# **DOCUMENTATION TECHNIQUE HIRING CHALLENGE DATA ANALYST**

## IDENTIFICATION DES SOURCES DE DONNEES

### ARXIV.ORG

#### Motivation du choix

arXiv est une archive ouverte de prépublications électroniques d'articles scientifiques dans les domaines de la physique, des mathématiques, de l'informatique, de la biologie quantitative, de la finance quantitative, de la statistique, de l'ingénierie électrique et des systèmes, et de l'économie1, et qui est accessible gratuitement par Internet.

Ayant été proposée comme source de donnée dans le challenge et s’avérant être une source sure pour la collecte d’informations, le site a bien entendu été sélectionné comme source de données.

L’url qui nous intéresse particulièrement est la suivante :

https://arxiv.org/search/?query=cybersecurity&searchtype=all&abstracts=show&order=-announced\_date\_first&size=50

elle donne tous les résultats de recherche relatifs à la cybersécurité.

#### Avantages

* non js-heavy
* aucun blocage d’adresse IP, ni d’header (pas besoin d’utiliser fake user agent, ni de proxy generator)

#### Contraintes

Pas de contraintes particulières

### SCHOLAR.GOOGLE.COM

#### Motivation du choix

Google Scholar est un moteur de recherche Web accessible gratuitement qui indexe le texte intégral ou les métadonnées de la littérature scientifique dans un large éventail de formats et de disciplines de publication. Publié en version bêta en novembre 2004, l'index Google Scholar comprend des revues et des livres universitaires en ligne évalués par des pairs, des documents de conférence, des thèses et des mémoires, des prépublications, des résumés, des rapports techniques et d'autres publications scientifiques, y compris des avis de tribunaux et des brevets.

Cette plateforme s’inscrit comme étant la plateforme de référence pour la recherche d’articles scientifique. Bien que recommandée dans l’intitulé du challenge, ne pas la sélectionner limiterais considérablement le champ de recherche de l’étude.

L’url qui nous intéresse particulièrement est la suivante :

<https://scholar.google.com/scholar?hl=fr&as_sdt=0%2C5&as_vis=1&q=cybersecurity&btnG=>

elle donne tous les résultats de recherche relatifs à la cybersécurité.

#### Avantages

* Grand nombre de résultats
* Non js-heavy
* Multi sources
* Accessible à tous

#### Contraintes

* Bloqueur de robot
* Bloqueur d’api
* Bloqueur d’adresse IP et d’header (besoin de proxy generator et de fake user agent)
* Grand pool d’adresse IP bloqué
* Erreur se serveur à partir de la 99e page de résultats

### SCIENCEDIRECT.COM

#### Motivation du choix

ScienceDirect est un site web géré par l'éditeur Elsevier. Lancée en mars 1997, la plateforme permet d'accéder à plus de 3 800 revues académiques qui forment plus de 14 millions de publications scientifiques revues par des pairs.

Choix numéro trois en tant que source de données de notre étude, le site appartient à Elsevier B.V., [groupe éditorial](https://fr.wikipedia.org/wiki/Maison_d%27%C3%A9dition), filiale de la [multinationale](https://fr.wikipedia.org/wiki/Multinationale) britannique [RELX](https://fr.wikipedia.org/wiki/RELX). Elsevier, créée en [1880](https://fr.wikipedia.org/wiki/1880) , est l'un des plus gros [éditeurs](https://fr.wikipedia.org/wiki/Maison_d%27%C3%A9dition) mondiaux de [littérature scientifique](https://fr.wikipedia.org/wiki/Litt%C3%A9rature_scientifique).

#### Avantages

* Grand nombre d’articles
* Bien structuré

#### Contraintes

* Js-heavy
* Possibilité de doublons avec les articles sur google scholar

## MISE EN PLACE DU SYSTÊME DE SCRAPING

### CHOIX DE LA SOLUTION DE SCRAPING

#### Beautiful soup

was developed solely for data scraping. It is a Python web scraping library with very powerful features for not just fast but efficient data scraping. This web scraping tool works with a parser and it comes with an HTML parser in the library. However, you can use it with several other Python parsers as well.

In addition, Beautiful Soup is great for extracting data out of HTML and XML web pages and also a few other markup languages. Using the software to scrape data requires knowledge of Python programming but the procedures involved aren’t very complex. Fortunately, there is well-detailed documentation on how to use the software and they even include Python scripts and codes.

