

# Projet Allociné



Franck Le Fur

# Table des matières

<b>1</b>	<b>Présentation</b>	<b>2</b>
1.1	Présentation du projet . . . . .	2
1.2	Spécifications techniques . . . . .	2
1.3	Organisation du travail . . . . .	3
<b>2</b>	<b>Préliminaires</b>	<b>4</b>
2.1	Préparation de l'environnement . . . . .	4
2.2	Création d'un repository GitHub . . . . .	4
<b>3</b>	<b>Webscrapping</b>	<b>5</b>
3.1	Introduction . . . . .	5
3.2	Présentation des outils de web scraping . . . . .	5
3.3	Scraper les informations des films de 1960 à 2024 . . . . .	6
3.4	Scraper les informations des films de 2025 . . . . .	7
3.5	Difficultés rencontrées . . . . .	8
<b>4</b>	<b>Requêtage d'une API publique</b>	<b>9</b>
4.1	Présentation de l'API OMDB . . . . .	9
4.2	Récupération des informations . . . . .	10
4.3	Système BigData . . . . .	10
<b>5</b>	<b>Nettoyage et agrégation des données</b>	<b>11</b>
5.1	Agrégation des données . . . . .	11
5.2	Nettoyage et formatage des données . . . . .	11
<b>6</b>	<b>Création de la base de données</b>	<b>12</b>
6.1	Introduction au SGBD . . . . .	12
6.2	Modèles MCD et MPD . . . . .	12
6.3	Création de la base et des tables MySQL . . . . .	14
6.4	Remplissage de la base MySQL . . . . .	15
6.5	Création et remplissage de la base MongoDB . . . . .	16
6.6	Exemples de requêtes SQL . . . . .	16
<b>7</b>	<b>Création d'une API</b>	<b>17</b>
7.1	Généralités sur les APIs . . . . .	17
7.2	Création de l'API . . . . .	17
7.3	Règle d'authentification . . . . .	19
7.4	Exemples d'utilisation de l'API . . . . .	20
<b>8</b>	<b>Automatisation</b>	<b>21</b>
8.1	Présentation de Crontab . . . . .	21
8.2	Création d'un environnement virtuel sous WSL . . . . .	21
8.3	Script à exécuter . . . . .	21
8.4	Création d'une tâche Crontab . . . . .	21

# 1 Présentation

## 1.1 Présentation du projet

Ce projet présente une application permettant de faire des requêtes sur une base de données concernant le cinéma. L'application offre les options classiques de filtres de films selon des mots clés dans le titre, les noms d'acteurs, le réalisateurs, le compositeur, l'année de production, les catégories de films.

Elle offre également la possibilité de combiner ensemble tous ces filtres et ainsi de répondre à des questions telles que :

- Quels films ont réuni les acteurs Pierre Richard et Gérard Depardieu ?
- Dans combien de films de la franchise "Terminator" ont joué ensemble Linda Hamilton et Schwarzenegger ?
- Combien de films avec Pierre Richard ont vu leur musique composée par Vladimir Cosma ?

Cette application peut être proposée à des professionnels du cinéma, par exemple des critiques de films, ayant besoin de filtres avancés pour faire des recherches sur des associations dans le cinéma.

L'**objectif fonctionnel** est de construire une base de données et de l'exposer via une API

## 1.2 Spécifications techniques

Ce projet sera mené en **Python**, ce langage offre toutes les librairies, de façon gratuites, nécessaires à la conduite de ce projet.

Les données seront collectées à partir du site [allocine.fr](http://allocine.fr) ne utilisant les librairies python **Beautifulsoup** et **Selenium**.

Des données supplémentaires seront récupérées par requêtage de l'api publique **OMDB** en utilisant la librairie python **requests**.

Manipulation, nettoyage et agrégation des données seront fait à l'aide de librairies python **Panda** et **Numpy**.

Nous utiliserons deux SGBD : **MySQL** et **MongoDB**, la mise en base et manipulation des données seront faites à l'aide des librairies python **mysql.connector** et **pymongo**.

Les tâches de scrapping, de requêtage de l'api public, de nettoyage et aggrégation des données seront automatisées à l'aide de l'outil **Crontab** disponible sur la machine Linux **WSL**.

L'API sera créée avec **FastAPI**, la documentation sera faite selon le modèle **OpenAPI**.

Un repo **github** sera créé pour le versioning des fichiers.

Enfin, le présent rapport sera rédigé en **Latex** via l'utilitaire en ligne **overleaf**.

### 1.3 Organisation du travail

- 1 Scrapping du site allociné,
- 2 Requête de l'api publique OMDb,
- 3 Ecriture des données en CSV,
- 4 Nettoyage et agrégation des données,
- 5 Insertion des données dans MySQL,
- 6 Insertion des données dans MongoDB,
- 7 Création de l'API avec Fast,
- 8 Requetes sur les bases pour notre API,
- 9 Automatisation de l'extraction des données.

