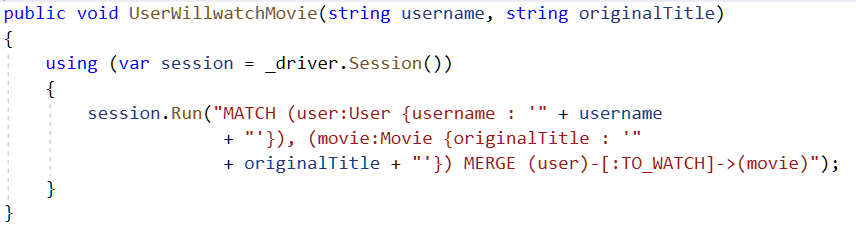


## Document (MongoDB)



# Requêtes avancées

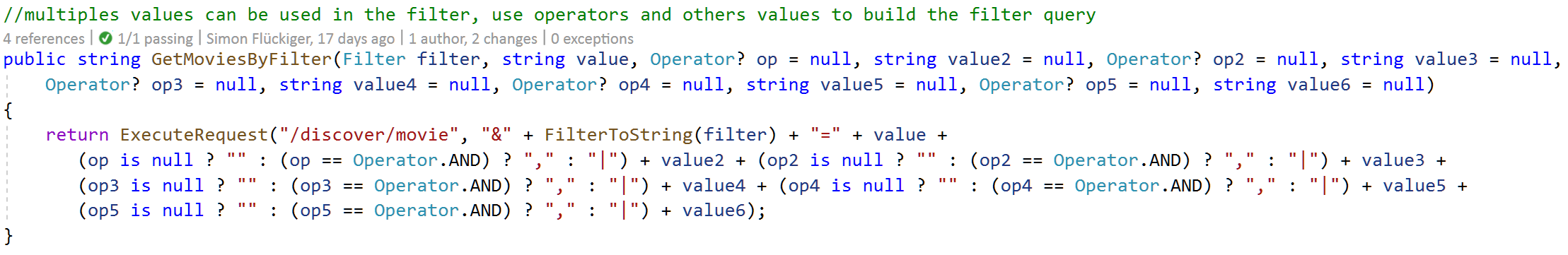
* Notre système comportant trois sources de données (i.e. Neo4JS, MongoDB, tmdbAPI), chacun de ces systèmes contient des requêtes qui lui sont propres.
* Neo4JS :
  + Définition d’une relation désignant le souhait d’un utilisateur de regarder un film :

