Rapport projet data engineering

scrapping d’un site internet, stockage dans une base de données MongoDB et affichage dans une interface web

Léo Bruynooghe

Sommaire

I. Description du projet

A. Objectif du projet

L’objectif de ce projet est de récupérer les données présentes sur un site internet, de les stocker dans une base MongoDB et de les afficher sur une application web.

Le choix de la page web étant libre, nous avons choisie de scrapper les données du site ‘*https://myanimelist.net/manga.php*’. C’est un site stockant des informations sur des mangas. On peut y trouver le titre, le synopsis, la note, le genre et le nombre d’épisodes des mangas. L’objectif de notre scrapping sera donc d’extraire ces différentes données du site.

Une fois les données extraites, nous allons devoir les stocker sur une base MongoDB, c’est une base de données open source noSQL. Pour se faire, non utiliseront une pipeline.

Enfin, nous allons devoir afficher ces données sur une interface web, nous avons choisie la bibliothèque Dash afin de faciliter la création de cette interface. Le code est un fichier jupyter notebook, ce qui facile l’accès à la page web qu’il créer.

B. Guide d’utilisation

Dans ce guide, nous expliquerons comment lancer les différentes partie du projet.

Tout d’abord, ouvrez un terminal anaconda prompt et allez dans le répertoire du projet :

```
(base) c:\>cd CHEMIN_D_ACCES
```

Une fois dans le projet, allez dans le répertoire partie_scraping :

```
(base) c:\LEO\ESIEE\E4\Data_Engineering\projet>cd partie_scraping
```

Lancez la commande scrapy crawl manga :

```
(base) c:\LEO\ESIEE\E4\Data_Engineering\projet\partie_scraping>scrapy crawl manga
```

Cette commande permet de lancer le scrapping et de générer la base de donnée MongoDB. Une fois le code exécuté, retournez dans le répertoire projet grâce à la commande cd .. :

```
(base) c:\LEO\ESIEE\E4\Data_Engineering\projet\partie_scraping>cd ..
```

Puis lancez jupyter notebook avec la ligne de commande jupyter notebook :

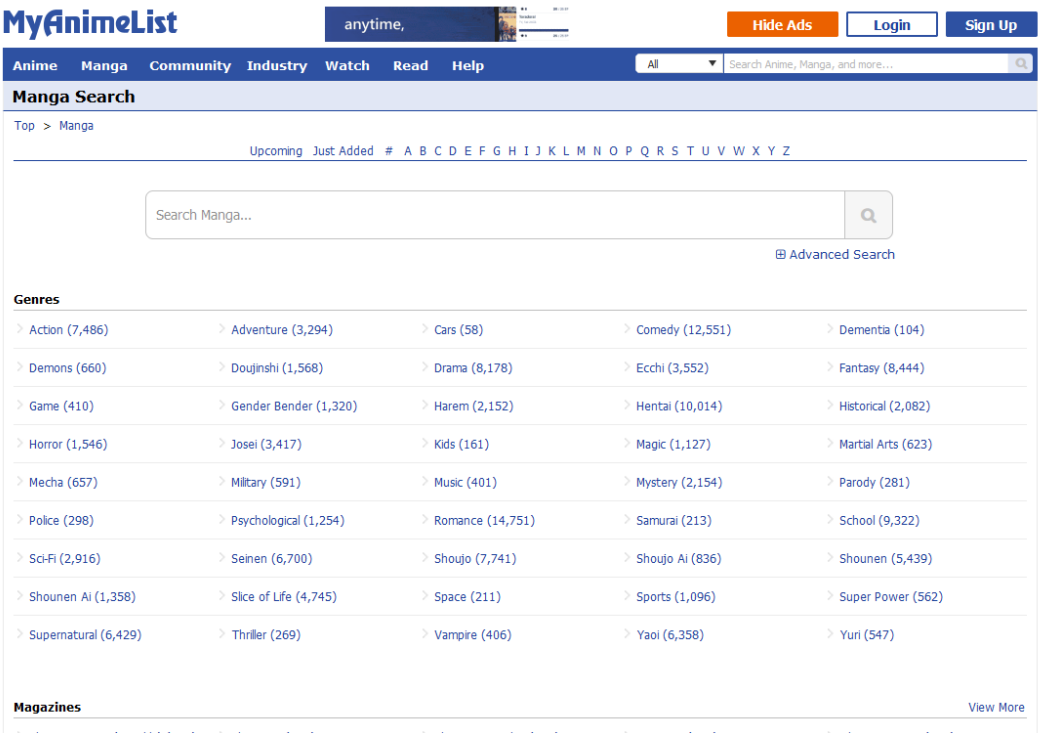
```
(base) c:\LEO\ESIEE\E4\Data_Engineering\projet>jupyter notebook
```