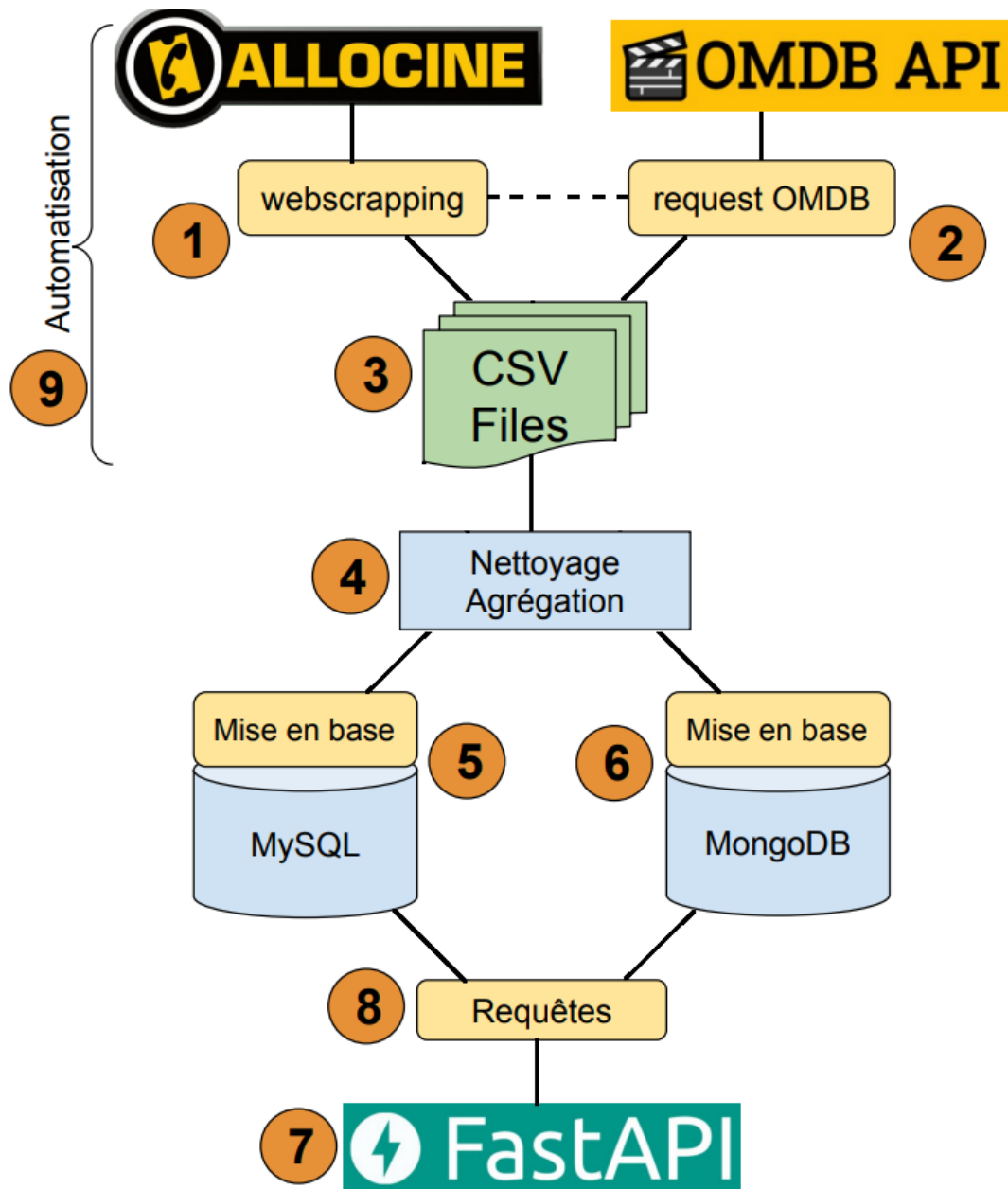


FIGURE 1 – Organigramme de travail

## 2 Préliminaires

### 2.1 Préparation de l'environnement

Dans un premier temps nous créons un nouvel environnement virtuel sous conda avec toutes les librairies python nécessaires.

#### Création d'un nouvel environnement

```
1 C:\Users\Utilisateur>conda create --name block1
2 C:\Users\Utilisateur>conda activate block1
```

#### Installation des packages python

```
1 C:\Users\Utilisateur>conda install numpy pandas tqdm
2 C:\Users\Utilisateur>conda install requests beautifulsoup4 selenium
3 C:\Users\Utilisateur>conda install mysql-connector-python
4 C:\Users\Utilisateur>conda install pymongo
5 C:\Users\Utilisateur>conda install unicodecode
6 C:\Users\Utilisateur>conda install fastapi pyjwt uvicorn
```

### 2.2 Création d'un repository GitHub

Création d'un repository **Github** via l'interface github puis connexion de notre répertoire de travail au repository **Github**.

```
git init
git add .gitignore
git branch -m master main (pour renommer la branche)
git commit -m "first commit"
git remote add origin https://github.com/Franck-LF/projectBlock1
git push -u origin main
```

Ensuite à chaque ajout d'une fonctionnalité dans le code, où modification majeure, nous pousserons les mises à jour sur le repository Github.

## 3 Webscrapping

### 3.1 Introduction

Le **webscrapping** est un processus d'extraction automatique de données à partir d'un site web. Il s'agit de parcourir des pages web et d'y récupérer le contenu souhaité. Ce processus doit être fait en respectant les conditions d'utilisation des sites et les lois sur la protection des données.

Dans ce projet les données seront scrappées à partir du site **allocine.fr**, ces données sont entièrement publiques, en effet il s'agit exclusivement de titres de films, noms d'acteurs, réalisateurs et compositeurs.

Il nous faut vérifier que le site **allocine.fr** autorise les pratiques de scrapping sur la section **films** qui nous intéresse, pour cela nous consultons le fichier "robots.txt".

Le fichier robots.txt n'indique aucune restriction sur le chemin `/films/`, point d'entrée exclusif de notre web scraping, nous pouvons donc collecter en toute légalité les données souhaitées.



```
User-Agent:*
Disallow: /rechercher/
Disallow: /salle/recherche/
Disallow: /blogvision/
Disallow: /communaute/membre/reportabuseforum_gen_nofil=*
Disallow: /*?iframe*
Disallow: /ws/AcVisiondata.ashx?*
Disallow: /ws/AcVisiondataV4.ashx?*
Disallow: /ws/ProfillightGetModerationLink.ashx?*
Disallow: /ws/geocoding.ashx
Disallow: /ws_v7/geocoding.ashx
Disallow: /ws/WinnerListHandler.ashx
Disallow: /ws/PostRollTrailer.ashx?*
Disallow: /ws/partnerproxy.ashx?*
Disallow: /ws_v7/Disqus.asmx/
Disallow: /ws_v7/get_complete_photos_list.ashx?*
Disallow: /120157152/
Disallow: /doubleclick/
Disallow: /_/
Disallow: /_video/partner/

User-Agent: seekbot
Disallow: /

User-Agent: Exabot
Disallow: /

User-agent: Slurp
Disallow: /

User-agent: Cliqzbot
Disallow: /
```

FIGURE 2 – fichier robots.txt

### 3.2 Présentation des outils de web scraping

Nous utilisons la librairie python **requests** pour récupérer le contenu de pages html, ensuite le web scraping se fera à l'aide de la librairie **Beautifulsoup**, cette librairie permet d'analyser et de parser le contenu html d'une page web.

(Un usage réduit et très spécifique de la librairie **Selenium** sera détaillée [ici](#).)



## Exemple classique d'utilisation de BeautifulSoup

On commence par utiliser l'inspecteur du navigateur pour détecter les balises html qui contiennent les informations souhaitées, ensuite on utilise la méthode `find` (ou `find_all`) sur un objet `soup` de BeautifulSoup pour récupérer le contenu de ces balises, par exemple la commande :

```
elt_categories = soup.find_all('div', class_='filter-entity-section')
```

renvoie toutes les balises html `div` ayant un attribut `class` égal à "filter-entity-section".

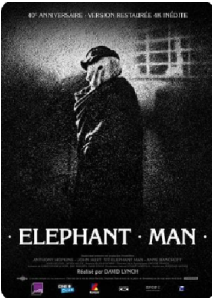
Ensuite on peut récupérer le texte d'une balise : `elt.a.text`

ou bien la valeur d'un attribut : `elt.get_attribute('title')`

## 3.3 Scraper les informations des films de 1960 à 2024

Dans un premier temps nous collectons les informations des films de 1960 à 2024, nous nous limitons aux films ayant plus de 30 avis utilisateurs et nous fixons un maximum de 250 films par année. Finalement **8800** films seront collectés entre 1960 et 2024.

