Projet 5 Utilisez les données publiques de l'OpenFoodFacts

Lien GitHub : https://github.com/antonyboivin/Openfoodfacts

Le projet 5 relatif à la manipulation de données de l'API d'OpenFoodFacts m'a permis de me familiariser avec l'utilisation de différents formats de données volumineux, tels que le JSON et le CSV.
En effet l'API, c'est plus de 350 000 produits dont les seuls produits français sont répartis dans plus de 10 000 catégories et chaque produit comporte environ 150 champs différents.

Afin d'optimiser cette masse d'informations, j'ai dû déterminer les champs à utiliser.
J'ai commencé par le champ pays, afin de ne conserver que les produits trouvés en France.
De plus, pour répondre au cahier des charges, j'ai décidé de conserver les champs suivants:

- code-barre,
- url,
- nom du produit,
- marque du produit,
- catégorie à laquelle appartient le produit,
- magasin où le produit est vendu,
- pays,
- le grade nutritionnel
- score nutritionnel du produit.

Ma première difficulté fut de choisir un format d'export de la base de données, l'énoncé du projet semblant préconiser le format JSON.
Pourtant, en parcourant la documentation, l'utilisation d'une base de données téléchargeable au format CSV s'avérait plus pratique et plus commode à exploiter.

Bien que possible, l'utilisation du format JSON aurait eu, à mon sens, pour conséquence d'alourdir et compliquer le code de l'application car il le rend moins lisible (utilisation d'objets imbriqués dans d'autres objets, et ainsi de suite).
L'API semble réserver l'utilisation de ce format (JSON) pour la consultation d'un seul produit.
De plus, le format JSON exporte les données par page de 10 produits, ce qui impose de trouver un moyen de déterminer le nombre de pages nécessaire à l'exportation des données. C'est pourquoi, avec l'aval de Céline Martinet-Sanchez sur le workplace, j'ai décidé d'utiliser le format CSV pour l'export initial.

Il m'a suffit ensuite "d'épurer" le fichier des données non-pertinentes, c'est à dire les produits non trouvables en France ou possédant des champs importants, tel que le nutri-score vide, etc...
J'ai également choisi d'opérer un tri sur les catégories, afin de ne conserver que les plus pertinentes, c'est à dire celles contenant au minimum 10 produits. L'API comporte pléthore de catégories ne contenant qu'un seul produit, donc sans substitution possible, et ne présentant ainsi aucun intérêt pour ce projet.

Et c'est donc avec ce fichier "parsé" contenant environ 50 000 produits que j'ai entamé la 2ème partie du projet, relative à la manipulation d'une base de données MySQL ou plutôt MariaDB.

Pourquoi MariaDB ?
Après le rachat de MySQL par Oracle, MySQL devient de plus en plus commercial, alors que MariaDB reste open source.
MariaDB est la distribution qui tend à prendre la place de MySQL (elle provient du même créateur, Michael Widenius). En outre, elle est plus rapide, utilise XtraDB ou InnoDB et ajoute des capacités qu'Oracle ne fournit que sur la version commerciale de MySQL.

La création de la base de données, la création des tables, l'analyse tels que précisé auparavant et l'insertion des données se font par le biais d'un fichier unique : db_creator.py.
Une fois ce fichier exécuté, la base de données est opérationnelle.

Il ne me reste alors qu'à écrire le script de l'application.
Une autre difficulté fut de gérer la quantité de flux à l'affichage.
Afin d'alléger le nombre de catégories à l'affichage, j'ai opté pour l'utilisation d'un moyen de réglage grâce au champ score-nutritionnel.
Ce champ permet de sélectionner une "fourchette" réglable de produits.
Par exemple : On va demander à l'application d'afficher uniquement les catégories comprenant des produits ayant un score nutritionnel supérieur à X.
L'utilisateur se verra ainsi proposer des produits ayant vraiment un impact néfaste d'un point de vue nutritionnel sur sa santé.
Les autres produits contenus dans ces catégories, ayant un score nutritionnel inférieur à X, serviront de produits de substitution.

Mon application va ensuite rechercher et afficher les 5 produits ayant le plus petit score nutritionnel de la catégorie choisie par l'utilisateur.
Cet algorithme permet de toujours proposer à l'utilisateur un produit de substitution.

En conclusion, l'application pourrait être rendue encore plus performante en faisant effectuer une recherche SQL sur un champ INTEGER plutôt que sur un champ VARCHAR.
C'est-à-dire intégrer un table Catégorie qui serait en relation avec la table Products par un ID.
Par cette conception de base de donnée et de par le fonctionnement de MySQL, une telle recherche sur des nombres permettrait un gain de performance.