Une fois jupyter notebook lancé, ouvrez le fichier ‘web_app.ipynb’ et exécutez le.

Le message Dash is running on http://127.0.0.1:8050/ devrait s’afficher, cliqué sur le lien hypertext pour accéder à l’interface web.

II. partie scrapping

Dans cette partie, nous expliquerons comment nous avons fait pour scrapper les données du site ‘*https://myanimelist.net/manga.php*’. Intéressons nous dans un premier temps à l’infrastructure du site.



Sur la page d’accueil, nous remarquons que les mangas sont triés de plusieurs manières différentes, tout en haut on a un tri par ordre alphabétique, avec les liens vers les lettres de l’alphabet de A à Z. Ensuite on a un trie par genre, où les mangas sont triés en fonction de leur genre, si ils sont d’action, d’horreur, etc.
Et enfin, on a un trie par Magazines où les mangas sont trié en fonctions de leurs sujets.

Nous avons choisie de parcourir les liens des différentes lettres de l’alphabet, comme ça nous sommes sûr de ne pas scrapper un mangas deux fois. Par exemple, si un manga est à la fois catégorisé dans action et dans horreur et que l’on parcourt la liste des mangas d’action, puis la liste des mangas d’horreurs, on va le scrapper deux fois. Ce problème ne risque pas d’arriver avec le titre du manga, d’où ce choix.

Le premier objectif est donc de réussir à se rendre sur les différents liens correspondants aux lettres de l’alphabet, pour se faire nous allons regarder le code html de la page web à l’aide de la commande CRTL + U.

```
<div id="horiznav_nav" class="ac mt8 ml0 mr0">
  <ul>
    <li>
      <a href="https://myanimelist.net/manga.php?em=0&amp;ed=0&amp;ey=0&amp;c=2&amp;w=1&amp;c%5B0%5D=a&amp;c%5B1%5D=d&amp;cv=1&sd=24&amp;sm=4&amp;sy=2021">Upcoming</a>
    </li>
    <li>
      <a href="https://myanimelist.net/manga.php?c=9&amp;c%5B0%5D=a&amp;c%5B1%5D=d&amp;cv=2">Just Added</a>
    </li>
    <li>
      <a href="https://myanimelist.net/manga.php?letter=.">#</a>
    </li><li><a href="https://myanimelist.net/manga.php?letter=A">A</a></li><li><a href="https://myanimelist.net/manga.php?letter=B">B</a></li><li><a href="https://myan
</div>

<div>

  <form id="advancedsearch" data-type="manga" class="js-advancedsearch" method="GET" action="https://myanimelist.net/manga.php">
    <input type="hidden" name="cat" value="manga">

        <div class="anime-search-form-block po-r">
      <div class="anime-search-form-search clearfix mb8">
```

Dans le code html, on remarque que les lettres de l'alphabet sont contenue dans la Div d'id *'horiznav_nav'*, plus précisément, se sont des A contenue dans des Li elles-même contenue dans la Div d'id *'horiznav_nav'*, pour les extraire, on utilise donc la commande

```
urls = response.xpath("//div[@id='horiznav_nav']//li/a/@href").extract()
```