Nous collectons les liens vers les années de production des films, ensuite pour une année donnée, nous parcourons la liste des films et collectons les informations sur la page du film ainsi que sur la section **infos techniques** et la section **Casting complet et équipe technique**



**8 avril 1981 en salle** | 2h 05min | Biopic, Drame  
Date de reprise 22 juin 2020  
De **David Lynch** | Par **Christopher De Vore, David Lynch**  
Avec **Anthony Hopkins, John Hurt, Anne Bancroft**  
Titre original **The Elephant Man**

**LES INDÉS**

Presse **4,8**  
5 critiques





Spectateurs **4,4**  
20470 notes, 690 critiques

Mes amis  
==  
★★★★★

### Infos techniques

Nationalité	U.S.A.
Distributeur	Carlotta Films
Récompenses	4 prix et 16 nominations
Année de production	1980
Date de sortie DVD	11/12/2001
Date de sortie Blu-ray	03/11/2009
Date de sortie VOD	08/02/2007
Type de film	Long métrage
Secrets de tournage	23 anecdotes
Budget	5 000 000 USD

### Acteurs et actrices

 <p><b>Anthony Hopkins</b> Rôle : Dr. Frederick Treves (chirurgien)</p>	 <p><b>John Hurt</b> Rôle : John Merrick, Elephant Man</p>	 <p><b>Anne Bancroft</b> Rôle : Mrs. Madge Kendal (actrice)</p>	 <p><b>John Gielgud</b> Rôle : Carr Gomm (directeur de l'hôpital)</p>	<b>Lesley Dunlop</b> infirmière Nora
				<b>Helen Ryan</b> La princesse Alex
				<b>Kenny Baker</b> Nain à plumes
				<b>John Standing</b> Fox
				<b>Dexter Fletcher</b> Le garçon de Bytes
				<b>Phoebe Nicholls</b> La mère de Merrick
				<b>Pat Gorman</b> Bobby au champ de foire
				<b>Claire Davenport</b> Femme obèse
				<b>Oria Pederson</b> Homme squelettique
				<b>Patsy Smart</b> Femme désespérée
				<b>Frederick Treves</b> Alderman
				<b>Richard Hunter</b> Hodges
				<b>James Cormack</b> Pierce
				<b>Alfie Curtis</b> Le livreur de lait
				<b>Robert Lewis Bush</b> Le messager
				<b>Roy Evans</b> Le chauffeur de taxi
				<b>Joan Rhodes</b> Le cuisinier
				<b>Nula Conwell</b> L'infirmière Kathleen
				<b>Tony London</b> Porter jeune





 <p><b>Wendy Hiller</b> Rôle : L'infirmière en chef</p>	 <p><b>Freddie Jones</b> Rôle : Bytes (propriétaire d'Elephant Man)</p>	 <p><b>Hannah Gordon</b> Rôle : Mrs. Treves</p>	 <p><b>Michael Elphick</b> Rôle : Le gardien de nuit</p>
--	--	--	---

FIGURE 3 – Vignette du film, infos techniques et casting

Nous collectons les informations suivantes :

- Le titre du film
- Le titre original du film
- La date de sortie
- La durée
- La liste des catégories associées
- Les pays de production
- La liste des acteurs
- La liste des réalisateurs
- La liste des compositeurs
- La note associée au film
- Le nombre de notes spectateurs
- Le nombre d'avis spectateurs
- Le synopsis
- L'url de l'affiche

### 3.4 Scraper les informations des films de 2025

Ici nous collectons les informations des films sortis en salle le mercredi, nous ne travaillons donc pas par année mais par semaine à partir de la section dédiée.

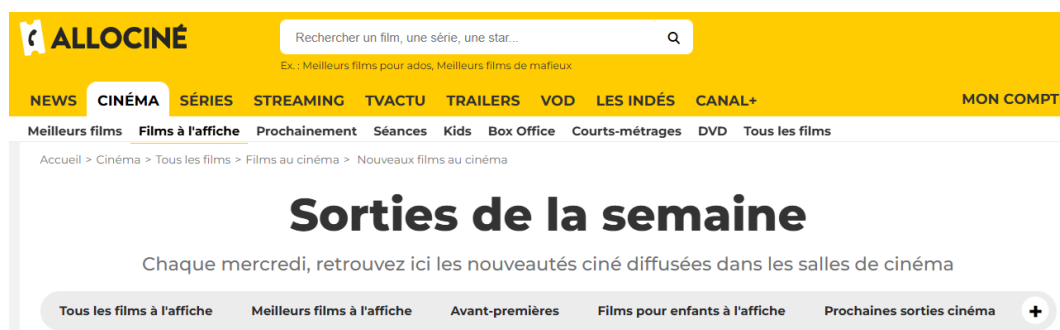


FIGURE 4 – Section "Sorties de la semaine"

La tâche de scrapping est très semblable au scrapping par année, notons tout de même quelques différences :

- Les films récents n'ont pas de section "ratings" ou bien ont des sections "ratings" vides, des tests supplémentaires ont été ajoutés dans la phase de web scraping pour gérer ce cas.
- Les films récents ont parfois une section "rating" mais avec zéro avis utilisateur alors que jusqu'à présent nous ne prenions que les films avec au moins 30 avis, des options supplémentaires ont été créées et sont transmises pour ne pas tenir compte du nombre d'avis.

Cette tâche de web scraping sera automatisée de façon hebdomadaire pour récupérer les informations de tous les nouveaux films à l'affiche (voir section Crontab).

Toutes les informations collectées lors du web scraping seront stockées dans des **fichiers csv**.



### 3.5 Difficultés rencontrées

- 1 Nous avons collecté la liste des pays de production des films à partir de la section dédiée mais il apparaît que cette liste est incomplète, par exemple le **Bostwana** n'est pas dans la liste mais il existe bien des films sur allociné dont le pays de production est le **Bostwana**. Finalement, nous construisons la liste des pays directement à partir des informations des films et non pas à partir de la section "pays" du site. Au moment d'insérer un nouveau film dans la base de données, nous interrogeons la base pour vérifier que le pays correspondant est bien dans la table "countries", dans le cas contraire nous l'ajoutons.
- 2 Lors du web scraping des films à l'affiche début 2025, certains films avaient une année de production "2024" et avaient déjà été scrapés, nous avons donc créé des doublons en base. Un nettoyage des doublons a été fait à partir de la base MySQL.
- 3 Lors de l'utilisation de BeautifulSoup, certains liens apparaissent **décorés** et sont donc inexploitable, l'utilisation de la librairie **Selenium** permet de résoudre ce problème.

```
<li class="filter-entity-item"> == $0
  <a class="xXx item-content" title="2030 - 2039" href="/films/decennie-2030/">2030 - 2039</a>
  <span class="light">(47)</span>
</li>
▶ <li class="filter-entity-item">...</li>
▶ <li class="filter-entity-item">...</li>
```

FIGURE 5 – Inspecteur Chrome

```
<span class="ACrL2ZACrpbG1zL2RlY2Vubml1LTlwMzAv item-content" title="2030 - 2039">
  2030 - 2039
  décoration
</span>
```

FIGURE 6 – Résultat BeautifulSoup

## 4 Requête d'une API publique

### 4.1 Présentation de l'API OMDB

Nous avons choisi l'api publique [OMDB](#) pour collecter des informations de **synopsis** de films et d'**url d'affiche** de films, à noter que ces informations de texte et d'url sont non-structurés et seront destinés à aller en base NoSQL.

