Rapport TP Web Scraping

Cécile CESA

Lucie GUILLAUD SAUMUR

Valentin PACHURKA

# I – Introduction

Eléments centraux :

* Modules Python (**pymongo**, **requests**, **beautifulsoup**, **time**, **urllib.parse**, **threading**, time)

Scraping Web:

* Récupérer une page web à une URL donnée et récupérer tous les liens de la page web
* Recommencer récursivement sur ces nouveaux liens
* La même page web ne peut pas être récupéré plusieurs fois pour une même session de scraping
* Limiter le scope (le site) se limiter à un certain nom de domaine ou sous domaine ou à un certain préfixe de répertoires dans l’url
* Plusieurs sessions en parallèle pour deux sites distincts ou en séquence pour le même site
* Limite (10) du nombre max d’URL distinctes à récupérer, pour une session donnée
* Réutiliser les cookies envoyés par le serveur pour les requêtes des pages liées
* Gérer les erreurs réessayer à 60 seconde d’intervalle. Recommencer max 10 fois.

Fonctionnement distribué :

* Mutualiser les ressources (machine et réseau) de plusieurs machines
* Répartir la charge
* Utiliser liste d’attente d’URL scapées dans une base de données
* Conserver une trace des évènements

Tolérance aux pannes :

* Détecter un scrapeur en panne
* Reprendre le traitement

Prétraitement du contenu :

* Extraire les titres **<title>, <h1>, <h2>**
* Extraire les emphases **<b>, <strong>, <em>**

Interface :

* Programme en ligne de commande (**argparse**)
* API **flask**

CAHIER DES CHARGES

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lundi 03/07 | | Mardi 04/07 | | Mercredi 05/07 | | Jeudi 06/07 | | Vendredi 07/07 | |
| Matin | Ap midi | Matin | Ap midi | Matin | Ap midi | Matin | Ap midi | Matin | Ap midi |
| Scraping web | Scraping web | Scraping web | Scraping web | Fonctionnement distribué |  | Livrable intermédiaire |  |  | Livraison finale |
| Prétraitement du contenu | Prétraitement du contenu | Mise en place de fonction | Fonctionnement distribué | Architecture MongoDB |  | Documentation sur archi |  |  | Présentation |
|  |  |  |  |  |  | Première version fonctionnelle |  |  |  |

# II - Architecture du code

Version du 06/07/2023

Nous avons créé un script de scraping web écrit en Python. Il utilise différentes bibliothèques telles que **requests**, **BeautifulSoup**, **pymongo**, **threading**, et **queue** pour extraire des informations à partir de pages web et les enregistrer dans une base de données MongoDB.

Quelques explications du code

1 - Les imports :

* **import sys** : permet d'accéder à certaines fonctionnalités du système.
* **import requests** : utilisé pour envoyer des requêtes HTTP et récupérer le contenu des pages web.
* **from bs4 import BeautifulSoup** : fournit des fonctionnalités pour extraire des données HTML et XML.
* **from pymongo import MongoClient** : permet d'interagir avec une base de données MongoDB.
* **from urllib.parse import urljoin, urldefrag, urlparse** : utilisé pour manipuler et normaliser les URL.
* **import threading** : utilisé pour exécuter plusieurs tâches en parallèle (threading).
* **from queue import Queue** : fournit une file d'attente pour gérer les URL à traiter.
* **import time** : fournit des fonctionnalités pour la gestion du temps.

2 - La classe **Scraper** :

* Le constructeur **\_\_init\_\_** : initialise les attributs de la classe, notamment les URLs de départ, la profondeur maximale de recherche, les limites de domaine, le préfixe du répertoire, et les connexions à la base de données MongoDB.
* La méthode **scrape\_website** : c'est la méthode principale du scraping. Elle met en place la file d'attente d'URLs, lance les threads de scraping et attend que toutes les URLs soient traitées.
* La méthode privée **\_scrape\_links** : cette méthode est exécutée par chaque thread de scraping. Elle récupère une URL à partir de la file d'attente, envoie une requête HTTP pour récupérer la page correspondante, extrait les liens et les métadonnées, et les enregistre dans la base de données MongoDB.
* Les autres méthodes privées (**\_get\_url\_links**, **\_insert\_links**, **\_insert\_metadata**, **\_insert\_pending\_links**, **\_insert\_journal**) sont utilisées par la méthode **\_scrape\_links** pour effectuer des opérations spécifiques comme l'extraction des liens, l'insertion des données dans la base de données, etc.