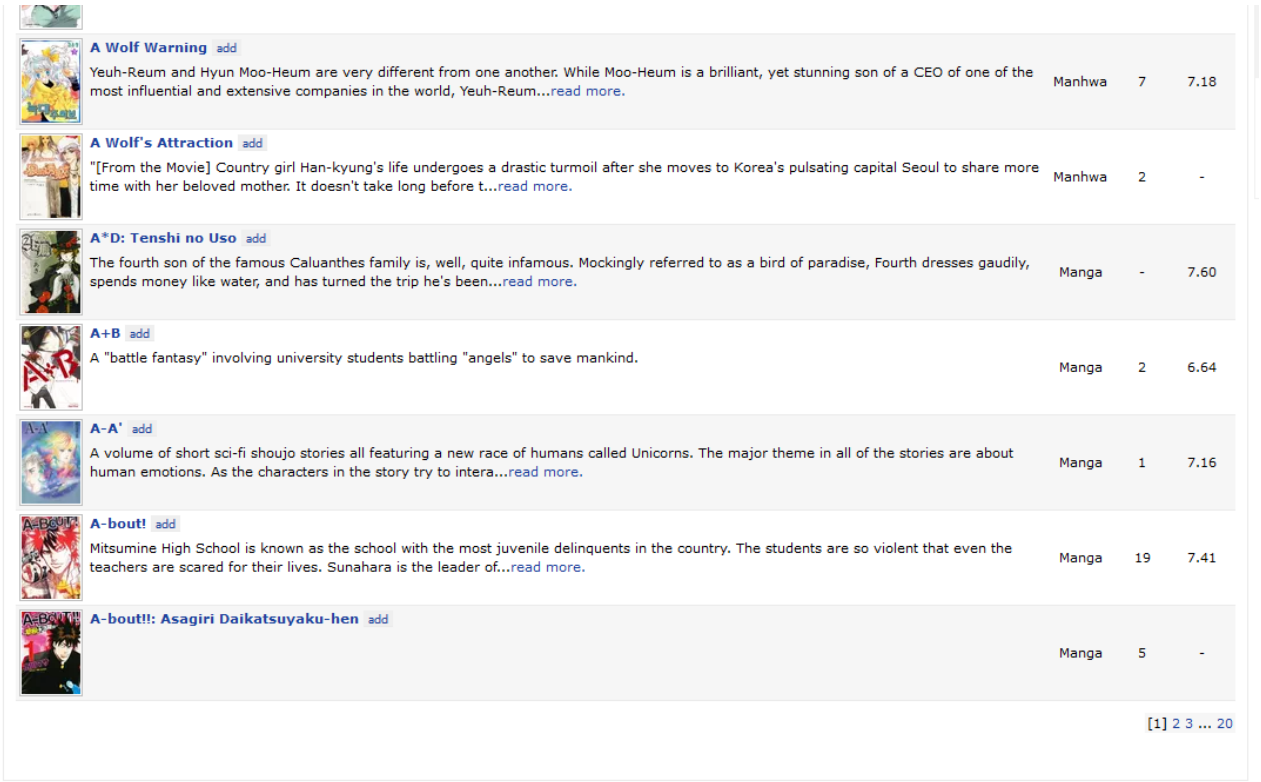
se faisant, nous extrayons les urls menant vers les pages correspondant aux différentes lettres.

Il ne reste plus qu'à parcourir les urls et à appeler une fonction sur chacune d'entre elles grâce à la commande :

```
for url in urls:
        yield Request(url, callback=self.manga_parse)
```

Notre scrapping parcourt maintenant toute les pages des différentes lettres, et pour chacune de ces pages, applique la fonction manga_parse.

Pour poursuivre, il nous faut regarder à quoi ressemble les pages web des différentes lettres.