Pour accéder à l'API d'OMDB nous demandons une clé dans la section dédiée, cela nous offre un accès gratuit à 1000 requêtes par jour.

FIGURE 7 – Générer une clé API

La documentation de l'API OMDB nous indique la forme des requêtes, par exemple pour le film **In The Lost Lands** :

**`https://www.omdbapi.com/?apikey=xxxx&t=in+the+lost+lands`**

A l'aide de l'utilitaire **requests** nous obtenons le résultat sous forme d'un json.

```
1 { 'Title': 'In the Lost Lands',
2   'Year': '2025',
3   'Released': '07 Mar 2025',
4   'Runtime': '101 min',
5   'Genre': 'Action, Adventure, Fantasy',
6   'Director': 'Paul W.S. Anderson',
7   'Writer': 'Constantin Werner, Paul W.S. Anderson, George R.R. Martin',
8   'Actors': 'Milla Jovovich, Dave Bautista, Arly Jover',
9   'Plot': 'A queen sends the powerful and feared sorceress Gray Alys to the
10          ghostly wilderness of the Lost Lands in search of a magical power, where
11          the sorceress and her guide, the drifter Boyce, must outwit and outfight
12          man and demon.',
13   'Language': 'English',
14   'Country': 'Germany, Canada, United States',
15   'Poster': 'https://m.media-amazon.com/images/M/
              MV50WYxYjEyYTtY2FkZC00Y2wLTk1ZjMtMAyMTAyMzQ3MDiXkEyXkFcgGc@._V1_SX300.jpg' }
```

## 4.2 Récupération des informations

Nous automatisons la création de requêtes OMDb au bon format à partir des titres de films déjà collectées par web scrapping, ensuite nous requêtons l'api OMDb et agrégeons les nouvelles données collectées aux données existantes.

A l'issue de cette étape les données sont enregistrées dans des fichiers CSV.

**Remarque :** Nous avons commencé à stocker les affiches de films mais ce n'est pas pertinent d'un point de vue gestion de la mémoire, nous préférons garder en base les urls des affiches, les affiches étant stockées sur internet.

## 4.3 Système BigData

Plusieurs pistes de systèmes BigData ont été explorées, elles se sont montrées incomplètes, incohérentes et / ou difficiles à mettre en place dans notre projet.

Voici par exemple une requête SPARQL avec le système **WikiData**.

```
1 SELECT ?film ?filmLabel WHERE {  
2   ?film wdt:P31 wd:Q11424.           # ?film est une instance de "film"  
3   ?film wdt:P577 ?date.               # ?film a une date de sortie  
4   FILTER(YEAR(?date) = 2020)         # Filtre les films sortis en 2020  
5   SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }  
6 }  
7 LIMIT 100
```

film	filmLabel
<a href="#">Q wd:Q171861</a>	The Artist
<a href="#">Q wd:Q681269</a>	Jazz on a Summer's Day
<a href="#">Q wd:Q681269</a>	Jazz on a Summer's Day
<a href="#">Q wd:Q903541</a>	Delicate Sound of Thunder
<a href="#">Q wd:Q1217981</a>	La Baie des Anges
<a href="#">Q wd:Q1415194</a>	Kin-dza-dza!
<a href="#">Q wd:Q1702996</a>	Scooby-Doo et le Fantôme de la sorcière
<a href="#">Q wd:Q1976114</a>	La Fille à l'écho
<a href="#">Q wd:Q2095178</a>	L'Homme qui tua Don Quichotte

FIGURE 8 – Résultat Wikidata

Parmi les difficultés rencontrées avec ce système, citons : le langage SPARQL très spécifique, des données difficiles à collecter en dehors de l'interface dédiée (il faudrait utiliser le wrapper **SPARQLWrapper** pour automatiser des requêtes).

## 5 Nettoyage et agrégation des données

### 5.1 Agrégation des données

Après la tâche de web scrapping, les données sont sous forme d'un dataframe **Pandas**. Lors de la phase de requêtage de l'API **OMDB** nous vérifions que la requête nous fournit des informations supplémentaires (**synopsis** du film et **url d'affiche**), dans quel cas nous l'ajoutons au Dataframes.

### 5.2 Nettoyage et formatage des données

Avant d'insérer les données en bases, nous devons nous assurer qu'elle sont au bon format.

#### Exemples de formatage

- 1 Les informations scrapées concernant les catégories de films, les acteurs, réalisateurs et compositeurs ont été mis sous la forme d'une chaîne de caractères où les entités sont séparées par une virgule.

```
"Alain Delon,Olga Georges-Picot,Charles Bronson,Brigitte Fossey,Bernard Fresson,
```

```
Jean-Paul Tribout,Ellen Bahl,Stéphane Bouy"
```

 Nous transformons cette chaîne de caractères en liste python de chaînes de caractères

```
["Alain Delon", "Olga Georges-Picot" ...]
```

 en nous assurant qu'il n'y a pas de doublon.

- 2 Certaines informations sont manquantes sur le site allociné (durée du film, nom du réalisateur, date de sortie et même la liste des acteurs pour les films d'animation etc ....), les champs dans nos Dataframe Pandas sont alors représentés par des **NaN** en python, nous les remplaçons par des chaînes de caractères vides ou bien par des valeurs nulles selon le type de champs souhaités.
- 3 Les durées sont stockées sous forme de chaînes de caractères de la forme **1h 35min**, nous les transformons en durée en minutes (95) de type **entier**.
- 4 Les notes sont stockées sous la forme d'une chaîne de caractères "3,5" où parties entière et décimale sont séparées par une virgule, nous remplaçons les virgules par des points.
- 5 Les dates de sortie de film sont stockées au format **5 mars 2025** sont converties au type Pandas **datetime**, pour se faire nous devons au préalable convertir les mois du français vers l'anglais.

**Remarque :** Un nettoyage a déjà été fait lors du scrapping, par exemple les valeurs 0 (pour des entiers ou flottants) et chaîne de caractère vide (pour des string) auront été attribuées aux données dans le cas où celles-ci sont manquantes sur le site allociné.

## 6 Création de la base de données

### 6.1 Introduction au SGBD

Un SGBD (Système de Gestion de Base de Données) est un logiciel prévu pour stocker, manipuler et extraire des données.

