HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

# Méthodes d'accès aux données

## Projet - Rapport



| | |
|---|---|
| **Étudiants** | Gilliand Loris - Tutic Mateo - Wachter Luc |
| **Nom du professeur** | Fatemi Nastaran |
| **Nom des assistants** | Meier Christopher - Hochet Guillaume |
| **Date** | 20.01.2020 |

# Table des matières

# 1. Informations utiles

Notre application est un bot Telegram. Pour le tester, démarrez une discussion avec @TeleShows_Bot en lançant la commande `/start`. Pour lister les différentes actions possibles avec ce bot, tapez la commande `/help`.

## 1.1. Description des commandes

| **Commande** | **Description** |
|---|---|
| `start` | Démarre le *bot* |
| `help` | Retourne les informations sur l'utilisation du *bot* |
| `follow <nom>` | Recherche une série et permet de la suivre |
| `followed` | Affiche les séries suivies **et permet de marquer des épisodes comme vus** |
| `progress` | Voir la progression en terme d'épisode visionnés |
| `friends` | Trouver les utilisateurs qui ont le plus de séries en commun avec l'appelant |

# 2. Cahier des charges

## 2.1. Description

Pour le projet du cours *Méthodes d'accès aux données*, nous avons décidé de réaliser un outil qui permettra aux utilisateurs de garder un historique sur leurs séries visionnées.

Plus précisément, chaque utilisateur pourra sélectionner les séries qui l'intéressent et garder une trace des épisodes regardés dans une série donnée. Il pourra également aimer les épisodes regardés. L'outil permettra de récupérer une liste des séries les plus populaires du moment, selon plusieurs filtres.

Ces interactions seront faites via un bot Telegram. Plusieurs commandes permettront d'interagir avec la base de données des séries.

## 2.2. Entités

L'outil fera usage de quatre types d'entités :

- Série
- Saison
- Episode
- Utilisateur

Les séries sont représentée par leur titre, les années durant lesquelles elles étaient diffusées et l'url d'un poster la concernant.

Les saisons sont représentées par une description et le numéro de la saison.

Les épisodes sont représentés par une description et le numéro de l'épisode.

Les utilisateurs sont représentés par un nom d'utilisateur.

### 2.2.1. Ressources consommées

Les données sur les séries et les épisodes seront consommées depuis une API comme IMDB ou OMDB. C'est avec ces données que les utilisateurs interagiront.

### 2.2.2. Ressources générées

Les ressources générées seront principalement les utilisateurs. Ceux-ci seront stockés dans notre base de données lorsqu'une nouvelle discussion sera établie avec le bot Telegram. En effet, l'API Telegram donne accès aux développeurs à un objet `User` contenant entre autres un `id` unique. Une fois une série suivie par un utilisateur, cette dernière est ajoutée dans notre base de donnée. A la création, nous profitons pour déjà ajouter les noeuds pour chaque saison ainsi que l'épisode 1 de chaque saison.

## 2.3. Relations

Les différentes entités seront liées à l'aide des relations suivantes :

- `FOLLOWS` : l'utilisateur suit une série
- `HAS_SEEN` : l'utilisateur a vu un épisode d'une série
- `INCLUDES` : la série inclut une saison.
- `CONTAINS` : la série contient un épisode.

## 2.4. Fonctionnalités

Différentes interactions avec la base de données seront possibles depuis le bot Telegram. Une séparation est faite entre les actions de récupération de données et les actions de création de données.

### 2.4.1. Récupération de données

L'utilisateur pourra récupérer :

- une liste de séries selon une partie de titre
- la liste des séries que l'utilisateur suit
- la liste des épisodes vus dans une série donnée
- la liste des utilisateurs suivant le plus de séries en commun avec l'appelant (5 utilisateurs au maximum)

Non-implémentées :

- une liste des séries les plus regardées
- une liste des séries les plus populaires du moment
- une liste des utilisateurs selon une partie de nom.