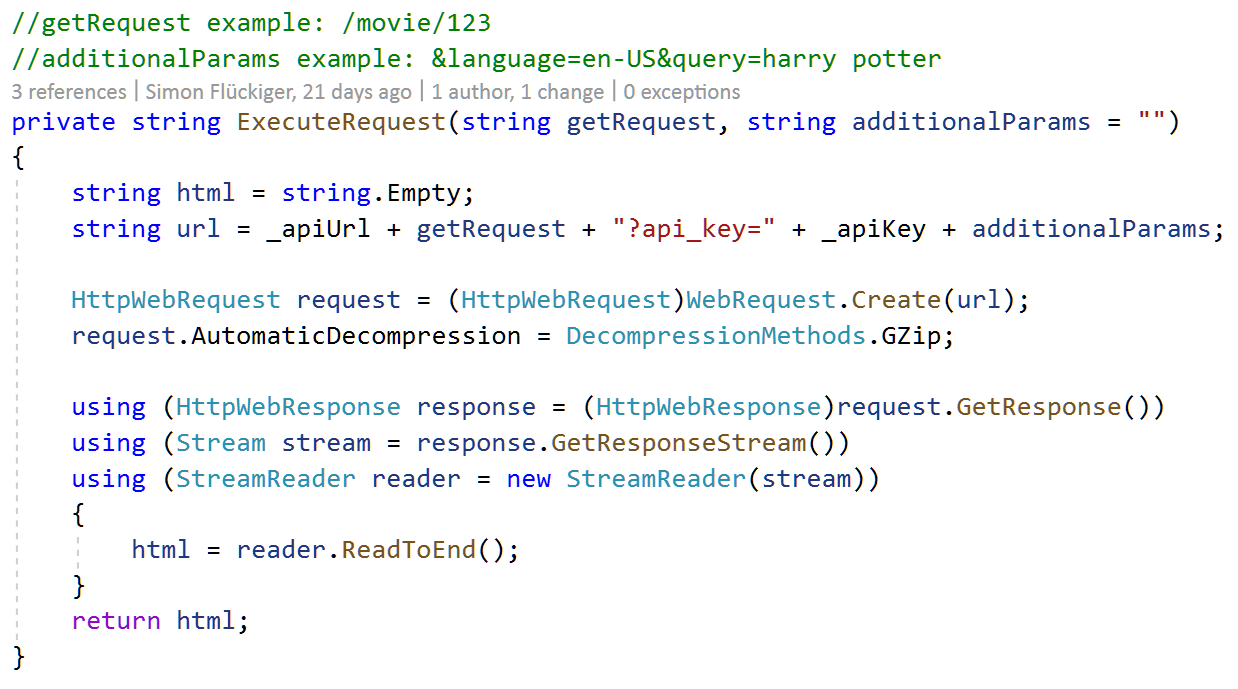


* + Récupération d’une liste de suggestions de films résultant de la moyenne pondérée de la notation des films par les *amis* d’un utilisateur tel que cité dans le descriptif du projet

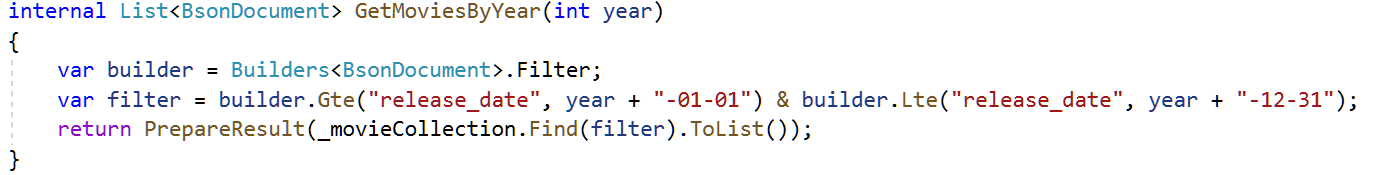


* TmdbAPI
  + Création d’une méthode C# d’abstraction d’un *query builder*, permet d’utiliser les opérateurs *AND* et *OR* lors du passage des paramètres d’une requête. Cette méthode permet une recherche utilisant le *Modèle Booléen* à plusieurs paramètres à l’aide des deux opérateurs précités, celle-ci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

