Documentation technique

Data Engineering

Antoine MOREAU

Description

Aujourd'hui, de nombreux sites existent pour répertorier le données météorologique des différents spots de surf. Or étant moi même surfeur je me suis rendu compte que cela prenait beaucoup de temps d'analyser les nombreuses données pour chaque spots et chaque jours lorsque je recherche un endroit pour aller surfer.

Le but de cette application est de récolter les données des différents spots de surf en France sur le site "https://www.surf-report.com/" à l'aide **scrappy**, une fois c'est données stockées l'affichage se fera une application web en utilisant le framework **Flask**.

Mongodb a été utilisé pour créer la base de données et permettre les requêtes.

Afin d'analyser via des graphiques les conditions météorologiques moyennes des différents spots **Dash** a été utilisé .

Enfin **Docker** a permis la virtualisation d'un serveur pour la base de donnée mongo

Scarping

Afin de récupérer les données du site de prévisions météorologique de surf j'ai utilisé Scrapy pour parcourir et récupérer les informations voulue.

Spyder

La spider « Spot2.py » permet grace à l'utilisation de scrapy la récupération des différentes caractéristiques des spots de surf tel que la taille des vagues, la températures, les liens, la vitesse du vents, le jour souhaité…

Pour l'extraction de données j'ai utilisé le css (cf fig a.1) car plus facile a prendre en mains que HtmlxPathSelector.

```
response.css(".grid_1 .card-content .title").css('b::text').extract(),
response.css(".grid_1 .card-content .title").css('a::attr(href)').extract()
```

Fig a.1 : Extraction de données via css

La spider s'execute (cf fig a.2) dans un premier temps sur les deux pages principales du site (cf fig a.3) qui présentent tous les spots de surf de France sur ces deux pages on récupère tout les liens des pages contenant les caractéristiques de chaque spot (cf fig a.4). Une fois ces liens récupérés on va appliquer une fonction d'extraction des caractéristiques nommée « météo_parse » a chacune de ces pages, pour cela j'ai utilisé un callback de la fonction d'extraction qui s'appliquera à chacun des liens (cf fig a.5).

```
class ExampleSpider(scrapy.Spider):
    name = 'Spot2'
    allowed_domains = ['www.surf-report.com']
    start_urls = ['https://www.surf-report.com/meteo-surf/trance/',
                  'https://www.surf-report.com/meteo-surf/france/?pageId=2'
                 ]
```

Fig a.2 : Page de commencement lors de l'execution du spider

Fig a.4 : Pages de commencement

```
def parse(self, response):
    all_links = {
        name:response.urljoin(url) for name,url in zip(
        response.css(".grid_1 .card-content .title").css('a::text').extract(),
        response.css(".grid_1 .card-content .title").css('a::attr(href)').extract()
        )
    }
   # x=0
    for link in all_links.values():
        #x=x-1
       # if x<10:
        yield Request(link, callback=self.meteo_parse)
            #print(link)
```

Fig a.5 : Pages de commencement

Pipelines

Les pipelines ont pour but le traitement de la donnée scrapée ce traitement ce fait au moment du scraping.

J'ai créer un pipeline pour le stockage de la donnée dans MongoDB (fig b.1), puis pour le nettoyage de chaines de caractères (fig b.2).

```python
class MongoPipeline(object):

    collection_name = 'scrapy_items'

    def open_spider(self, spider):
        self.client = pymongo.MongoClient("0.0.0.0:27018")
        self.db = self.client.spotSurf
        self.scrapy_items_col = self.db[self.collection_name]

    def close_spider(self, spider):
        self.client.close()

    def process_item(self, item, spider):
        self.db[self.collection_name].insert_one(dict(item))
        return item
```

Fig b.1 : Pipeline MongoDB

```python
class TextPipeline(object):
    def clean_spaces(string):
        if string:
            return " ".join(string.split())
```

Fig b.2 : Pipeline clean_spaces

Setting

Le fichier setting permet notamment la déclaration des pipelines et permet de choisir l'ordre dans lequel on les exécute (fig c.1).
Jai également choisi de générer un fichier CSV contenant les données récupérée lors du scraping afin d'avoir un aperçu de celle ci lors de mon travail (fig c.2).

```
ITEM_PIPELINES = {
    'ScrapSurfSpot.ScrapSurfSpot.pipelines.TextPipeline': 100,
    'ScrapSurfSpot.ScrapSurfSpot.pipelines.MongoPipeline': 300,
}
```

Fig c.1 : Déclaration des pipelines

```
#Exporter en tant que flux CSV
FEED_FORMAT = "csv"
FEED_URI = "spots.csv"
```

Fig c.2 : Génération du fichier CSV

Exécution du scraping

Afin de lancer le scraping de façon automatique depuis l'application flask j'ai crée la fonction run_spider (fig d.1) qui une fois appelé dans l'application flask permettra le lancement de la collecte de donnée (fig d.2).

