Lamy-Verdin Tiphanie Lim Annie Vichaikit Jean-Philippe

Documentation Technique

DSIA-4201C Data Engineering

Courivaud R. 2019-2020

Sommaire

Présentation

Le projet a pour but d'afficher de manière actualisée le classement musical des meilleurs 200 albums de la semaine. Les données sont directement explorées et récupérées du site officiel Billboard, pour ensuite y traiter les informations pertinentes pour les réafficher à l'aide de graphiques. Il est également possible d'effectuer des recherches sur les albums ou artistes présents parmi ce top 200, s'afficheront alors plus de détails tels que leur positionnement de la semaine précédente, leur plus haut classement et durée à cette position.

Docker

Afin de lancer notre application dans des conteneurs logiciels, nous avons utilisé Docker qui nous a notamment été utile pour monter une instance MongoDB qui sera accessible depuis internet. De même nous avons utilisé Docker pour conteneuriser ElasticSearch pour stocker et effectuer des requêtes sur nos données.

Un fichier docker-compose permet de générer nos images MongoDB et ElasticSearch.

Scrapy

Nous utilisons le framework Scrapy qui permet l'exploration de sites web et y extrait des données de façon structurée. Ce framework de Spider peut gérer le Crawling et le Scraping de site web très efficacement.

Spider

Notre spider *billboard.py*récupère directement les données de classement musical de la semaine depuis le site Billboard.

Il est possible de lancer la spider à l'aide de la commande :

scrapy crawl billboard

La méthode *parse()* retourne un dictionnaire correspondant à chaque album présent dans le classement, contenant leurs informations respectifs : nom de l'album, artiste, rang, plus haut rang atteint, durée à ce haut rang, position de la semaine précédente.

```
def parse(self, response):
    for i in range(0,200):
        album = response.css("span.chart-element__information")[i].css("span.chart-element__information__song.text--truncate.color--primary::text").extract_first()
        artist = response.css("span.chart-element__information")[i].css("span.chart-element__information__artist.text--truncate.color--secondary::text").extract_first()
        rank = response.css("span.chart-element__rank.flex--column.flex--xy-center.flex--no-shrink")[i].css("span.chart-element__rank__number::text").extract_first()
        peak = response.css(".chart-element__metas.display--flex.flex--y-center")[i].css(".chart-element__meta.text--center.color--secondary.text--peak::text").extract_first()
        duration = response.css(".chart-element__metas.display--flex.flex--y-center")[i].css(".chart-element__meta.text--center.color--secondary.text--week::text").extract_first()
        last_week = response.css(".chart-element__metas.display--flex.flex--y-center")[i].css(".chart-element__meta.text--center.color--secondary.text--last::text").extract_first()
        yield items.ArticleItem(
          album=album,
          artist=artist,
          rank=rank,
          peak=peak,
          duration=duration,
          last_week=last_week
        )
```

Item

Nous structurons nos données en les récupérant sous la forme d'un modèle, défini dans la classe item. Chaque item, correspondant à chaque album du classement, défini dans le fichier *items.py* possède : nom de l'album, artiste, rang, plus haut rang atteint, durée à ce haut rang, position de la semaine précédente.

```
class ArticleItem(scrapy.Item):
    album = scrapy.Field()
    artist = scrapy.Field()
    rank = scrapy.Field()
    peak = scrapy.Field()
    duration = scrapy.Field()
    last_week = scrapy.Field()
    image = scrapy.Field()
```

MongoDB

MongoDB est une base de données noSQL avec une structure très flexible et optimisée, adaptée au contexte de notre projet. Elle stocke les données sous la forme de document JSON et a pour avantage l'optimisation de la mémoire.

```
4   class MongoPipeline(object):
5
6       collection_name = 'scrapy_items'
7
8
9       def open_spider(self, spider):
10          self.client = pymongo.MongoClient() #par défaut, paramétré sur localhost port 27017
11          self.db = self.client["billboard"]
12
13      def close_spider(self, spider):
14          self.client.close()
15
16      def process_item(self, item, spider):
17          self.db[self.collection_name].insert_one(dict(item))
18          return item
19
```

Nous pouvons alors créer notre base de données « billboard » automatiquement lors du scraping de la page.

