Description technique

Data Engineering

De Smedt Chloé

Rochon Juliette

Vue d'ensemble

Dans le cadre de l'unité Data engineering nous avons réalisé un projet en binôme ayant comme objectifs de récupérer les données d'un site web, de les stocker et de les mettre en avant sur un site programmé avec Flask. Notre choix s'était arrêté sur le site d'Allociné et sur la récupération des films de Noël, afin de créer un moteur de recherche dédié aux différents films sur le thème de Noël. Cependant, trop de données manquantes rendaient le traitement de ces dernières très complexe. C'est la raison pour laquelle nous avons préféré scraper les données de SensCritique. Nous n'avons scrapé que la première page par souci de temps.

Nous avons choisi ce sujet car il a été montré que les films de noël sont bons pour le moral, et qu'en période de covid, les films de noël ont été plus regardé que durant les autres années : 20% des personnes ont commencé à en regarder dès le mois d'aout, 37% durant le mois de septembre, et 80% des gens interrogés ont déclaré qu'ils ont regardé plus de films de noël en 2020 que durant les autres années

Le moteur de recherche que nous avons créé a pour objectif d'être visuel et simple d'utilisation.

Flask

Une des consigne principale était de réaliser l'application web avec le framework python Flask. Pour être plus précis c'est un microframework open-source qui comprend un certain nombre d'extensions.

C'est notre fichier "chrismasMovies.py" présent dans le premier dossier newscrawler qui comporte tout le code concernant l'application web et les différentes requêtes. Nous avons défini plusieurs routes menant aux différentes pages. La page not_found affiche ERROR, la home_page est l'accueil du site web, la search_page contient le moteur de recherche et la info_page réunit différentes informations complémentaires.

Nous avons décidé d'utiliser un template pour la partie frontend de l'application afin d'avoir un design particulier. C'est sur le site https://templated.co/ que nous avons obtenu un template HTML et CSS complet que nous avons modifié par la suite pour qu'il s'adapte à nos besoins.

Scrapy & ElasticSearch

I. Récupération des données avec Scrapy

Le scraping des données est une des parties les plus importantes du projet, nous utilisons une bibliothèque nommée Scrapy qui permet aux développeurs de scraper/récupérer les données d'un site web.

Les fichiers utiles se trouvent dans le dossier newscrawler/newscrawler/spiders.

Il y a une spider qui renvoie des informations qui contiennent le titre du film et sa photo, et lorsqu'elles sont renseignées, la moyenne des notes que le film a reçues.

Pour exécuter le scraping à la main, il faut indiquer dans la console la commande : `scrapy crawl cinespi`. En effet, `cinespi` est le nom de notre spider.

Afin de pouvoir récupérer les données de SensCritique il a fallu modifier le Obey robot.txt en False car il nous en empêchait. Ce paramètre se trouve dans le fichier settings.py

```
20
21  # Obey robots.txt rules
22  ROBOTSTXT_OBEY = False
23
```

Il aurait été possible de scraper des données en temps réel. Cependant, nous n'avons pas trouvé cela nécessaire. Effectivement, contrairement à des sites où le temps réel est nécessaire (par une actualisation fréquente de ses informations notamment), SensCritique n'actualise pas ses films de Noël assez régulièrement pour que ce soit pertinent de scraper en temps réel.

II. Recherche des données avec ElasticSearch

ElasticSearch est un moteur de recherche créé pour l'indexation et la recherche de données. Un de ses avantages est qu'il permet de faire de la recherche en texte libre. C'est la raison pour laquelle nous avons préféré l'utiliser. Si nous avions eu plus de temps, nous aurions aimé pouvoir coupler ElasticSearch et MongoDB. Ainsi, nous aurions pu bénéficier de l'avantage de MongoDB sur la mémoire.

Nous avons réalisé deux requêtes ElasticSearch au sein de ce projet.

Une requête a été utilisée pour sélectionner les 10 films de la base de données ayant obtenus les meilleures notes.

Une deuxième requête a été utilisée pour faire les requêtes liées au moteur de recherche. L'utilisateur rentre dans la barre de recherche un mot en lien avec ce qui l'intéresse et la requête va extraire tous les films contenant ce mot.

Docker

Dans ce projet, nous avons créé un conteneur Flask, et un conteneur ElasticSearch. Pour les relier, il a fallu utiliser Docker-compose.

Dans le cas où on ne cherche à lancer que le conteneur Flask, il faut renommer le fichier Dockerfile_flask en Dockerfile.

Ensuite, il faut taper la commande `docker build -t dockerfile_flask .` pour le build. Puis, pour le run, nous devons utiliser la commande `docker run -d -p 5000:5000 dockerfile_flask`. Cela signifie que sur le port 5000 de notre ordinateur tourne le conteneur (ce port est le port de référence pour les applications Flask).

Pour ce conteneur, le souci rencontré a été dans le fichier de notre application, nommé `chrismasMovies.py`. En effet, le problème venait du fait qu'on ne se liait qu'à l'interface localhost, alors que nous devions être liées à 0.0.0.0 pour que le conteneur soit accessible de l'extérieur. Il a donc fallu remplacer app.run() par app.run(host='0.0.0.0').

Le moteur de recherche et le projet sera disponible à l'URL suivant : http://localhost:5000 (ou http://127.0.0.1:5000/).

Docker compose

Nous avons vu qu'il était nécessaire d'avoir le fichier Dockerfile_flask qui est un conteneur qui contient l'application flask et la lance sur le port 5000, port de référence pour les applications flask.

Nous avons aussi créé le conteneur elastic, qui est une image provenant de Docker Hub. Elle utilise le port 9200 de notre machine.

Pour lancer le projet, il faut taper la commande `docker-compose up -d --build`. Si le docker-compose a déjà été build au préalable, on préfèrera utiliser la commande `docker-compose up -d` qui ne build pas les conteneurs. Pour stopper les conteneurs, on utilisera la commande `docker-compose down`.

Ensuite, on devra exécuter le fichier chrismasMovie.py et ensuite ouvrir le lien http://127.0.0.1:5000/.