### 2.4.2. Création de données

L'utilisateur pourra :

- démarrer une nouvelle discussion avec le bot, ce qui stockera l'utilisateur dans la base de données
- suivre une série
- marquer un épisode comme vu

Non-implémentées :

- aimer un épisode.
- annuler une action

# 3. Modèle de données

## 3.1. Technologie

Nous avons décidé d'utiliser `ArangoDB`. Cette technologie permet de regrouper le modèle de données graphe et le modèle de données document (entre autres), et propose son propre langage de requêtes.

Elle semble adaptée à un tel projet et nous permettrait d'expérimenter avec une technologie un peu différente des technologies de base de données mono-modèle.

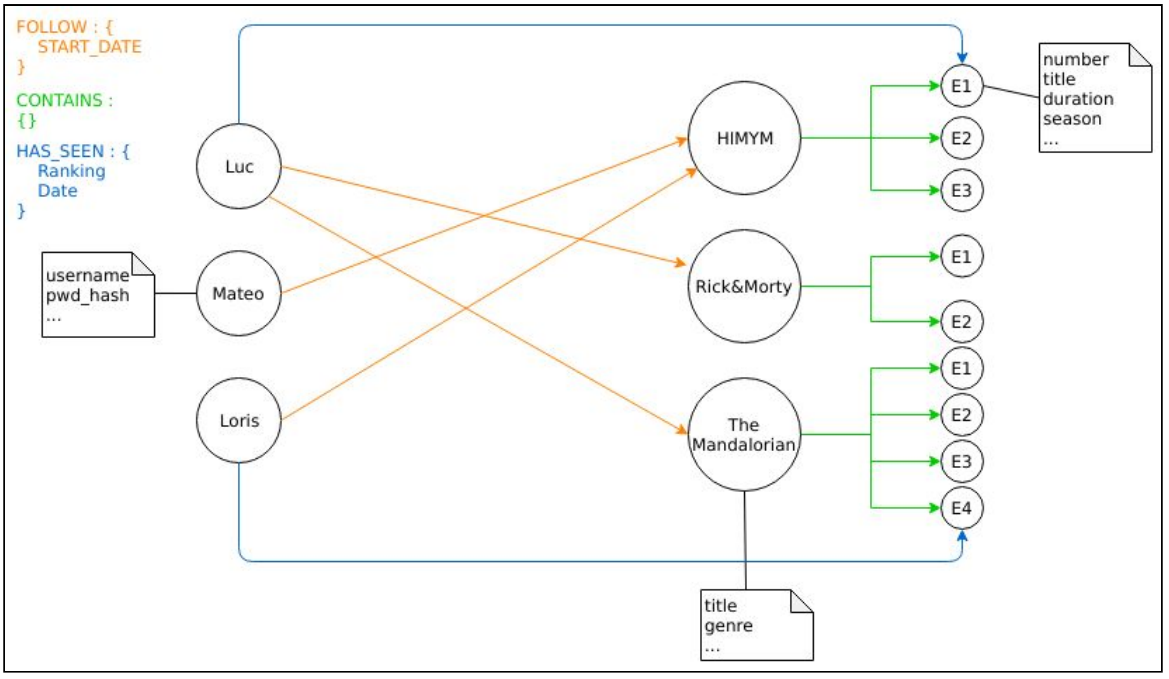
## 3.2. Modèle de données graphe

Vous trouverez dans la figure suivante une représentation basique d'un graphe possible avec notre modèle de données. On y trouve trois ensembles distincts de sommets, représentant respectivement les *utilisateurs*, les *séries* et leurs *épisodes*.

Les arcs sont également de trois types, et représentent les relations suivantes :

- l'utilisateur **suit** cette série
- l'utilisateur **a vu** cet épisode
- la série **inclut** cette saison
- la saison **contient** cet épisode

(Les deux dernières relations ne sont pas représentées correctement dans le modèle ci-dessous)



## 3.3. Modèle de données document

Les sommets de type `User` contiendront les informations nous permettant d'identifier un utilisateur du bot. Ces informations nous seront disponibles à travers l'API de Telegram (https://python-telegram-bot.readthedocs.io/en/stable/telegram.user.html).

