Scraping GitHub

Réalisé par :

M^r SABRI Aymane

Encadre Par :

M^r Tajmaouti Moad

Resume

Le présent projet visait à obtenir des informations précieuses sur les dépôts de code hébergés sur GitHub afin de mieux comprendre les tendances de développement, les langages de programmation les plus utilisés, d'identifier des projets intéressants, de suivre les évolutions technologiques et d'explorer de potentielles opportunités de collaboration .

La première étape de ce projet consistait à bien comprendre le contexte et les attentes du projet . Cela m'a permis de définir clairement les objectifs à atteindre.

Ensuite, j'ai procédé à la définition précise des objectifs spécifiques du projet. J'ai identifié les indicateurs clés que je souhaitais collecter, tels que les langages de programmation populaires, les projets les plus actifs et les tendances émergentes.

Pour la recherche des outils à utiliser, j'ai exploré différentes options en tenant compte de la scalabilité, de la qualité des données et des contraintes techniques spécifiques à ma situation.

Une fois les outils sélectionnés, j'ai commencé la collecte des données en naviguant et en extrayant les informations pertinentes des dépôts GitHub ciblés.

Afin de garantir la qualité des données collectées, j'ai effectué des tests approfondis. J'ai vérifié la fiabilité des résultats en comparant les informations extraites avec celles disponibles directement sur GitHub.

Enfin, j'ai procédé au nettoyage des données collectées et les ai stockées dans un format approprié pour une utilisation ultérieure. Ce processus m'a permis de maintenir l'intégrité et la pertinence des informations obtenues.

1. Context du projet :

1.1. Definition des objectifs :

Dans cette première étape, j'ai pris le temps de comprendre le contexte et les objectifs du projet concernant la collecte de données sur GitHub. J'ai eu des discussions approfondies avec les membres de groupe et j'ai analysé les attentes et les besoins spécifiques

J'ai identifié les points clés suivants :

1. Analyser les objectifs et les indicateurs clés que nous souhaitons obtenir à partir des données collectées.
2. Prendre en compte les contraintes de temps, de ressources et de faisabilité technique pour ce projet.
3. Établir une vision claire de la manière dont les informations collectées sur GitHub seront utilisées pour prendre des décisions éclairées et identifier de nouvelles opportunités.

1.2. Indicateurs clés :

Dans cette étape, j'ai précisément défini les objectifs spécifiques de notre projet de collecte de données GitHub. Mon principal objectif était de récupérer des informations clés sur les dépôts de code, j'ai identifié les indicateurs suivants :

1. Données des dépôts : J'ai ciblé les informations telles que le nom du dépôt, le nombre d'étoiles le nombre de contributeurs et les langages de programmation utilisées. Ces données nous donnent une vision globale des dépôts pertinents sur GitHub.
2. Identification de projets intéressants : En collectant des informations sur les dépôts les plus populaires et les mieux notés, nous sommes en mesure d'identifier des projets intéressants et innovants. Ces projets peuvent représenter des opportunités pour notre recherche, que ce soit pour la collaboration, la recherche d'inspiration ou le suivi des meilleures pratiques .
3. Suivi des évolutions technologiques : En analysant les langages de programmation émergents et en suivant les tendances de développement, nous sommes en mesure de rester à jour avec les nouvelles technologies et les évolutions de notre domaine.
4. Exploration d'opportunités de collaboration : Les informations collectées sur les dépôts nous permettent d'explorer des opportunités de collaboration avec d'autres développeurs ou organisations. Nous pouvons identifier des projets complémentaires ou des partenariats potentiels qui pourraient renforcer notre travail et stimuler l'innovation.

En définissant ces objectifs spécifiques, j'ai pu orienter ma collecte de données vers les informations pertinentes pour notre projet La prochaine étape consistera à rechercher les outils adéquats pour réaliser efficacement ce projet

2. Etape De Réalisation :

2.1. Github API :

Lors de la réalisation de ce projet, j'ai utilisé l'API GitHub pour effectuer le scraping des données nécessaires à partir des dépôts de code. L'API GitHub fournit un accès structuré aux informations des dépôts, ce qui facilite la collecte des données telles que le nom du dépôt, le nombre d'étoiles, le nombre de contributeurs, le nombre de forks et le langage de programmation utilisé .