On voit que la page web se présente sous la forme d'un tableau contenant le titre, le synopsis, le genre, la note et le nombre d'épisode du manga. En bas de la page, nous avons des liens pour consulter les autres mangas dont le titre commence par la même lettre. On observe maintenant le code html.

```
<tr>
  <td class="borderClass bgColor1" valign="top" width="50">
    <div class="picSurround">
      <a class="hoverinfo_trigger" href="https://myanimelist.net/manga/41153/A_Boy_Who_Fishes_Dreams" id="sarea41153" rel="#sinfo41153">
        <img width="50" height="70" alt="A Boy Who Fishes Dreams" border="0" data-src="https://cdn.myanimelist.net/r/50x70/images/manga/5/111613.web
      </a>
    </div>
  </td>
  <td class="borderClass bgColor1" valign="top"><div id="sarea41153">
      <div class="hoverinfo" id="sinfo41153" rel="m41153"></div>
    </div><a class="hoverinfo_trigger fw-b" href="https://myanimelist.net/manga/41153/A_Boy_Who_Fishes_Dreams" id="sinfo41153" rel="#sinfo41153"><st

    <a href="https://myanimelist.net/login.php?error=login_required&amp;from=%2Fmanga.php%3Fletter%3DA" title="Quick add manga to my list" class="bu

    <div class="pt4">This one shot has won 3rd spot of Daewon manwha contest, and thus the author has a high(er) chance of being employed by Issue m
  </td><td class="borderClass ac bgColor1" width="45">
    One-shot
  </td><td class="borderClass ac bgColor1" width="40">
    -
  </td><td class="borderClass ac bgColor1" width="50">
    6.61
  </td></tr>
```

On remarque que les informations que l'on cherche à extraire sont effectivement stocké dans un tableau. La partie affiché ci-dessus du code html correspond à une ligne de ce tableau. Une ligne est en effet encapsuler dans un composant html Tr. Chaque ligne contient les informations d'un manga.

Nous cherchons donc dans un premier temps à parcourir chaque composant Tr du tableau, pour cela nous utilisons la commande :

```
for manga in response.css('div.js-categories-seasonal tr ~ tr') :
```

Maintenant que nous parcourons chaque ligne de la table html, il ne nous reste plus qu'à en extraire les données que l'on souhaite. Pour le titre, il est dans le compartiment Strong du compartiment A de class '*hoverinfo_trigger fw-b*'. Pour l'extraire, il faut donc utiliser la commande :

```
titre = manga.xpath('//a[@class="hoverinfo_trigger fw-b"]/strong/text()').extract()
```

Pour le synopsis, il est dans la Div de class '*pt4*', pour l'extraire, il faut donc utiliser la commande :

```
synopsis = manga.xpath('//div[@class='pt4'/text()').extract()
```

Pour le genre, le nombre d'épisodes et la note, ils sont respectivement dans le troisième, quatrième et cinquième Td du compasant Tr. Nous pouvons donc les extraire grâce au commande :

```
genre = manga.css('td:nth-child(3/4/5)::text').extract_first()
```

Une fois que nous avons terminé l'extraction des données de la page, nous devons passer aux pages suivantes. Les liens des ces dernières se trouvent dans le A de la Div de class '*spaceit*', nous pouvons les extraire grâce à la commande :

```
urls = 'https://myanimelist.net' + response.xpath("//div[@class='spaceit']//a/@href").extract()
```

Malheureusement, et pour des raisons que nous ne somme pas parvenu à expliquer, l'extraction des données de la première page bloque le déplacement vers les autres pages. Nous devons donc arrêter notre scrapping à l'extraction de ces premières pages. Si cette extraction n'avait pas bloquer le déplacement vers les autres page, l'objectif aurait été de se déplacer vers la deuxième page du site ayant un url ressemblant à '*https://myanimelist.net/manga.php?letter=A&show=50*'. Une fois sur cette page, nous aurions extrait les données avec la même technique que pour la première page et nous aurions rappelé la fonction de manière récursive sur l'url '*https://myanimelist.net/manga.php?letter=A&show=100*', qui ne diffère que par le show=100 à la place du show=50. De cette manière nous aurions pu parcourir toute les données du site.