Pour ce projet, nous avons choisi les SGBD :

- **MySQL** pour toutes les **données structurées** c'est-à-dire les données numériques ou catégorielles, les entités concernées (films, acteurs, réalisateurs, compositeurs, catégories) ont des relations étroites et des requêtes de jointures ou de filtres complexes seront nécessaires, **MySQL** semble adaptée à ce besoin,
- **MongoDB** pour toutes les **données non-structurées** comme du texte ou des urls, dans notre projet il s'agira des synopsis des films et des urls vers les affiches de films, données qui sont uniquement liées avec l'entité film et ne nécessite donc pas de requête complexe.

### 6.2 Modèles MCD et MPD

**MCD** : Modèle conceptuel de données, il est indépendant du SGBD choisi, c'est "une carte mentale des données offrant une compréhension partagée entre développeurs et décideurs").

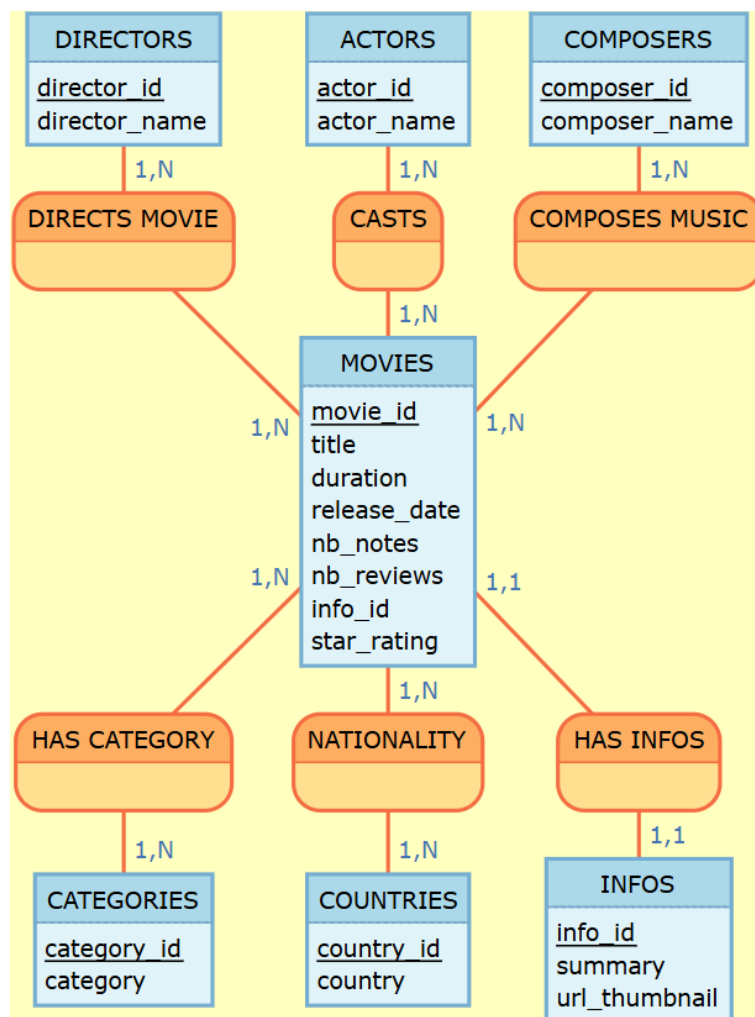


FIGURE 9 – MCD avec l'utilitaire Mocodo

Pour concevoir un **MCD** nous devons comprendre "le métier", c'est-à-dire :

- connaître les entités mises en jeu (ici les films, acteurs, catégories ...),
- identifier leurs attributs (titre, nom etc ...),
- savoir si ces attributs sont de type quantitatif ou catégoriel,
- identifier les relations entre les entités,
- définir les cardinalités dans ces relations (c'est-à-dire le nombres d'entités qui peuvent reliés à une autre entité).

### Remarque sur les cardinalités

- 1 La cardinalité dans la relation **movies - directors** est de type **many-to-many** avec cardinalité minimal 1, l'interprétation est qu'un film est réalisé par au minimum un réalisateur et peut avoir été réalisé par plusieurs réalisateurs et réciproquement un réalisateur a réalisé au moins un film (sinon il ne serait pas dans nos données puisque nous sommes partis des films pour récupérer les informations) et peut aussi avoir réalisé plusieurs films.
- 2 La cardinalité dans la relation **movies - infos** est de type **1 - 1** car un film possède exactement une fois les informations qui le concernent (synopsis et url vers l'affiche de film) et que réciproquement une entité **infos** correspond exactement à un film.

Ensuite nous pouvons passer à la conception de notre **MPD** (Modèle physique de données), il dépend directement du SGBD choisi, ici **MySQL**, il représente les tables, les colonnes, les clés primaires et étrangères, on y précise les types de données, les contraintes etc...



FIGURE 10 – MPD avec l'utilitaire DrawDB



Pour concevoir un **MPD** nous devons :

- convertir chaque entité du MCD en table du MPD,
- convertir les attributs des entités du MCD en colonne,
- définir le type de données pour chaque colonne,
- définir les clés primaires (identifiant unique de chaque enregistrement),
- convertir les relations en clés étrangères.

Pour exprimer une relation **many-to-many** nous avons besoin de créer une nouvelle table, appelée **table de jonction**, elle sert uniquement à faire le lien entre deux tables. Sa clé primaire est le couple composée des 2 clés étrangères des tables qu'elle relie. Dans le cadre de notre projet nous avons ajouté 5 tables de jonctions.

### 6.3 Création de la base et des tables MySQL

A partir du diagramme de tables conçu avec l'utilitaire **DrawDB** nous générons un fichier .sql contenant la structure de nos tables SQL, voici un extrait de ce fichier légèrement adapté.

```
1 CREATE TABLE actors (  
2     actor_id  VARCHAR(255)    NOT NULL,  
3     actor_name VARCHAR(255)    NOT NULL,  
4     PRIMARY KEY (actor_id),  
5     UNIQUE KEY (actor_name)  
6 );  
7  
8 CREATE TABLE actor_movie (  
9     actor_id  VARCHAR(255)    NOT NULL,  
10    movie_id   VARCHAR(255)    NOT NULL,  
11    PRIMARY KEY (actor_id, movie_id),  
12    FOREIGN KEY (actor_id) REFERENCES actors (actor_id) ON DELETE CASCADE,  
13    FOREIGN KEY (movie_id) REFERENCES movies (movie_id) ON DELETE CASCADE  
14 );
```

**Remarque :** La clause **ON DELETE CASCADE** sur une clé étrangère spécifie le comportement lors de la suppression d'un enregistrement dans la table parente. Concrètement si un acteur est supprimé de la base alors seront supprimés toutes les références à cet acteur dans la table de jonction **actor\_\_movie** afin de ne pas laisser d'enregistrements "orphelins".