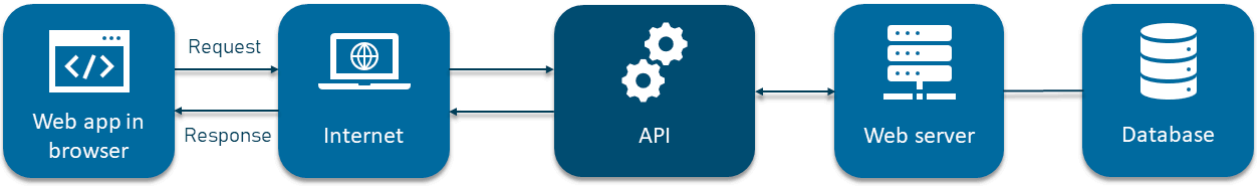
2.2. Défis Rencontrés :

Cependant, j'ai rencontré quelques défis lors de l'utilisation de l'API GitHub. Tout d'abord, il y avait des limitations de taux (rate limits) imposées par l'API, ce qui signifie que je devais faire attention à ne pas dépasser le nombre maximal de requêtes autorisées par unité de temps.

Pour résoudre ce problème, j'ai mis en place une stratégie de gestion des requêtes en respectant les limites imposées et en utilisant les options de pagination fournies par l'API.

Un autre défi auquel j'ai été confronté était la gestion des données volumineuses. Les dépôts GitHub peuvent contenir un grand nombre de fichiers et d'informations, ce qui peut ralentir le processus de collecte de données. Pour optimiser les performances, j'ai utilisé des techniques telles que la pagination, la mise en cache des données collectées et l'optimisation des requêtes pour minimiser le temps de réponse .

En surmontant ces défis, j'ai pu obtenir les données nécessaires à partir de l'API GitHub de manière efficace et précise. Cela m'a permis d'avancer vers les étapes suivantes du projet, telles que le nettoyage et le stockage des données collectées .



2.3. Etape De Scraping :

- Importation des libraries nécessaires

```
import requests
import csv
import time
from datetime import datetime, timedelta
```

- Reception des repo cree dans une date spécifique

```
def get_repositories_within_date_range(start_date, end_date, token):
    headers = {"Authorization": f"Bearer {token}"}
    base_url = "https://api.github.com/search/repositories"
    per_page = 10
    repositories = []

    # Divide the date range into smaller intervals
    date_ranges = divide_date_range(start_date, end_date)

    for date_range in date_ranges:
        page = 1
        total_pages = 1

        while page <= total_pages:
            params = {
                "q": f"created:{date_range[0]}..{date_range[1]}",
                "sort": "stars",
                "order": "desc",
                "page": page,
                "per_page": per_page
            }

            response = requests.get(base_url, headers=headers, params=params)

            if response.status_code == 403:
                print("Rate limit exceeded. Waiting for reset...")
                reset_time = int(response.headers["X-RateLimit-Reset"])
                wait_time = reset_time - time.time()
                time.sleep(wait_time)
                continue

            if response.status_code != 200:
                print(f"Error: {response.status_code}")
                print(f"Response Content: {response.content}")
                break

            result = response.json()
            repositories.extend(result["items"])

            total_pages = (result["total_count"] + per_page - 1) // per_page
            page += 1
            print(f"Processing page {page}/{total_pages}")

    return repositories
```

- La fonction `get_repositories_within_date_range` récupère les dépôts dans une plage de dates spécifiée à partir de l'API GitHub. Elle prend trois paramètres : `start_date` , `end_date` et `token`. Elle divise la plage de dates en intervalles plus petits et effectue des requêtes vers l'API GitHub pour chaque intervalle. Les dépôts sont stockés dans une liste et retournés à la fin.

```
def divide_date_range(start_date, end_date):
    interval = timedelta(days=30)  # Number of days per interval
    date_ranges = []
    current_date = start_date

    while current_date < end_date:
        next_date = current_date + interval
        if next_date > end_date:
            next_date = end_date
        date_ranges.append((current_date.isoformat(), next_date.isoformat()))
        current_date = next_date + timedelta(days=1)

    return date_ranges
```

- La fonction `divide_date_range` prend `start_date` et `end_date` en entrée et divise la plage de dates en intervalles plus petits.

```
def save_repositories_to_csv(repositories, filename):
    keys = ["name", "language", "stargazers_count", "url", "contributors_count"]

    with open(filename, mode="w", newline="") as file:
        writer = csv.DictWriter(file, fieldnames=keys)
        writer.writeheader()

        for repo in repositories:
            row = {
                "name": repo["name"],
                "language": repo["language"],
                "stargazers_count": repo["stargazers_count"],
                "url": repo["html_url"],
                "contributors_count": get_contributors_count(repo["contributors_url"])
            }
            writer.writerow(row)
```

- La fonction `save_repositories_to_csv` prend en entrée la liste `repositories` et le nom du fichier (`filename`). Elle écrit les données des dépôts dans un fichier CSV. La fonction utilise la classe `csv.DictWriter` pour écrire les données dans un format structuré. Les clés pour les en-têtes CSV sont définies dans la liste `keys`, qui comprend des champs tels que "name" ,"language" "stargazers_count", "url" et "contributors_count". La fonction parcourt les dépôts, extrait les informations pertinentes pour chaque dépôt, puis écrit une ligne dans le fichier CSV.