3 - La valeur **time\_threshold** : il s'agit d'une constante définissant une durée (en secondes). Elle est utilisée pour vérifier si une URL a été visitée récemment afin d'éviter les visites fréquentes.

4 - URLs de départ : dans cet exemple, il y a une seule URL de départ qui est **'**[**https://fr.wikipedia.org/wiki/France**](https://fr.wikipedia.org/wiki/France)**'** (page de l'article France sur Wikipédia).

5 - Création d'une instance de la classe **Scraper** : une instance de la classe **Scraper** est créé avec les URLs de départ, une profondeur maximale de 3 (c'est-à-dire qu'elle suivra les liens jusqu'à 3 niveaux de profondeur), la limite de domaine définie sur **'fr.wikipedia.org'** et le préfixe de répertoire défini sur **'/'**.

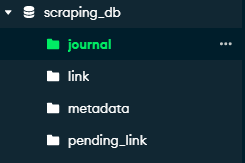
* L'appel à la méthode **scrape\_website** : cela démarre le processus de scraping en appelant la méthode **scrape\_website** de l'instance **scraper** créée précédemment.

En résumé, ce code représente un scraper web qui récupère les liens et les métadonnées des pages web à partir d'une liste d'URLs de départ spécifiée, puis les enregistre dans une base de données MongoDB. Il utilise des threads pour effectuer le scraping en parallèle, ce qui permet de traiter plusieurs URLs simultanément.

# III – Database MongoDB

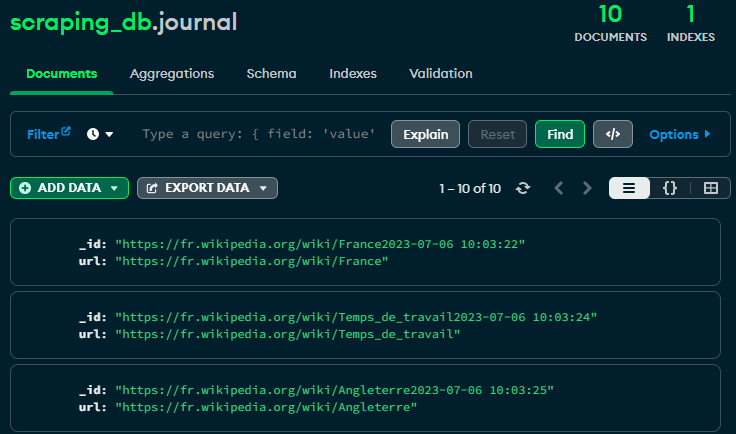
Voici un visuel dans MongoDB du rendu de notre code :

1 – Database nommée **scraping\_db** et 4 collections **journal**, **link**, **metadata** et **pending**\_link

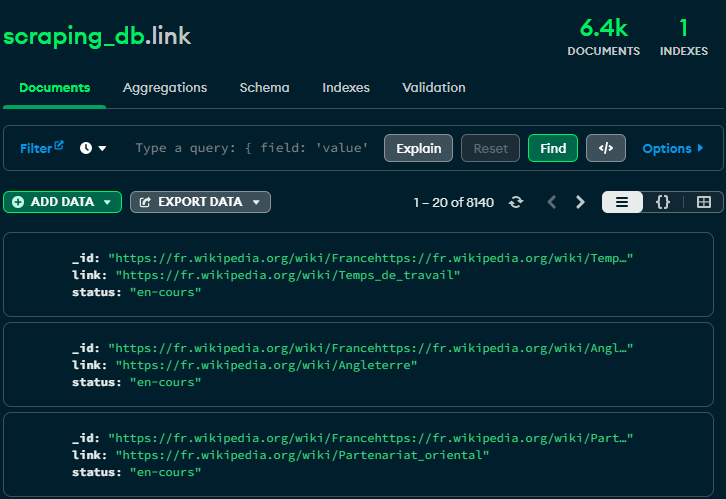


2 – **Journal** permet d’enregistrer les liens déjà scrapper et la date à laquelle cela a été fait.

Dans la collection **journal**, nous avons enregistré dans un champ **\_id** (url de la page et la date à laquelle il a été récupéré) et un champ url (url de la page).

