Notes au sujet du projet de programmation

Laure Camiat

13 décembre 2019

1 CHOIX TECHNIQUES ET ARCHITECTURAUX

Pour réaliser ce projet, j'ai choisi d'utiliser le langage Python (version 3.6.3). Suite à des difficultés systèmes et une installation Linux qui s'est avérée impossible (dual-boot et machine virtuelle), je me suis résolue à travailler sous Windows 10. L'ensemble des commandes à exécuter sont donc adaptées à ce système d'exploitation.

Pour monter l'application, j'ai eu recours à *Flask* pour monter le serveur web, *SQLAlchemy-Marshmallow* pour gérer le mapping objet-relationnel ainsi que la (dé)sérialisation des données et j'ai utilisé *SQLite* pour créer la base de données.

La base de données est constituée de deux tables. La première table permet de stocker les domaines crawlés et de les associer au job_id généré lors du crawling.

domain_id	**job_id**	**url**
1	5cdca9d5-cb27-4a7c-83b4-b52bf4444739	https ://www.mims.ai/
2	5cdca9d5-cb27-4a7c-83b4-b52bf4444739	https ://grandsballets.com/
...	...	...

La seconde table permet d'enregistrer les images récupérées lors du crawling et de les associer également au job_id correspondant.

image_id	**job_id**	**url**
1	5cdca9d5-cb27-4a7c-83b4-b52bf4444739	http ://www.url.ca/image1.png
2	5cdca9d5-cb27-4a7c-83b4-b52bf4444739	http ://www.url.ca/image2.jpg
...	...	...

2 AXES D'AMÉLIORATION

J'ai détecté plusieurs points sujet à amélioration :

— La gestion d'erreur n'a été que sommairement traitée. Il serait nécessaire de gérer par exemple les cas où les paramètres entrés lors du lancement de la commande avec "curl" ne sont pas correctement renseignés. Il faudrait aussi gérer les erreurs qui peuvent apparaître côté base de données.
— En ce qui concerne la manière dont j'ai stocké les données en base, je ne suis pas très satisfaite avec les choix de table et attribut que j'ai faits. Il existe certainement un moyen d'enregistrer les données de façon plus intelligente et moins répétitive.
— Algorithmiquement, il y aurait sûrement des moyens plus simples de procéder au crawling en utilisant la bibliothèque *Requests-HTML* ou la bibliothèque d'opérations sur les expressions régulières *re*. J'aurais souhaité utiliser les arbres avec des regex pour éviter des filtres supplémentaires à appliquer mais je n'ai pas réussi à trouver la bonne syntaxe me permettant d'y parvenir.
— La construction des fichiers JSON est à modifier aussi pour éviter la création de listes secondaires (partie du code ci-dessous) et des itérations sur de gros volumes.

```
urls = list()
for image in images:
    urls.append(image.get('url'))

crawled_domain = list()
for domain in domains:
    crawled_domain.append(domain.get('url'))
```

Si l'on souhaitait aller plus loin, on aurait très bien pu imaginer incorporer une interface graphique à cette application permettant de visualiser les images collectées sur le domaine, en s'assurant d'en afficher une quantité limitée pour éviter des problèmes de latence.

3 LACUNES

Comme je l'ai fait savoir dans l'un des mails que j'ai fait parvenir la semaine dernière, je n'ai pas pu installer Docker sur ma machine. Le fichier **README.md** contient l'ensemble des commandes à réaliser pour pouvoir lancer l'application.

Le code que j'ai écrit est dépourvu de tests, mais il est clair que tout développement informatique se doit d'en avoir. Il aurait été intéressant de réaliser ce projet en TDD avec le framework *Unittest*. Cela aurait très certainement eu l'avantage de m'aider à écrire les parties de l'algorithme de crawl qui m'ont posé le plus de difficultés.

L'application n'est pas thread-safe. Je pensais que l'ajout d'un verrou suffirait à protéger la variable nb_thread des accès et des modifications concurrentes mais j'ai observé des erreurs à ce niveau là dans la mesure où, quand j'exécute l'application avec n threads, j'ai en moyenne 2n-1 threads lancés.

Enfin, concernant la méthode GET, je ne suis pas parvenue à implémenter une méthode permettant de suivre en temps réel l'avancement du crawling des urls. La méthode, telle que je l'ai écrite, permet de récupérer l'ensemble des informations énoncées dans le sujet une fois que la méthode POST a rendu réponse, mais pas avant.