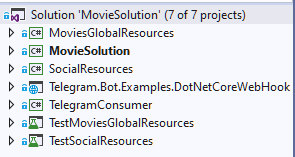




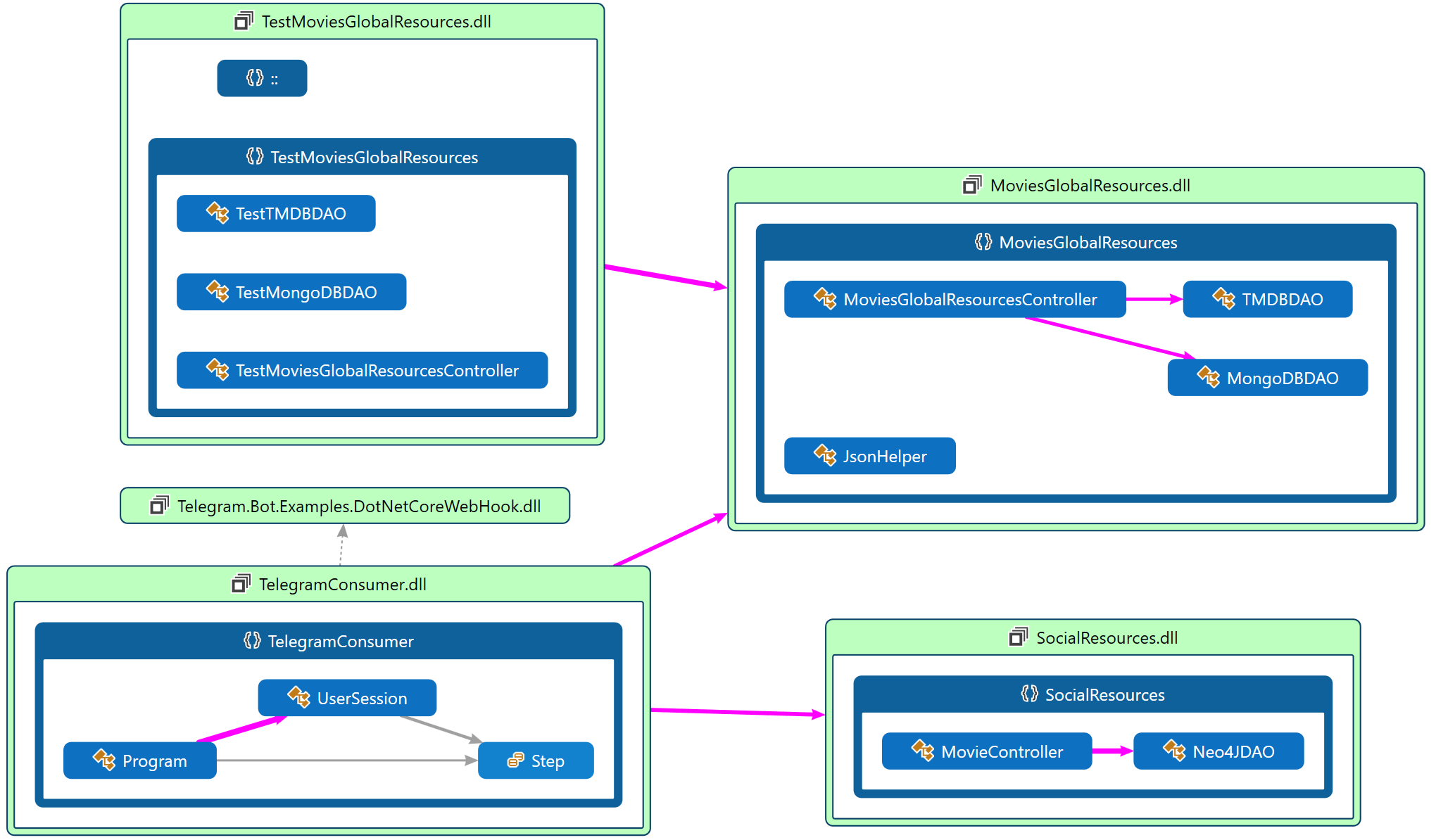
* MongoDB
  + Utilisation des librairies *MongoDB.Driver* et *MongoDB.Bson* , les objets *Bson* représentent un document MongoDB formaté en json dans une entité C# . L’exemple ci-dessous est la sélection des films par année



# Architecture du code

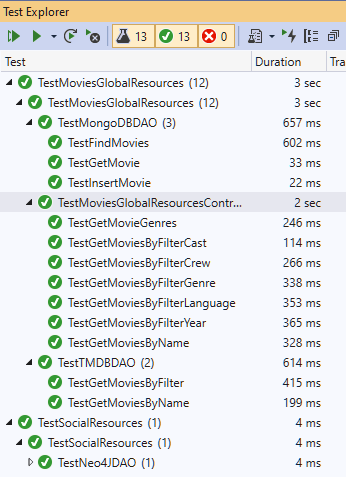


* MoviesGlobalResources : projet contenant la logique et le traitement des données de films globales (API TMDB + cache MongoDB)
* SocialResources : projet contenant la logique et le traitement des données du réseau social (Neo4J graphe DB)
* Telegram.Bot.Examples.DotNetCoreWebHook : web hook library 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*



# 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



# 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.eyJhdWQiOiJkY2EzOWFhNGRhM2MxNTRhYTFjMWIwZDI5M2U5YmE1YiIsInN1YiI6IjVlMDczOGUwMjZkYWMxMDAxNDczMTBmMyIsInNjb3BlcyI6WyJhcGlfcmVhZCJdLCJ2ZXJzaW9uIjoxfQ.VyGq-6ZAjUxOifbooMGSmcoeyP7R4tG4hy02iRKbhSM

## MongoDB

DB name: Movie

Collection name: MovieCollection