Nous pouvons lancer l'exécutable **mysql.exe** en lui donnant le fichier .sql qui contient l'architecture complète de notre base MySQL.

```
1 C:\Users\Utilisateur\Documents\Block1>"C:\Program Files\MySQL\MySQL Server  
2 8.0\bin\mysql.exe" < movies.sql -u root -p  
3 Enter password: *****  
4 CREATING DATABASE STRUCTURE  
5 storage engine: InnoDB  
6 EVERYTHING IS OK
```

Nous pouvons visualiser nos tables à partir de la console **MySQL Shell**,

```

MySQL localhost:33060+ ssl SQL > SHOW DATABASES;
+-----+
| Database |
+-----+
| employees |
| games |
| gamesfromdumps |
| information_schema |
| list_books |
| movies |
| mysql |
| onetoone |
| performance_schema |
| sakila |
| sys |
| world |
+-----+

MySQL localhost:33060+ ssl movies SQL > SHOW TABLES;
+-----+
| Tables_in_movies |
+-----+
| actor_movie |
| actors |
| categories |
| category_movie |
| composer_movie |
| composers |
| countries |
| country_movie |
| director_movie |
| directors |
| infos |
| movies |
+-----+

```

## 6.4 Remplissage de la base MySQL

Nous commençons par remplir les tables **categories**, **actors**, **directors**, **composers**, **countries**, ces tables n'ont pas de clé étrangère, elles ne dépendent pas directement d'autres tables et doivent être créées en premier.

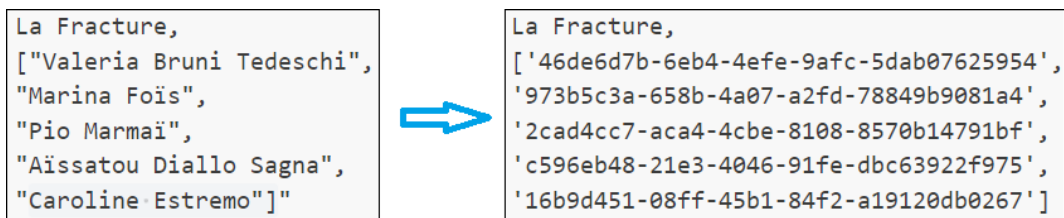
Chaque enregistrement aura un id unique créé avec la librairie **uuid**.

```

1 id = str(uuid.uuid4())
2 f"INSERT INTO actors (actor_id, actor_name) VALUES (%s, %s)"
3 val = (id, "Cameron Diaz")
4 cursor.execute(query, val)

```

Ensuite nous parcourons la liste de nos films, pour chaque film nous remplaçons la liste d'acteurs par la liste des ids d'acteurs.



Ensuite pour chaque film nous générons un id de film, nous disposons alors des couples (movie\_id, actor\_id) pour tous les acteurs ayant joué dans le film, nous remplissons donc la table de jonction **actor\_movie**, idem avec les autres tables de jonction.

title	actor_name
Finch	Skeet Ulrich
Finch	Lora Martinez-Cunningham
Finch	Oscar Avila
Finch	Marie Wagenman
Finch	Tom Hanks
Finch	Laura Harrier

→

movie_id	actor_id
11a1697e-4734-4b52-a06a-5142a767fe0d	09357251-04f8-4f6a-b59e-94196906daa0
11a1697e-4734-4b52-a06a-5142a767fe0d	12cbcda8-5c6e-4c1e-a6a3-d23628c847eb
11a1697e-4734-4b52-a06a-5142a767fe0d	890bddaa-87a6-41a8-b207-14c8274539b0
11a1697e-4734-4b52-a06a-5142a767fe0d	98835973-ace4-4665-8041-7960f51aca62
11a1697e-4734-4b52-a06a-5142a767fe0d	db463ddf-39c8-4ea8-9ef1-5a6dcae572f0
11a1697e-4734-4b52-a06a-5142a767fe0d	edfdf753-58a8-4091-a5c0-4d8b13cfeb5b

## 6.5 Création et remplissage de la base MongoDB

Nous créons nos bases et collections directement en python à l'aide de la librairie **pymongo** et nous insérons les documents à partir des dataframe Pandas.

```
1 # Connect to MongoDB
2 client = pymongo.MongoClient("mongodb://localhost:27017/")
3
4 # Create database "allocine" (or selects it if already exists)
5 mydb = client["allocine"]
6
7 # Create a collection "movies" (or select it if already exists)
8 col_movies = mydb["movies"]
9
10 # Insertion of movie plots in MongoDB database
11 col_movies.insert_many(df_movies.to_dict(orient='records'))
```

## 6.6 Exemples de requêtes SQL

**Requête** : Lister les films avec **Pierre Richard** et dont la musique a été composée par **Vladimir Cosma**.

```
SELECT m.title AS 'title',
       m.release_date AS 'date',
       d.director_name AS 'director'
FROM movies AS m
JOIN actor_movie AS am ON am.movie_id = m.movie_id
JOIN actors AS a ON a.actor_id = am.actor_id
JOIN composer_movie AS cm ON cm.movie_id = m.movie_id
JOIN composers AS c ON c.composer_id = cm.composer_id
JOIN director_movie AS dm ON dm.movie_id = m.movie_id
JOIN directors AS d ON d.director_id = dm.director_id
WHERE a.actor_name = 'Pierre Richard'
AND c.composer_name = 'Vladimir Cosma'
ORDER BY date;
```

	title	date	director
	Alexandre le Bienheureux	1968-02-09	Yves Robert
	Le Distrait	1970-12-09	Pierre Richard
	Le Grand Blond avec une chaussure noire	1972-12-05	Yves Robert
	La Moutarde me monte au nez	1974-10-09	Claude Zidi
	Le Retour du grand blond	1974-12-18	Yves Robert
	Le Jouet	1976-12-08	Francis Veber
	Je suis timide, mais je me soigne	1978-08-23	Pierre Richard
	C'est pas moi, c'est lui	1980-01-23	Pierre Richard
	Le coup du parapluie	1980-10-08	Gérard Oury
	La Chèvre	1981-12-09	Francis Veber
	Les compères	1983-11-23	Francis Veber
	Le Jumeau	1984-10-10	Yves Robert
	Les Rois du gag	1985-03-06	Claude Zidi
	Les Fugitifs	1986-12-17	Francis Veber
	Les Malheurs d'Alfred	2011-04-01	Pierre Richard
	La course à l'échalote	2019-12-11	Claude Zidi

**Requête** : Afficher les acteurs ayant jouer dans plusieurs films de la franchise **Terminator**.

```
SELECT DISTINCT a.actor_name, COUNT(*)
FROM actors AS a
JOIN actor_movie AS am ON am.actor_id = a.actor_id
JOIN movies AS m ON m.movie_id = am.movie_id
WHERE m.title LIKE '%Terminator%'
GROUP BY a.actor_name
HAVING COUNT(*) > 1
ORDER BY a.actor_name;
```

name	nb
Arnold Schwarzenegger	5
Earl Boen	3
Edward Furlong	2
Linda Hamilton	3
Michael Papajohn	2