Although the software was developed as a Python 2 library, it is automatically converted into Python 3 once the installation is complete. You can install Beautiful Soup on Linux and Windows computers. However, since it’s not a standard Python library, installation is very technical and might not be very easy for beginners.

#### Selenium

Cet outil a été développé pour tester des applications Web. Cependant, son utilisation va bien au-delà car il peut gérer plusieurs tâches d’automatisation. Sélénium est de trois types : Selenium WebDriver, Selenium IDE et Selenium Grid. Pour le web scraping, vous pouvez utiliser Selenium WebDriver avec les API d'automatisation du navigateur proposées.

#### Scrapy

Anciennement connu sous le nom de ScrapingHub, Scrapy est un logiciel open source pour l'extraction de données sur n'importe quel site Web. En tant que logiciel open source, de nombreuses personnes ont contribué à son développement, mais il est actuellement géré par Zyte. Scrapy est l'outil d'extraction de données de choix pour beaucoup car il est rapide, puissant et facilement extensible.

Les requêtes Web que vous envoyez à l'aide de Scrapy sont planifiées et traitées de manière asynchrone ; pendant qu'une demande est envoyée, une autre peut être envoyée même si la première n'est pas encore complètement traitée. C'est l'un des avantages majeurs du logiciel. Scrapy peut être utilisé sur plusieurs systèmes d'exploitation, notamment Windows, Mac, Linux et BSD.

#### Scrapy vs selenium vs beautifoulsoup

En comparant leurs performances, Scrapy est le meilleur des trois pour gérer des tâches complexes. Grâce à son système asynchrone, Scrapy fonctionne très rapidement et extrait les données en un temps record. Beautiful Soup est fiable mais il est lent par rapport à Scrappy. D’un autre côté, Selenium ne peut pas gérer certaines tâches de récupération de données.

### MISE EN PLACE DU PROJET SCRAPY

#### Gestionnaire de dépendances

Pip est l'outil de gestion de paquets par défaut pour Python. Il est utilisé pour installer, désinstaller et mettre à jour des paquets Python. Pip est un outil simple et facile à utiliser, mais il peut être difficile à gérer des dépendances complexes.

Poetry est un outil de gestion de paquets plus récent et plus avancé que Pip. Il offre plusieurs fonctionnalités supplémentaires, telles que la création de fichiers de verrouillage des dépendances, la publication de paquets sur PyPI, et la prise en charge de plusieurs environnements virtuels. Poetry est un outil plus complexe que Pip, mais il peut être plus efficace pour gérer des projets Python complexes.

|  |  |  |
| --- | --- | --- |
| Fonctionnalité | Pip | Poetry |
| Installation de paquets | ✓ | ✓ |
| Désinstallation de paquets | ✓ | ✓ |
| Mise à jour de paquets | ✓ | ✓ |
| Création de fichiers de verrouillage des dépendances | ✗ | ✓ |
| Publication de paquets sur PyPI | ✗ | ✓ |
| Prise en charge de plusieurs environnements virtuels | ✗ | ✓ |
| Gestion des dépendances complexes | ✗ | ✓ |

#### Initialisation du projet

##### Installation de poetry

L’installation dépend de votre environnement :

[**https://python-poetry.org/docs/#installing-with-the-official-installer**](https://python-poetry.org/docs/#installing-with-the-official-installer)

##### création un projet poetry

poetry new project\_name

##### Creation du projet scrapy

poetry add scrapy

scrapy startproject <project\_name>

#### Description des fichiers

* **settings.py** est l'endroit où sont contenus tous les paramètres de votre projet, comme l'activation des pipelines, des middlewares, etc. Ici, vous pouvez modifier les délais, la concurrence et bien plus encore.
* **items.py** est un modèle pour les données extraites. Vous pouvez définir un modèle personnalisé (comme un ProductItem) qui héritera de la classe Scrapy Item et contiendra vos données récupérées.
* **pipelines.py** est l'endroit où l'élément généré par l'araignée est transmis, il est principalement utilisé pour nettoyer le texte et se connecter aux sorties de fichiers ou aux bases de données (CSV, JSON SQL, etc.).
* **middlewares.py** est utile lorsque vous souhaitez modifier la façon dont la demande est effectuée et que Scrapy gère la réponse. scrapy.cfg est un fichier de configuration permettant de modifier certains paramètres de déploiement, etc.