III. Item, pipeline et stockage dans la base MongoDB

Dans cette partie, nous allons expliquer les différents code python présent dans le répertoire partie_scraping et qui nous permettent de créer des items, de gérer la forme de nos données et de les stocker dans une base mongoDB.

A. Gestion des items

Dans le code items.py, on définie la class mangaItem. Les items fonctionnent comme des dictionnaires mais ne permettent que l'insertion d'objet dont les clés ont été préalablement déclarées. Cela permet un meilleur contrôle et favorise l'homogénéité d'un projet. Dans notre cas, on définie cinq champs pour notre mangaItem, le titre, le synopsis, le genre, le nombre d'épisodes et la note.

```
6   import scrapy
7
8
9   class mangaItem(scrapy.Item):
10      titre = scrapy.Field()
11      synopsis = scrapy.Field()
12      genre = scrapy.Field()
13      episodes = scrapy.Field()
14      note = scrapy.Field()
15
```

Maintenant, à la place de stocker les données scrappées dans des dictionnaires python classiques, on les stocke dans des mangaItems à la place.

B. Pipeline de mise en forme des données

Les données sont maintenant stocké dans des mangaItems mais leurs mise en forme reste la même que sur le site d'origine. Nous allons donc les faire passer dans une pipeline qui permettra leur mise en forme. Pour cela nous créons la class TextPipeline qui va nous permettre de modifier les items que nous avons créé. La class applique des str.strip() à toute les valeurs de mangaItem et test si les valeurs note et episodes contiennent bien des nombres, si ce n'est pas le cas, elle remplace les valeurs par des None, ce qui permettra d'économiser de la place dans la base MongoDB.

```
class TextPipeline:
    def process_item(self, item, spider):
        item["titre"] = item["titre"].strip()
        item["synopsis"] = item["synopsis"].strip()
        item["genre"] = item["genre"].strip()

        if "0" in item["episodes"] or "1" in item["episodes"] or "2" in item["episodes"] or "3" in item["episodes"]
            item["episodes"] = item["episodes"].strip()
        else :
            item["episodes"] = None

        if "0" in item["note"] or "1" in item["note"] or "2" in item["note"] or "3" in item["note"] or "4" in item['
            item["note"] = item["note"].strip()
        else :
            item["note"] = None
        return item
```

C. Pipeline permettant le stockage dans la base MongoDB

Une fois la pipeline TextPipeline est créé, il ne nous reste plus qu'à stocker les données dans une base MongoDB, pour cela, on va une nouvelle fois se servir d'une pipeline. Nous créons donc la class MongoDBPipeline qui va permettre ce stockage. Cette pipeline contient trois fonctions, une permettant l'ouverture de la base MongoDB, une permettant la fermeture une fois le stockage terminé et une permettant le stockage des données dans la collection mangaCollection de la base de données mangaDB.

```
28  import pymongo
29  from scrapy.exceptions import DropItem
30
31  class MongoDBPipeline(object):
32
33      collection_name = 'mangaCollection'
34
35      def open_spider(self, spider):
36          self.client = pymongo.MongoClient()
37          self.db = self.client["MangaDB"]
38
39      def close_spider(self, spider):
40          self.client.close()
41
42      def process_item(self, item, spider):
43          self.db[self.collection_name].insert_one(dict(item))
44          return item
```

Une fois ces deux pipelines créées, il ne nous reste plus qu'à modifier le code setting.py afin qu'elles soient prises en compte lors du scrapping. Nous ajoutons donc les lignes :

```
ITEM_PIPELINES = {
                        'partie_scraping.pipelines.TextPipeline' : 300,
                        'partie_scraping.pipelines.MongoDBPipeline' : 500
}
```

Ces lignes permettent la prise en compte des deux pipelines dans notre projet. Les numéros 300 et 500 donnent l'ordre de passage de nos pipelines. 300 étant inférieur à 500, la pipeline TextPipeline passera avant MongoDBPipeline, ce qui permettra de stocker des données déjà mise en forme dans notre base de données mangaDB.

