Section 2 : Web Scraping et traitement de données

Ayoub Abraich

1. Extraction de données

Pour extraire les données des pages web mentionnées, j'ai créé un script Python appelé `web_scraper.py`. Ce script utilise les bibliothèques `requests` pour récupérer le contenu HTML des pages, `BeautifulSoup` pour parser le HTML et `urllib.parse` pour manipuler les URLs.

La classe `WebScraper` a été implémentée avec les méthodes suivantes :

- `__init__` : initialise la classe avec une liste d'URLs à scraper et un dictionnaire vide pour stocker les données.
- `_get_base_url` : extrait l'URL de base à partir d'une URL donnée.
- `_extract_documents` : extrait les liens vers des documents PDF de la page HTML.
- `_extract_paragraphs` : extrait les paragraphes contenant le mot "taux" de la page HTML.
- `_extract_conditions` : filtre les liens de documents contenant le mot "condition".
- `extract_content` : méthode principale qui récupère le contenu HTML d'une URL, appelle les méthodes d'extraction et stocke les données dans le dictionnaire.
- `get_data_all_urls` : boucle sur la liste d'URLs pour extraire les données de toutes les pages.
- `get_data` : extrait les données d'une seule URL donnée.

Voici un exemple d'utilisation :

```
from web_scraper import WebScraper

list_of_urls = [
    "https://www.creditmutuel.fr/fr/particuliers/epargne/livret-de-developpement-durable.html",
    "https://www.monabanq.com/fr/produits-bancaires/livret-developpement-durable/en-resume.html",
    "https://www.banquepopulaire.fr/bpaura/epargner/livret-transition-energetique/"
]

scraper = WebScraper(list_of_urls)
output_dict = scraper.get_data_all_urls()
print(output_dict)
```

2. Création d'APIs/Web Services

J'ai créé une API RESTful en utilisant le framework FastAPI. Le fichier `main.py` contient le code de l'API

3. Base de données NoSQL

J'ai choisi d'utiliser RavenDB comme base de données NoSQL pour stocker les données extraites. RavenDB est une base de données orientée documents qui convient bien pour stocker des données semi-structurées comme celles issues du web scraping.

Voici un exemple de schéma pour stocker les données extraites dans RavenDB :

```
{
    "id": "documents/1",
    "bank_name": "Credit Mutuel",
    "url": "https://www.creditmutuel.fr/fr/particuliers/epargne/livret-de-developpement-durable.html",
    "documents": [
        "https://www.creditmutuel.fr/documents/pdf/livret-developpement-durable.pdf",
        "https://www.creditmutuel.fr/documents/pdf/conditions-generales.pdf"
    ],
    "taux": [
        "Le taux d'intérêt annuel brut est de 0,75% au 01/01/2023.",
        "Le taux d'intérêt est révisable chaque année par le Crédit Mutuel."
    ],
    "conditions": [
        "https://www.creditmutuel.fr/documents/pdf/conditions-generales.pdf"
    ]
}
```

Ce schéma représente un document RavenDB contenant les informations suivantes :

- `id` : identifiant unique du document
- `bank_name` : nom de la banque
- `url` : URL de la page web scrapée
- `documents` : liste des liens vers les documents PDF
- `taux` : liste des paragraphes contenant le mot "taux"
- `conditions` : liste des liens vers les documents contenant le mot "condition"

Pour importer les données dans RavenDB, j'ai créé une classe `RavenDB` dans `main.py` qui initialise une connexion avec la base de données et fournit des méthodes pour ouvrir une session et stocker des données.

L'endpoint `/scrape/` de l'API utilise cette classe pour stocker les données extraites dans la base de données RavenDB.

4. Git et Collaboration

J'ai créé un dépôt GitHub pour partager le code de cette solution : https://github.com/zetaneh/projet_test/

Le dépôt contient les fichiers suivants :

```python
from pydantic import BaseModel
from ravendb import DocumentStore
import uvicorn
from web_scraper import *

app = FastAPI()

list_of_urls = [
    "https://www.creditmutuel.fr/fr/particuliers/epargne/livret-de-developpement-durable.html",
    "https://www.monabanq.com/fr/produits-bancaires/livret-developpement-durable/en-resume.html",
    "https://www.banquepopulaire.fr/bpaura/epargner/livret-transition-energetique/"
]
# Define the RavenDB class
class RavenDB:
    def __init__(self, urls, database):
        self.store = DocumentStore(urls=urls, database=database)
        self.store.initialize()

    def open_session(self):
        return self.store.open_session()

    def close(self):
        self.store.dispose()

# Define the model for incoming data
class BankData(BaseModel):
    bank_name: str
    data: str


# Initialize RavenDB
urls = ["http://localhost:8080"]
database = "BankData"
raven = RavenDB(urls, database)



# root endpoint
@app.get("/")
async def root():
    return {"message": "Welcome to the Bank Data API"}

# FastAPI endpoint to scrape data from URLs and store in RavenDB
@app.post("/scrape/")
async def scrape_and_store_data(data: BankData):
    with raven.open_session() as session:
        session.store(data.dict(), collection_name="BankData")
        session.save_changes()
        return {"message": "Data scraped and stored successfully"}
```

```
    scraper = WebScraper(list_of_urls)
    output_dict = scraper.get_data(url)
    return output_dict



# Run the FastAPI server
if __name__ == "__main__":
    scraper = WebScraper(list_of_urls)
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

- `web_scraper.py` :

```
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin, urlparse

class WebScraper:
    def __init__(self, list_of_urls):
        self.list_of_urls = list_of_urls
        self.data = {}

    def _get_base_url(self, url):
        parsed_url = urlparse(url)
        return parsed_url.scheme + "://" + parsed_url.netloc

    def _extract_documents(self, soup, base_url):
        links = soup.find_all("a", href=True)
        documents = [urljoin(base_url, l.get("href")) for l in links if 'pdf' in l.get("href")]
        return documents

    def _extract_paragraphs(self, soup):
        paragraphs = soup.find_all("p")
        taux_paragraphs = [p.text.strip() for p in paragraphs if 'taux' in p.text.strip().lower()]
        return taux_paragraphs

    def _extract_conditions(self, documents):
        return [doc for doc in documents if 'condition' in doc]

    def extract_content(self, url):
        try:
            response = requests.get(url)
            if response.status_code == 200:
                soup = BeautifulSoup(response.content, "html.parser")
                base_url = self._get_base_url(url)
                documents = self._extract_documents(soup, base_url)
```

```
        return self.data[url]

if __name__ == "__main__":
    list_of_urls = [
        "https://www.creditmutuel.fr/fr/particuliers/epargne/livret-de-developpement-durable.html",
        "https://www.monabanq.com/fr/produits-bancaires/livret-developpement-durable/en-resume.html",
        "https://www.banquepopulaire.fr/bpaura/epargner/livret-transition-energetique/"
    ]

    scraper = WebScraper(list_of_urls)
    output_dict = scraper.get_data_all_urls()
    print(output_dict)
```