## 7 Création d'une API

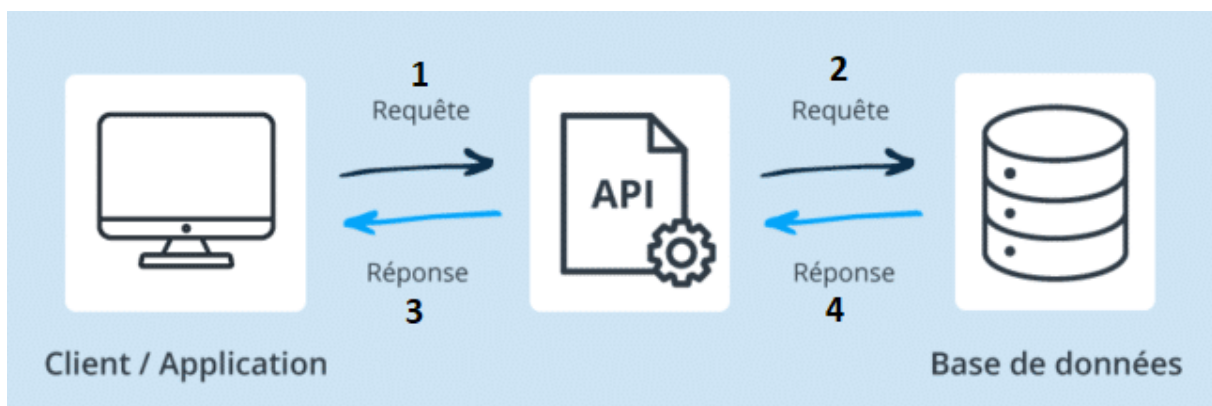
### 7.1 Généralités sur les APIs

Une **API** est un programme permettant à deux logiciels (ou programmes) de communiquer entre eux. Dans notre cas, l'objectif de l'API est d'exposer notre base de données à l'utilisateur sans que celui-ci ne connaisse les noms des tables ou des colonnes de la base, l'API fait donc la communication entre la base de données et l'interface utilisateur.

Nous utilisons le framework python **Fast API** pour la créer une API de type **REST**. Le framework **Fast API** est connu pour sa rapidité et génère automatiquement la documentation de l'API et l'interface utilisateur pour utiliser l'API. L'utilisateur n'aura qu'à consulter la documentation puis à renseigner les champs pour requêter la base de donnée.

#### Fonctionnement de l'API

- 1 Requête : L'utilisateur remplit des champs (acteurs, réalisateurs ...) à partir de l'interface créée par FastAPI, cela génère automatiquement une requête,
- 2 Traitement : L'API reçoit la requête, la traite en interrogeant la base de données,
- 3 Réponse de la requête : La base renvoie les données demandées à l'API,
- 4 Réponse de l'API : L'API renvoie le résultat au format JSON au client.



### 7.2 Création de l'API

Nous définissons dans un script **api.py** des **fonctions routes** pour les différentes requêtes que nous souhaitons mettre à disposition de l'utilisateur.

```
1 @app.get("/movies")
2 async def get_movies(
3     actor1: Optional[str] = Query(None, alias="actor1", description="filtrer
4     les films avec 'actor1'"),
5     actor2: Optional[str] = Query(None, alias="actor2", description="filtrer
6     les films avec 'actor2'"),
7     producer: Optional[str] = Query(None, alias="producer", description="
8     filtrer les films realises par 'producer'"),
9 )
```

Une fois l'API lancée avec la commande `uvicorn api:app -reload`, l'interface utilisateur ainsi que la documentation de l'api sont disponibles à l'adresse `http://127.0.0.1:8000/docs`.

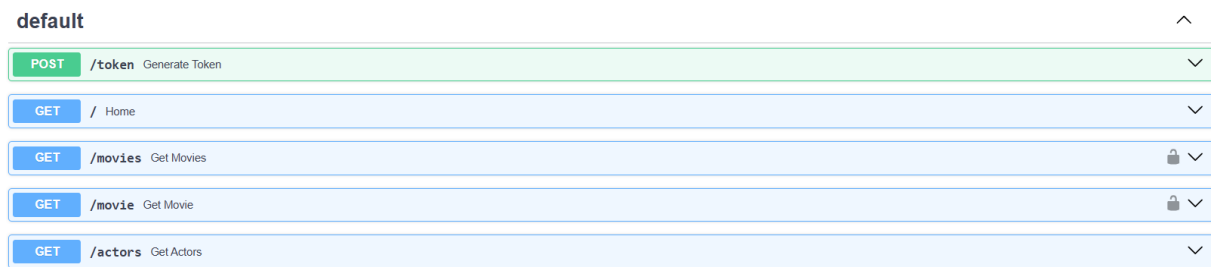


FIGURE 11 – Routes

La documentation des routes est automatiquement générée par **Fast API** dans le standard des documentations **Open API**. L'utilisateur n'a plus qu'à consulter la documentation et à remplir les champs pour obtenir les données.



FIGURE 12 – Documentation API

## 7.3 Règle d'authentification

Nous ajoutons un système d'authentification pour que notre API ne soit disponible qu'aux utilisateurs possédant un mot de passe.

Nous utilisons le système de Tokenisation JWT (JSON Web Token), c'est un standard pour communiquer des données sécurisées à travers des objets JSON (signée numériquement).

### Fonctionnement de JWT :

- 1 Le client (navigateur) envoie au serveur les informations d'authentification (password),
- 2 si l'authentification est acceptée, le serveur génère un jeton JWT (signé avec une clé secrète) et le transmet au client,
- 3 le client peut envoyer des requêtes pour accéder à des ressources, il y joint son token,
- 4 si le token est bon, le serveur envoie au client les ressources demandées.