### ECRITURE DES SPIDERS

#### Gscholar spider

class GscholarSpider(scrapy.Spider):  
 *"""  
 This class is a Scrapy spider that crawls Google Scholar for articles on cybersecurity.  
 """* name = "gscholar"  
 """  
 The name of the spider. This is used to identify the spider when crawling.  
 """  
 allowed\_domains = ["scholar.google.com"]  
 """  
 A list of domains that the spider is allowed to crawl.  
 """  
 start\_urls = ["https://scholar.google.com/scholar?start=980&q=cybersecurity&hl=fr&as\_sdt=0,5&as\_vis=1"]  
 """  
 A list of URLs that the spider will start crawling from.  
 """  
  
 def parse(self, response):  
 *"""  
 This method is called for each page that the spider crawls.  
 """* articles = response.css('.gs\_ri')  
 """  
 Select all articles on the page.  
 """  
 caf\_item = CafItem()  
 """  
 Create a new CafItem object to store the scraped data.  
 """  
  
 for article in articles:  
 caf\_item['link'] = article.css('.gs\_rt a ::attr(href)').getall()  
 """  
 Extract the link to the article.  
 """  
 caf\_item['authors'] = article.css('.gs\_a ::text').getall()  
 """  
 Extract the authors of the article.  
 """  
 receive = str("".join(str(elem) for elem in (caf\_item['authors'])))  
 """  
 Combine the authors into a single string.  
 """  
 year = "".join(str(elem) for elem in re.findall('\d', receive))  
 """  
 Extract the year from the author string.  
 """  
 if len(year) != 4:  
 string = "01" + year  
 else:  
 string = "01" + " Jan " + year  
 caf\_item['pub\_date'] = string  
 """  
 Format the publication date.  
 """  
 caf\_item['title'] = article.css('.gs\_rt a ::text').getall()  
 """  
 Extract the title of the article.  
 """  
 caf\_item['abstract'] = article.css('.gs\_rs ::text').getall()  
 """  
 Extract the abstract of the article.  
 """  
 caf\_item['subjects'] = "COMPUTER SCIENCE"  
 """  
 Set the subject of the article to "COMPUTER SCIENCE".  
 """  
 yield caf\_item  
 """  
 Yield the CafItem object to the pipeline.  
 """  
  
 next\_page = response.css('td[align="left"][nowrap] a ::attr(href)').get()  
 """  
 Select the link to the next page of results.  
 """  
 if next\_page is not None:  
 next\_page\_url = 'https://scholar.google.com' + next\_page  
 """  
 Construct the full URL to the next page.  
 """  
 yield response.follow(next\_page\_url, callback=self.parse)  
 """  
 Follow the link to the next page and call the `parse` method again.  
 """

#### Arxiv spider

import scrapy  
from caf.items import CafItem # Importing the CafItem class from the caf module  
import random  
  
class Arxiv2Spider(scrapy.Spider):  
 name = "arxiv2"  
 allowed\_domains = ["arxiv.org"]  
 start\_urls = ["https://arxiv.org/search/?query=cybersecurity&searchtype=all&abstracts=show&order=-announced\_date\_first&size=50"]  
  
 user\_agent\_list = [  
 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.82 Safari/537.36',  
 'Mozilla/5.0 (iPhone; CPU iPhone OS 14\_4\_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.3 Mobile/15E148 Safari/604.1',  
 'Mozilla/4.0 (compatible; MSIE 9.0; Windows NT 6.1)',  
 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.141 Safari/537.36 Edg/87.0.664.75',  
 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363',  
 ]  
  
 def parse(self, response):  
 # Extracting articles from the response  
 articles = response.css('li.arxiv-result')  
  
 for article in articles:  
 relative\_url = article.css('p.list-title a ::attr(href)').get()  
 if relative\_url is not None:  
 # Creating the absolute URL for the article  
 article\_url = 'https://arxiv.org/' + relative\_url  
 # Following the link to the article and specifying a callback function  
 yield response.follow(article\_url, callback=self.parse,  
 headers={"User-Agent": self.user\_agent\_list[random.randint(0, len(self.user\_agent\_list)-1)]})  
  
 # Extracting the URL for the next page  
 next\_page = response.css('a.pagination-next ::attr(href)').get()  
 if next\_page is not None:  
 # Creating the absolute URL for the next page  
 next\_page\_url = 'https://arxiv.org/' + next\_page  
 # Following the link to the next page and specifying a callback function  
 yield response.follow(next\_page\_url, callback=self.parse,  
 headers={"User-Agent": self.user\_agent\_list[random.randint(0, len(self.user\_agent\_list)-1)]})  
  
 def parse\_article(self, response):  
 # Extracting article information from the response  
 article = response.css('#content-inner')  
 caf\_item = CafItem()  
  
