Bookfarm

Application de recommandation de livres

Leclercq Nathan
Icher Paul-Henri

Qu'est-ce que c'est

- Pourquoi ça existe (quel besoin ça couvre)
- Pourquoi c'est génial
- Comment ça se met à jour (chargement et microservices qui vectorisent et garde les cluster et les embeddings à jour selon l'ensemble)

Table des Matières

Introduction - Générale

- Flux complet de récupération de données depuis le site du Furet du Nord vers la base.
- Microservices indépendant du flux complet pour vectoriser, clusteriser et figer les données éphémères (images) dans la base.
- Api pour exposer la base.
- Application interagissant avec l'api.

Introduction - Technique

- Pipeline de chargement dockerisée et paramétré avec docker-compose lancée par un cron régulièrement
- Microservices (avec api) dockerisés et paramétrés avec docker-compose pour le déploiement.
- Stockage intermédiaire en parquet, base psql avec pgvector et du jsonb pour des tags dynamiques.
- Tout le back-end est en python, api FastAPI, front-end en VueJS

Module collect

- Récupère la donnée de façon asynchrone, framework scrapy.
- Exploration minimisée en gérant la profondeur d'exploration et la pagination.
- Filtrage des données.
- Sortie en JSON (contrainte de la librairie)

Module store

- Traite la donnée brute (nettoyage, normalisation et organisation des champs).
- Initialisation et configuration de la base (si besoin, sinon gère les mises à jour de schéma).
- Chargement de la donnée en base.
- Suivi des métriques dans MLFlow.

Module microservices

- Enrichit et fige la donnée :
  - Génération de vecteurs Embedings (avec CamemBERT) pour de la reco sémantique.
  - Génération de vecteurs TFIDF pour du clustering dynamique pour garantir un temps de requête si on passe à l'échelle.
  - Téléchargement, standardisation et stockage des images de couvertures (pour l'application front-end, en cas de mise à jour du site).
- Mise en route dynamique des services en fonction des lignes dans la base.
- Suivi des runs dans MLFlow.

Module expose

- API Fast-API.
- Requete de recherche GET (filtrage sur des champs).
- Requete de recommandation (à partir d'un id).
- Requete de récupération d'image (à partir d'un id).
- Requete pour récupérer l'ensemble des collections.
- Requete pour récupérer les infos d'un livre.
- Requete de recherche POST des livres (car le GET était mal fait).

1. **GET /books**
   - Récupère une liste de livres avec des paramètres de filtrage et de pagination.
   - **Paramètres de requête** :
     - Filtrage par ID, titre, auteur, éditeur, etc.
     - Pagination avec page et page_size.

2. GET /books/{book_id}/similar

- Trouve des livres similaires en se basant sur des embeddings et des critères facultatifs.
- **Paramètres** :
  - Méthodes de similarité : cosine, euclidean, taxicab.
  - Filtres facultatifs : auteur, collection, éditeur, etc.

3. GET /books/{book_id}/image

- Récupère l'image d'un livre.
- Vérifie si l'image est téléchargée localement, sinon elle est téléchargée depuis l'URL associée.

4. GET /collections

- Récupère la liste de toutes les collections de livres distinctes.
- Retourne un tableau de noms de collections.

5. GET /book/{book_id}

- Récupère les détails complets d'un livre spécifique.
- **Paramètres** :
  - book_id : Identifiant unique du livre.
- Retourne toutes les informations détaillées du livre.

6. POST /search

- Recherche avancée de livres avec plusieurs critères.
- **Critères de recherche** :
  - nb_de_pages : Intervalle min/max
  - mot_clef : Recherche dans titre, auteur et résumé
  - collections : Liste de collections
  - date_de_parution : Intervalle avant/après

Démo App

Swagger API

Code

-> Code des réentrainement périodique

-> cron de la pipeline de chargement

-> makefile qui rchestre tout