Il nous sera également possible de récupérer cette base de données directement à partir de notre framework Flask.

Flask

Flask est un framework open-source de développement web nous permettant un affichage des données simple.

```python
@app.route('/MusicSearch', methods=('GET', 'POST'))
def MusicSearch():
    """
    Création de la barre de recherche ainsi que de l'affichage des données, et affichage du bargraph
    """
    form = SearchBar()
    if form.validate_on_submit():
        return redirect('/information/'+form.typing.data)
    div, script = create_bargraph()

    ranks, albums, artists = create_table()

    return render_template('music_search.html', form=form,
        the_div=div, the_script=script, ranks=ranks, albums=albums, artists=artists)
```

Le fichier *app.*py est utilisé pour l'affichage web. La fonction *MusicSearch* s'occupe de la génération d'une barre de recherche, d'un tableau ainsi que d'un bargraph, qui est envoyé au fichier *music_search.html* par le biais des variables dans la fonction *render_template()*.

```
<h2>Artists with the most albums in the top 200</h2>

{{ the_div|safe }}
<script src="http://cdn.pydata.org/bokeh/release/bokeh-1.4.0.min.js"></script>
<script src="http://cdn.pydata.org/bokeh/release/bokeh-widgets-1.4.0.min.js"></script>
{{ the_script|safe }}
```

Dans le fichier *music_search.html*, on récupère les données du backend vers le frontend grace à la syntaxe*{{ nom_variable }}*. Il s'agit ici du bargraph que l'on récupère.

Nous allons ainsi afficher le classement actuel de la semaine des 20 premiers albums du billboard 200 avec le nom de l'album, nom de l'artiste et son rang.

De même il y aura à la suite un bargraph correspondant aux artistes ayant le plus d'albums présents dans le top 200 de la semaine.

En haut de la page se trouvera une barre de recherche nous permettant d'affiner notre recherche selon le nom de l'album ou bien le nom d'un artiste.

ElasticSearch

ElasticSearch est un moteur de recherche permettant une recherche très efficace de données textuelles, numériques et géolocalisées.

```python

@app.route('/information/<search_word>')
def sucess(search_word):
    """
    Affichage des résultats de la recherche en fonction des artistes et albums.
    """
    df_billboard = pd.DataFrame(list(billboard_200.find()))
    df_billboard_clean = df_billboard.drop(labels='_id',axis='columns')
    documents = df_billboard_clean.fillna("").to_dict(orient="records")

    bulk(es_client, generate_data(documents))

    QUERY = {
      "query": {
        "multi_match" : {
          "query":    search_word,
          "fields": [ "artist", "album" ]
        }
      }
    }
    #result=["bonjour","merci"]
    result = es_client.search(index="albms", body=QUERY)
    source = result["hits"]["hits"]

    seen = set()
    new_source = []
    for d in source:
        t = tuple(d["_source"].items())
        if t not in seen:
            seen.add(t)
            new_source.append(d)

    album = [elt['_source']['album'] for elt in new_source]
    artist = [elt['_source']['artist'] for elt in new_source]
    rank = [elt['_source']['rank'] for elt in new_source]
    peak = [elt['_source']['peak'] for elt in new_source]
    duration = [elt['_source']['duration'] for elt in new_source]
    last_week = [elt['_source']['last_week'] for elt in new_source]

    return render_template('results.html',albums=album,artists=artist,ranks=rank, peak=peak, duration=duration, last_week=last_week)
```

Lorsque la recherche aboutit, les résultats sont stockés dans la variable « source ». On traite ensuite les données pour supprimer les duplications, et on affiche le tout dans la fonction render_template().

Nous affichons alors tous les albums contenant un terme correspondant à notre recherche, de même nous affichons tous les albums d'un artiste si nous faisions la recherche sur un artiste. Sur chaque ligne du tableau sera affiché le nom de l'album, son artiste, son rang actuel et son rang de la semaine précédente, ainsi que son plus haut rang et la durée à cette position.

La recherche ne prend pas en compte les majuscules mais le terme doit être correctement orthographié.