Les sommets de type `Series` contiendront les informations relatives aux séries récupérées depuis une API ouverte. Les champs qui nous intéressent sont principalement l'id, le titre, le genre et la date de sortie.

Les sommets de type `Episode` contiendront bien sûr les informations relatives aux épisodes des séries. Les champs qui nous intéressent sont surtout l'id, le numéro d'épisode, le titre, la longueur, la date de sortie et la saison.

## 3.4. Requêtes

### 3.4.1. Complexité moyenne

Concernant les requêtes, plusieurs ont été décrites dans le cahier des charges. Ces dernières sont de complexité moyenne et seront développées selon le temps à disposition. Au minimum les requêtes suivantes seront disponibles :

| Description de la requête | Noeuds et relations impliqués |
|---|---|
| **Récupérer** une liste de séries en fonction du titre. | Série (données consommées). |
| **Récupérer** la liste des séries suivies par l'utilisateur. | Série, utilisateur, relation FOLLOWS. |
| **Récupérer** la liste des épisodes vus dans une série donnée. | Série, épisode, utilisateur, relation HAS_SEEN. |
| **S'enregistrer** en tant qu'utilisateur (lors du lancement de la discussion avec le bot). | Utilisateur. |
| **Suivre** une série. | Utilisateur, série, relation FOLLOWS. |
| **Marquer** un épisode comme vu (et le noter). | Utilisateur, série, épisode, relation HAS_SEEN. |

### 3.4.2. Complexité avancée

Afin d'implémenter une requête plus complexe, qui utiliserait mieux les capacités de notre base de données basée graphes, nous avons pensé à la requête suivante.

*Quels sont les 5 utilisateurs ayant le plus de séries suivies en commun avec l'appelant ?*

La requête précédente traduite en *AQL* (langage d'Arango) à l'aide d'un parcours de graphe est la suivante :

```
FOR v, e, p IN 2..2 ANY
  'Users/399684924'
  GRAPH 'SeriesGraph'
  FILTER CONTAINS(v._id, 'Users/')
  COLLECT user = v WITH COUNT INTO seriesInCommon
  FILTER seriesInCommon > 0
  SORT seriesInCommon DESC
  LIMIT 5
  RETURN { user, seriesInCommon }
```

# 4. Structure du code source

Le code source de notre application se trouve dans le répertoire `src/`. On y trouve les fichiers et répertoires suivants :

| Fichier / Répertoire | Description |
|---|---|
| `bot_app.py` | Fichier principal qui lance le *bot* et écoute les différentes commandes saisies par l'utilisateur |
| `commands` | Répertoire dans lequel sont écrites toutes les *callbacks* des différentes commandes possibles avec le bot |
| `commands/bot.py` | Contient toutes les *callbacks* concernant les commandes de base avec le *bot* (`/start`, `/help`) |
| `commands/tvshows.py` | Contient toutes les *callbacks* concernant les commandes en rapport avec les séries (`/follow`, `/followed`, ...) |
| `commands/utils.py` | Contient quelques fonctions utilitaires pour l'écriture des *callbacks* |
| `data` | Répertoire dans lequel sont écrites toutes les fonctions qui permettent d'interagir avec la base de données |
| `data/database.py` | Initialise la connection à la base de donnée et permet d'exécuter différentes opérations |
| `data/graph.py` | Définit le modèle de données graphe |
| `data/series_api.py` | Fournit les fonctions pour récupérer les données depuis l'API `movie-database-imdb-alternative` de |

| | *RapidAPI* |
|---|---|
| `decorators` | Répertoire dans lequel sont écrits les différents décorateurs python |
| `decorators/singleton.py` | Décorateur utilisé pour le pattern singleton. Il est utilisé pour la classe `data/database.py` car on souhaite initier une seule instance de la base de données |