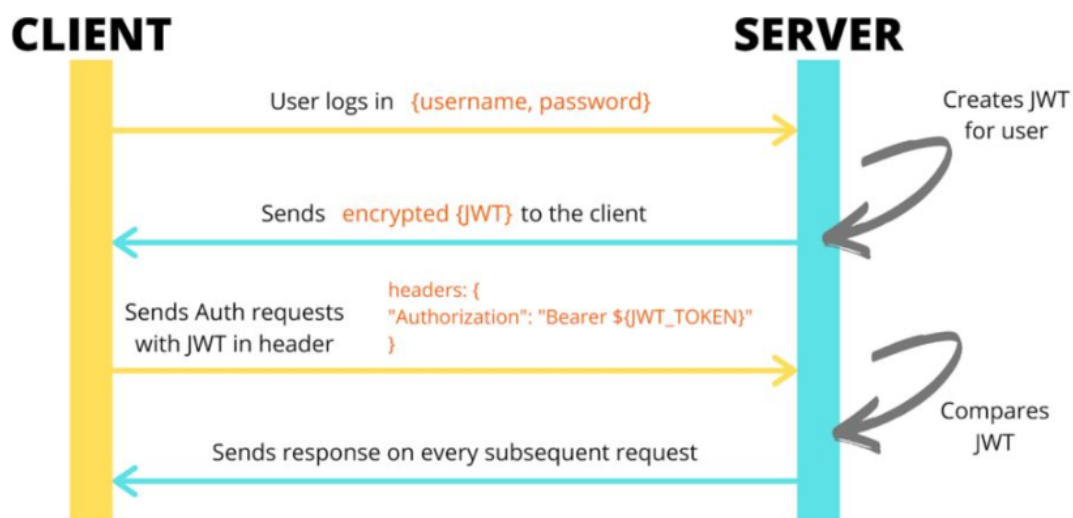


FIGURE 13 – Authentification JWT



## 7.4 Exemples d'utilisation de l'API

Un système de filtres a été mis en places pour pouvoir filtrer les films à partir de plusieurs noms d'acteurs et/ou du réalisateur et/ou du compositeur, **des filtres SQL** pour faire des jointures entre les tables de films, acteurs, réalisateurs et compositeurs et un **filtre pymongo** pour extraire des informations de la base MongoDB (**Synopsis** ou **url de l'affiche**).

GET	/movies	Get Movies
actor1	filtrer les films où a joué 'actor1'	
string   (string   null)		<input type="text" value="danny glover"/>
(query)		
actor2	filtrer les films où a joué 'actor2'	
string   (string   null)		<input type="text" value="mel gibson"/>
(query)		

Code	Details
200	<p>Response body</p> <pre>[   {     "movie_id": "5ea69def-2d58-4268-a381-ad982c6ec98a",     "title": "L'Arme fatale",     "release_date": "1987-08-05"   },   {     "movie_id": "889bb96b-5791-4ddb-a8db-f3a57975b7e2",     "title": "L'Arme fatale 3",     "release_date": "1992-08-12"   },   {     "movie_id": "9ced673a-95bb-4869-81bd-a56b1adbfaa9",     "title": "L'Arme fatale 4",     "release_date": "1998-07-22"   },   {     "movie_id": "c247d44b-6bb9-4ba9-a1fc-94f8ce96675e",     "title": "Maverick",     "release_date": "1994-08-03"   },   {     "movie_id": "fa6dcca0-239f-428d-a357-7a8bf71e9a30",     "title": "L'Arme fatale 2",     "release_date": "1989-08-02"   } ]</pre>

FIGURE 14 – Requête intersection SQL

GET	/movie	Get Movie
Affiche les informations d'un film		
Paramètre : id: identifiant du film recherché.		
Parameters		
Name	Description	
id	identifiant du film recherché	
string   (string   null)		<input type="text" value="5ea69def-2d58-4268-a381-ad982c6ec98a"/>
Response body		
<pre>{   "movie": [     {       "title": "L'Arme fatale",       "release_date": "1987-08-05",       "url_thumbnail": "https://m.media-amazon.com/images/M/MV5BMWVlNmZlODktMzhhNS00YTdhLWZyZWYyZU0NDfMzZk0MGVlXkEyXkFqcGc@._V1_SX300.jpg",       "plot": "Two newly paired cops who are complete opposites must put aside their differences in order to catch a gang of drug smugglers."     }   ],   "actors": [     {       "actor_id": "113ddf76-6373-4bd0-b2e6-64282afeb087",       "actor_name": "Steve Kahan"     },     {       "actor_id": "11c5e3d9-d1bb-4539-ab7e-133b871d1e85",       "actor_name": "Mel Gibson"     },     {       "actor_id": "16a8bc/d-b966-4cad-a3f5-c8046//2C9c5",       "actor_name": "Lycia Naff"     }   ] }</pre>		


  


FIGURE 15 – Requête MongoDB

## 8 Automatisation

### 8.1 Présentation de Crontab

**Crontab** est une application d'automatisation de tâches sur système Unix et Linux, plus précisément elle permet l'exécution de scripts en arrière-plan à des moments précis. Nous utiliserons Crontab pour collecter les informations des nouveaux films de la semaine. pour cela nous créerons un environnement sur une machine virtuelle Linux WSL puis une tâche qui exécutera notre script tous les mercredis matins à 8h (date de sortie des films).

### 8.2 Création d'un environnement virtuel sous WSL

```
python -m venv env_allocine
```

Activation de l'environnement

```
source env_allocine/bin/activate
```

Installation des packages nécessaires à l'exécution du script

```
pip install pandas requests beautifulsoup4 mysql-connector-python
```

### 8.3 Script à exécuter

Notre script comportera les étapes suivantes :

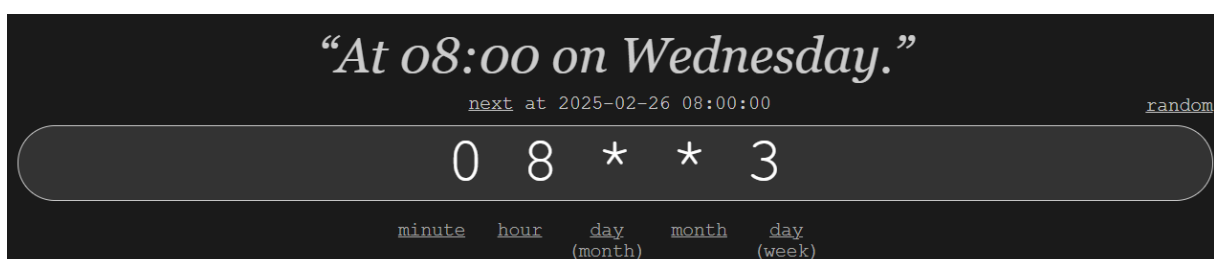
- Scrapping des films de la semaine à partir de l'url dédiée sur allocine.fr,
- Requête de OMDB pour récupérer des informations supplémentaires,
- Enregistrement des données en fichiers CSV.

### 8.4 Création d'une tâche Crontab

Préparation de l'environnement et création de la tâche Crontab

- 1 Création sous WSL d'un environnement virtuel venv : `python -m venv env`
- 2 Activer l'environnement : `source /env/bin/activate`
- 3 Installer les librairies nécessaires à l'utilisation du script "numpy", "pandas", "requests", "beautifulsoup4", "httpx", "selenium", "mysql-connector-python"
- 4 Copie du script.py dans WSL
- 5 Création de la tâche crontab qui doit se lancer chaque mercredi à 8 heures du matin, ajout d'information dans un fichier log.

```
0 8 3 * * cd /home/franck/testCron && . ~/testCron/env_allocine/bin/activate
&& cd /home/franck/testCron && python script.py »
~/testCron/allocine.log
```



Minute (0-59), Heure (0-23), Jour du mois (1-31), Mois (1-12), Jour de la semaine (0-7, où 0 et 7 représentent dimanche)

Problème lors de l'exécution de la tâche : il ne trouve pas mes modules python