 # Extracting and assigning values to CafItem attributes  
 caf\_item['link'] = response.url  
 caf\_item['pub\_date'] = article.css('.dateline ::text').get()  
 caf\_item['title'] = article.css('h1.title ::text').getall()  
 caf\_item['authors'] = article.css('.authors ::text').getall()  
 caf\_item['abstract'] = article.css('.abstract ::text').getall()  
 caf\_item['subjects'] = article.css('td.tablecell.subjects ::text').getall()  
  
 # Yielding the CafItem  
 yield caf\_item

## TRAITEMENT ET STOCKAGE DES DONNEES

### MODIFICATION DU FICHIER PIPELINES.PY

#### Premier niveau de nettoyage des données

class CafPipeline:  
 def process\_item(self, item, spider):  
 adapter = ItemAdapter(item)  
  
 field\_names = adapter.field\_names()  
 for field\_name in field\_names:  
 value = adapter.get(field\_name)  
  
 if field\_name == 'abstract':  
 if isinstance(value, str):  
 adapter[field\_name] = value.strip()  
 adapter[field\_name] = adapter[field\_name].strip(',')  
 adapter[field\_name] = adapter[field\_name].lstrip('Abstract')  
 adapter[field\_name] = adapter[field\_name].lstrip(':')  
 adapter[field\_name] = adapter[field\_name].lstrip(',')  
 else:  
 adapter[field\_name] = "".join(str(elem) for elem in value)  
 adapter[field\_name] = adapter[field\_name].strip()  
 adapter[field\_name] = re.split(r"\s+", adapter[field\_name])  
 adapter[field\_name] = " ".join(str(elem) for elem in adapter[field\_name])  
 adapter[field\_name] = adapter[field\_name].replace("…", "")  
 adapter[field\_name] = adapter[field\_name].strip(',')  
 adapter[field\_name] = adapter[field\_name].lstrip('Abstract')  
 adapter[field\_name] = adapter[field\_name].lstrip(':')  
 adapter[field\_name] = adapter[field\_name].lstrip(',')  
 adapter[field\_name] = adapter[field\_name].strip()  
  
  
 elif field\_name == 'pub\_date':  
 if isinstance(value, str):  
 adapter[field\_name] = value.strip()  
 adapter[field\_name] = adapter[field\_name].strip(',')  
 adapter[field\_name] = adapter[field\_name].strip('[')  
 adapter[field\_name] = adapter[field\_name].strip(']')  
 adapter[field\_name] = adapter[field\_name].strip(')')  
 adapter[field\_name] = adapter[field\_name].strip('(')  
 adapter[field\_name] = adapter[field\_name].lstrip('Submitted on')  
 adapter[field\_name] = adapter[field\_name].strip()  
 adapter[field\_name] = datetime.strptime(adapter[field\_name], "%d %b %Y")  
 adapter[field\_name] = adapter[field\_name].date()  
 adapter[field\_name] = adapter[field\_name].strftime("%d/%m/%Y")  
  
 else:  
 adapter[field\_name] = "".join(str(elem) for elem in value)  
 adapter[field\_name] = adapter[field\_name].strip()  
 adapter[field\_name] = adapter[field\_name].strip(',')  
 adapter[field\_name] = adapter[field\_name].strip('[')  
 adapter[field\_name] = adapter[field\_name].strip(']')  
 adapter[field\_name] = adapter[field\_name].strip(')')  
 adapter[field\_name] = adapter[field\_name].strip('(')  
 adapter[field\_name] = adapter[field\_name].lstrip('Submitted on')  
 adapter[field\_name] = adapter[field\_name].strip()  
 adapter[field\_name] = datetime.strptime(adapter[field\_name], "%d %b %Y")  
 adapter[field\_name] = adapter[field\_name].date()  
 adapter[field\_name] = adapter[field\_name].strftime("%d/%m/%Y")  
  
 elif field\_name == 'link':  
 if isinstance(value, str):  
 adapter[field\_name] = value.strip()  
 adapter[field\_name] = adapter[field\_name].strip('{')  
 adapter[field\_name] = adapter[field\_name].strip('}')  
  
 else:  
 adapter[field\_name] = "".join(str(elem) for elem in value)  
 adapter[field\_name] = adapter[field\_name].strip()  
 adapter[field\_name] = adapter[field\_name].strip('{')  
 adapter[field\_name] = adapter[field\_name].strip('}')  
  