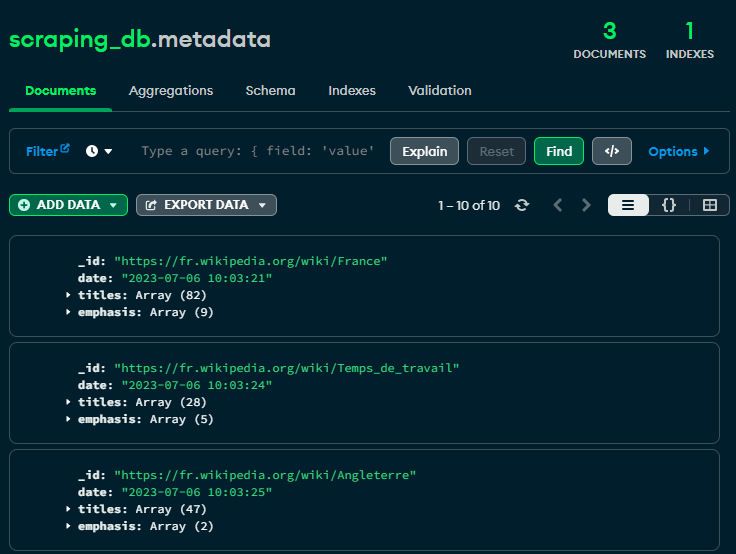


3 – **Link** permet d’enregistrer les liens générés par **l’url d’origine** avec le **status** du lien (scrapper ou pas encore)

Dans la collection **link**, nous avons un champ **\_id** (url de la page d’origine, url de la page générée et la date à laquelle il a été récupéré), **link** (url de la page générée) et **status** (en-cours ou fini).

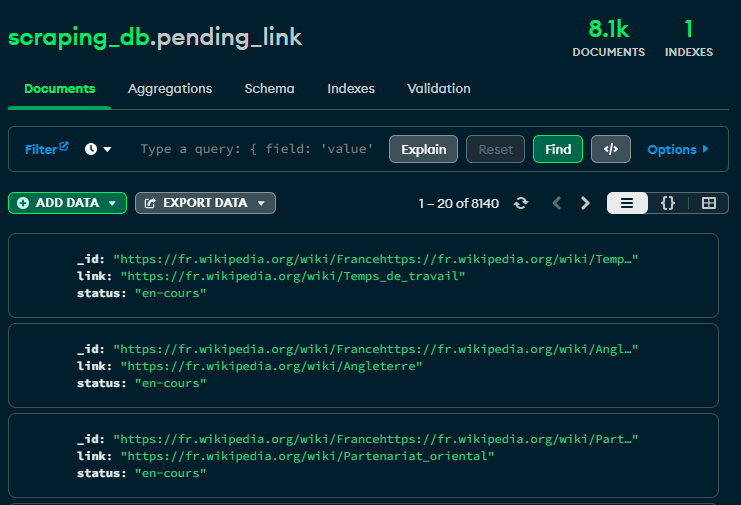
3 – **Metadata** permet de récupérer les métadata (balises HTML **titles** et **emphasis**) de chaque lien avec la date de création du document.

Nous avons le champ **\_id** qui correspond à **l’url** de la page traitée, le champ **date**, le champ **titles** (balises titles) et le champ **emphasis** (balises emphasis).



4 – **Pending\_link** qui correspond aux liens qui restent à traiter

Nous avons le champ **\_id** qui correspond à l’url de la page d’origine, l’url de la page générée et la date (concaténé), le champ **link** (url de la page généré) et **status** (en-cours ou fini)



# IV - Améliorations en cours

1 – Nous allons probablement ne pas garder la collection **pending\_link** pour des raisons de redondance.

2 – Pour le moment, le programme fonctionne bien avec MongoDB mais ne s’arrête pas dans la console Python malgré une limitation de 10 documents dans la collection **metadata**.

3 – Il faut faire communiquer les collections afin d’utiliser le champ **status**. Pour le moment, si on relance une session il ne prend pas en compte les liens déjà scrapper et ceux qui doivent l’être.

4 – Il nous faut ajouter une fonctionnalité permettant de pouvoir faire une **session sur plusieurs URL dès le départ**.

5 – Interface **flask**

6 - Revoir l’**id des sessions**

7 – Il nous faut également aborder la **tolérance aux pannes** et la **distribution des ressources** (machines, serveurs).