Dans le but que les données correspondent aux données en temps réel, on vide et recréer avec les donnée fraichement scrapée la base de donnée mongoDB à chaque lancement de l'application, idem pour le fichier CSV (fig d.3).

```
class Scraper:
    def __init__(self):
        settings_file_path = 'ScrapSurfSpot.ScrapSurfSpot.settings'
        os.environ.setdefault('SCRAPY_SETTINGS_MODULE', settings_file_path)
        self.settings = get_project_settings()
        self.process = CrawlerProcess(self.settings)
        self.spider = Spot2.ExampleSpider

    def run_spider(self):
        self.process.crawl(self.spider)
        self.process.start()
```

Fig d.1 : Fonction run_spider

```
app = Flask(__name__)

scraper = Scraper()
scraper.run_spider()
```

Fig d.2 : Appel de la fonction

```
client = MongoClient("0.0.0.0:27018")
database_spot = client.spotSurf
database_spot['scrapy_items'].drop()
collection = database_spot['scrapy_items']

#Supression du fichier csv
if os.path.exists('spots.csv'):
    os.remove('spots.csv')
```

Fig d.3 : Renouvellement des données

MongoDB

Afin de stocker les données scrapées j'ai utilisé la base donnée no SQL MongoDB qui a pour grand avantage l'optimisation de la mémoire. Dans une base relationnelle, chaque colonne doit être définie au préalable avec une empreinte mémoire et un type de donnée. Dans une base MongoDB si le champ n'est pas présent, il n'apparait pas dans un document et n'impacte pas la mémoire.

Au lancement de l'application flask on créer une collection (fig e.1) qui va être rendu au document html (fig e.2) pour permettre d'afficher les données (fig e.3).

```python
@app.route('/')
def accueil():
    documents = collection.find({'swell_day_0':  {'$gt':0}})
    response = []

    for document in documents:
        response.append(document)
```

Fig e.1 : Création de la collection

```python
    return render_template('index.html', spot = response)
```

Fig e.2 : Passage des données au templates

```
{% for i in spot2 %}
<div> <li> Aujourd'hui le spot <b><a href="{{i['url_clean_spot']}}">{{i['name_spot']}}
  une température d'air de <b>{{i['temp_air_day_0']}}°C</b> et la vitesse du vent est d
</div>
{% endfor %} </li>
```

Fig e.3 : Utilisation des données dans les templates

Dashboard

Afin d'avoir un aperçu global des conditions des spots en France j'ai utilisé un dashboard pour créer des histogrammes pour plusieurs données significative.

Jai créé une application dash que j'ai implantée dans mon site flask.

```
dash_app_0.layout = html.Div(children=[
    html.H1(children='Aperçu des conditions sur les différents spots en France '),

    html.Div(children="""Afin d'affiner vos recherches dans le choix de votre spot voici
    un aperçu des conditions sur les différents spots en France.
    """),
```

Fig f.1 : Création de l'application

```
dash_app_0 = dash.Dash(__name__, server=app, routes_pathname_prefix = '/dash_0/')
dash_app_1 = dash.Dash(__name__, server=app, routes_pathname_prefix = '/dash_1/')
dash_app_2 = dash.Dash(__name__, server=app, routes_pathname_prefix = '/dash_2/')
dash_app_3 = dash.Dash(__name__, server=app, routes_pathname_prefix = '/dash_3/')
dash_app_4 = dash.Dash(__name__, server=app, routes_pathname_prefix = '/dash_4/')
```

Fig f.2 : Génération de l'application

Docker Compose

Afin de créer un serveur virtuel pour la base de donnée MongoDB j'ai utilisé Docker Compose (fig g.1).

```yaml
mongo:
  image: mongo
  container_name: mongo
  environment:
    - MONGO_DATA_DIR=/data/db
    - MONGO_LOG_DIR=/dev/null
  volumes:
    - ./data/mongo:/data/db
  ports:
      - "27018:27017"
```

Fig g.1 : Configuration fichier .yml