 elif field\_name == 'title':  
 if isinstance(value, str):  
 adapter[field\_name] = value.strip()  
 adapter[field\_name] = adapter[field\_name].lstrip('Title')  
 adapter[field\_name] = adapter[field\_name].lstrip(':')  
 else:  
 adapter[field\_name] = "".join(str(elem) for elem in value)  
 adapter[field\_name] = adapter[field\_name].strip()  
 adapter[field\_name] = adapter[field\_name].lstrip('Title')  
 adapter[field\_name] = adapter[field\_name].lstrip(':')  
  
 elif field\_name == 'authors':  
 if isinstance(value, str):  
 adapter[field\_name] = value.strip()  
 adapter[field\_name] = adapter[field\_name].lstrip('Authors')  
 adapter[field\_name] = adapter[field\_name].lstrip(':')  
 adapter[field\_name] = adapter[field\_name].lstrip(',')  
 adapter[field\_name] = ",".join(filter(None, adapter[field\_name].split(',')))  
 else:  
 adapter[field\_name] = "".join(str(elem) for elem in value)  
 adapter[field\_name] = adapter[field\_name].strip()  
 adapter[field\_name] = re.split(r"\s+", adapter[field\_name])  
 adapter[field\_name] = "".join(str(elem) for elem in adapter[field\_name])  
 adapter[field\_name] = adapter[field\_name].lstrip('Authors')  
 adapter[field\_name] = adapter[field\_name].replace(" ", "")  
 adapter[field\_name] = adapter[field\_name].replace("-", " ")  
 # adapter[field\_name] = adapter[field\_name].replace("…", "")  
 adapter[field\_name] = adapter[field\_name].lstrip(':')  
 adapter[field\_name] = adapter[field\_name].lstrip(',')  
 adapter[field\_name] = ", ".join(filter(None, adapter[field\_name].split(',')))  
 adapter[field\_name] = "".join([i for i in adapter[field\_name] if not i.isdigit()])  
  
  
 elif field\_name == 'subjects':  
 if isinstance(value, str):  
 adapter[field\_name] = value.strip()  
 adapter[field\_name] = adapter[field\_name].lstrip(',')  
 adapter[field\_name] = "".join(  
 filter(None, [segment.strip() for segment in adapter[field\_name].split(',')]))  
 else:  
 adapter[field\_name] = "".join(str(elem) for elem in value)  
 adapter[field\_name] = adapter[field\_name].strip()  
 adapter[field\_name] = adapter[field\_name].lstrip(',')  
 adapter[field\_name] = "".join(  
 filter(None, [segment.strip() for segment in adapter[field\_name].split(',')]))  
 return item

#### Stockage des données en bases

class SaveToPostgresPipeline:  
 def \_\_init\_\_(self):  
  
 # Connection Details  
 hostname = 'localhost'  
 username = 'postgres'  
 password = 'Azerty123$'  
 database = 'articles'  
  
 # Create/Connect to database  
 self.connection = psycopg2.connect(host=hostname, user=username, password=password, dbname=database)  
  
 # Create cursor, used to execute commands  
 self.cur = self.connection.cursor()  
  
 ## Create books table if none exists  
 self.cur.execute("""  
 CREATE TABLE IF NOT EXISTS articles(  
 id serial PRIMARY KEY,  
 link text,  
 pub\_date DATE,  
 title text,  
 authors text,  
 abstract text,  
 subjects text  
 )  
 """)  
 def process\_item(self, item, spider):  
  
 ## Define insert statement  
 self.cur.execute(""" insert into articles (  
 link,   
 pub\_date,   
 title,   
 authors,   
 abstract,  
 subjects  
 ) values (  
 %s,  
 %s,  
 %s,  
 %s,  
 %s,  
 %s  
 )""", (  
 item["link"],  
 item["pub\_date"],  
 item["title"],  
 item["authors"],  
 item["abstract"],  
 item["subjects"]  
 ))  
  
 # Execute insert of data into database  
 self.connection.commit()  
 return item  
  
 def close\_spider(self, spider):  
 # Close cursor & connection to database  
 self.cur.close()  
 self.connection.close()

### MODIFICATION DU FICHIER MIDDLEWARES.PY

Code pour la récupération des « fake browser header ». fournit par **scrapeops.io** et qui ne dépend pas de nous.