IV. Affichage dans une interface web

Pour le développement de l'interface web permettant l'affichage des données, nous avons choisit la bibliothèque Dash. Elle permet d'encapsuler des composant html dans des lignes de code python de manière simple et intuitive.

A. Récupération des données de la base mangaDB

La première étape pour afficher les données sur une interface web consiste à les récupérer depuis la base MongoDB. Pour se faire, il suffit de créer un lien entre la base et le programme grâce à la commande :

```
client = MongoClient()
```

Une fois ce lien créé, il nous faut accéder à la bonne database Mongo, dans notre cas, nous voulons accéder à la database mangaDB, ce qui se fait via la commande :

```
db_Manga = client.MangaDB
```

Une fois dans la bonne database, nous choisissons la collection qui nous intéresse, dans notre cas il s'agira de la collection mangaCollection.

```
collection_Manga = db_Manga['mangaCollection']
```

Maintenant, collection_Manga contient les données que nous allons exploiter pour notre interface web. Pour accéder à ces données, il faut créer un curseur via la commande :

```
cursor = collection_Manga.find()
```

Ensuite, nous n'avons plus qu'à parcourir les documents de collection_Manga dans une boucle for.

```
for document in cursor :
```

B. Barre de recherche par titre

Dans la première partie du site web, nous cherchons à créer une barre de recherche permettant de trouver un manga grâce à son titre. Pour se faire, nous allons créer un composant input dans notre code via la commande :

```
dcc.Input(id='barre_recherche', value='enter manga title', type='text')
```

Ce composant est une barre de recherche permettant d'entrer du texte. La valeur du composant correspond au texte rentré. Nous définissons ensuite un callback qui prend en entrée cette valeur et en sortie une table que nous créons avec la commande :

```
html.Table(id = "specific_manga",children=[…])
```

Ce callback va comparer la valeur avec le début des titres de chaque manga, si le titre du manga a la même valeur que ce qui est passé dans la barre de recherche, les informations du manga sont affichés dans la table html.

C. Sélection d'un genre de manga

Pour la seconde partie du site web, nous cherchons à fournir un moyen de sélectionner tout les mangas d'un même genre. Pour y arriver, nous créons le dictionnaire liste_genre qui contient tout les genres des différents mangas accessible via deux clés, value et label. Chacune des clés contient la même valeur, un genre de manga. Il n'y à qu'une exception à cette règle, où value contient *'all'* et label *'Tout les genres'*.

Une fois le dictionnaire créé, nous ajoutons à notre site web un dropdown grâce à la commande :

```
dcc.Dropdown(id = "genre_selection",options=liste_genre, value='all')
```

Ce dropdown est un composant permettant la sélection des différents genre de manga présent dans liste_genre. Une fois ce composant créé, nous définissons un callback qui prend en entrée le genre selectionné et en sortie une table html. Ce callback va comparer le genre du dropdown avec le genre de chaque manga, si il ont la même valeur, les informations du manga seront affichés dans la table html.

V. Conclusion

Notre projet est constitué de deux partie totalement indépendantes. La partie scrapping et stockage des données dans une base MongoDB, qui a pour but de stocker des données présente sur un site web et de les mettre en forme. Et la partie affichage sur une interface web, qui s’occupe de la gestion et du trie des données ainsi que de la création de l’interface web permettant de les présenter.

Le projet est fonctionnel mais il serait toutefois possible de l'améliorer. En effet le scrapping ne récupère pas l’intégralité des mangas du site et il serait possible de l’améliorer afin que se soit le cas. De plus l’interface web pourrait bénéficier de plus de fonctionnalité, pour classer les mangas par note par exemple.