Elle permet de générer des headers aléatoires, ce qui permet d’éviter certains bloqueurs de robots (mais pas suffisant pour outrepasser les restrictions de google)

class ScrapeOpsFakeBrowserHeaderAgentMiddleware:  
  
 @classmethod  
 def from\_crawler(cls, crawler):  
 return cls(crawler.settings)  
  
 def \_\_init\_\_(self, settings):  
 self.scrapeops\_api\_key = settings.get('SCRAPEOPS\_API\_KEY')  
 self.scrapeops\_endpoint = settings.get('SCRAPEOPS\_FAKE\_BROWSER\_HEADER\_ENDPOINT',  
 'http://headers.scrapeops.io/v1/browser-headers?')  
 self.scrapeops\_fake\_browser\_headers\_active = settings.get('SCRAPEOPS\_FAKE\_BROWSER\_HEADER\_ENABLED', True)  
 self.scrapeops\_num\_results = settings.get('SCRAPEOPS\_NUM\_RESULTS')  
 self.headers\_list = []  
 self.\_get\_headers\_list()  
 self.\_scrapeops\_fake\_browser\_headers\_enabled()  
  
 def \_get\_headers\_list(self):  
 payload = {'api\_key': self.scrapeops\_api\_key}  
 if self.scrapeops\_num\_results is not None:  
 payload['num\_results'] = self.scrapeops\_num\_results  
 response = requests.get(self.scrapeops\_endpoint, params=urlencode(payload))  
 json\_response = response.json()  
 self.headers\_list = json\_response.get('result', [])  
  
 def \_get\_random\_browser\_header(self):  
 random\_index = randint(0, len(self.headers\_list) - 1)  
 return self.headers\_list[random\_index]  
  
 def \_scrapeops\_fake\_browser\_headers\_enabled(self):  
 if self.scrapeops\_api\_key is None or self.scrapeops\_api\_key == '' or self.scrapeops\_fake\_browser\_headers\_active == False:  
 self.scrapeops\_fake\_browser\_headers\_active = False  
 else:  
 self.scrapeops\_fake\_browser\_headers\_active = True  
  
 def process\_request(self, request, spider):  
 random\_browser\_header = self.\_get\_random\_browser\_header()  
 request.headers = Headers(random\_browser\_header)

### MODIFICATION DU FICHIER SETTINGS.PY

Paramètres de la classe précédente :

SCRAPEOPS\_API\_KEY = 'bd224ecb-62c7-4158-abea-82738f2a8a7e'  
SCRAPEOPS\_FAKE\_USER\_AGENT\_ENDPOINT = 'https://headers.scrapeops.io/v1/browser-headers'  
SCRAPEOPS\_FAKE\_USER\_AGENT\_ENABLED = True  
SCRAPEOPS\_NUM\_RESULTS = 5

Definition comme unique middleware:

DOWNLOADER\_MIDDLEWARES = {  
 # "caf.middlewares.CafDownloaderMiddleware": 543,  
 # 'scrapy\_selenium.SeleniumMiddleware': 800,  
 # "caf.middlewares.ScrapeOpsFakeUserAgentMiddleware": 400,  
 'caf.middlewares.ScrapeOpsFakeBrowserHeaderAgentMiddleware': 400,  
 # 'rotating\_proxies.middlewares.RotatingProxyMiddleware': 610,  
 # 'rotating\_proxies.middlewares.BanDetectionMiddleware': 620,  
  
}

Définition des pipelines et de leurs priorités :

ITEM\_PIPELINES = {  
 # "caf.pipelines.CafPipeline": 300,  
 "caf.pipelines.SaveToPostgresPipeline": 400,  
}

## Lancer les spiders

* Pour stocker en fichier csv :
  + Commenter la ligne pour la sauvegarde en dans PostgreSQL
* ITEM\_PIPELINES = {  
   "caf.pipelines.CafPipeline": 300,  
   # "caf.pipelines.SaveToPostgresPipeline": 400,  
  }
  + scrapy crawl nom\_du\_spider > nomfichierdelog.txt 2>&1 -O nomdufichier.csv
* pour stocker en base de données
  + set les paramètres dans le fichier pipelines.py. exemple :
* # Connection Details  
  hostname = 'localhost'  
  username = 'postgres'  
  password = 'Azerty123$'  
  database = 'articles'
  + décommenter le ligne caf.pipelines.SaveToPostgresPipeline
  + et commenter la ligne caf.pipelines.CafPipeline
  + lancer le spider : scrapy crawl nom\_du\_spider > nomfichierdelog.txt 2>&1