Python\_Projet5\_Démarche\_PierreSempéré

1 – Cahier des charges

Ce projet a présenté des problèmes non anticipés, une grande partie de ceux -ci ont été sans doute liés à une implémentation maladroite de la méthodologie (voir partie 4.).

Le cahier des charges initial prévoyait une base de données MySQL contenant 3 tables (cf. modele\_bdd.png). Même si ces tables ont considérablement évolué dans leurs formes au cours du projet, le modèle à trois tables (catégories, produits et sauvegardes) est resté inchangé.

L’approche du programme est d’utiliser l’API via un script Python pour récupérer les données utilisées pour la comparaison, créer la base de données, insérer les données récupérées et permettre à l’utilisateur de formuler ses demandes. L’approche utilisée pour récupérer les données a évoluée mais l’idée générale de ce mode de fonctionnement a été établie dès le début du projet.

Il a été important de mettre l’utilisateur potentiel au centre du programme. Rendre le script relativement transparent pour l’utilisateur tout en encadrant au maximum la gestion des inputs pour éviter les erreurs a été préféré, au dépens parfois de l’efficacité du code.

2 – Connaissances nécessaires

Il a été premièrement déterminé les connaissances nécessaires à la réalisation du projet. Ont été établis plusieurs axes de recherche :

* Les APIs (et par conséquence certaines connaissances basique sur le format .json) : le projet semble porter en premier lieu sur la compréhension de cette notion et il était indispensable de comprendre correctement le fonctionnement des APIs (en particulier celle d’Open Food Facts (OFF))
* L’autre composante essentielle du projet est l’utilisation de MySQL. A noter que l’utilisation de clés étrangères n’est pas présente dans le projet étant donné qu’il a été préféré l’utilisation de classes pour garder en mémoire les indexes.
* La compréhension de MySQL Connector, l’utilisation des regex (et la librairie re), le formatage des strings (via deux méthodes différentes dans le programme : %s et f-string) a été nécessaire pour améliorer l’UX et permettre le fonctionnement des requêtes SQL.

3 – Structure Globale

Le programme est structuré autour du script « main » (nom sujet à changement) et l’import de modules contenant les fonctions nécessaires au fonctionnement du programme.

* Le module le plus « lourd » est « display\_queries.py », il regroupe toutes les fonctions servant à l’affichage et la sélection de données dans la base de données.
* Le module « request\_script.py » fonctionne differment des autres modules étant donné qu’il n’est pas utilisé pour mettre des fonctions a disposition du script principal mais pour retourner les données de l’API. Il est aisément modifiable pour que l’on puisse ajuster la quantité et le type de donnée qu’on souhaite récupérer.
* Les modules « sql\_executor.py » et « data\_insertion.py » sont deux modules servant uniquement lors de la création de la base de données : le premier « lit » le fichier sql passé en paramètre (ici le script de création de la BDD) et exécute les commandes via le curseur (le script a pour but d’ignorer les commentaires, accoler les lignes ne comportant pas le caractère « ; » (signifiant la fin d’une commande) jusqu’à ce qu’une ligne comportant ce caractère soit détectée. A ce moment, la commande est exécutée par le curseur). Le module d’insertion de donnée se sert simplement des données récoltées par le script de requête et insère ces données dans la base.
* Le module « input\_regulation.py » est simplement utilisé pour anticiper les erreurs pouvant être commises par l’utilisateur et permet d’éviter la plupart des erreurs mettant fin prématurément au programme
* Le module « history.py » utilise uniquement la table contenant les données sauvées par l’utilisateur et lui permet de sélectionner une entrée précédemment sauvée dans la BDD pour revoir les informations principales du substitut
* Le script « main » exécute dans l’ordre : historique, création de BDD, requête et insertion, puis « display\_queries.py » pour finir par un prompt demandant à l’utilisateur s’il souhaite sauver sa recherche.

Le parti pris est d’exécuter le script une seule fois et de ne pas engager une répétition du script en boucle jusqu’à ce que l’utilisateur utilise une commande d’arrêt. Cela est aisément modifiable mais je trouve que ce format est mieux adapté.

4 – Problèmes et solutions

Le principal problème rencontré lors de ce projet a été l’implémentation de la méthodologie. Force est de reconnaitre que l’utilisation de la méthode agile a été utilisée bien trop tard et de façon trop superficielle pour être un atout du projet. J’ai majoritairement utilisé un support papier pour organiser mon travail, ce qui, rétrospectivement, est particulièrement inefficace.  
 L’utilisation de méthode agile, l’utilisation d’un KANBAN via Taiga.io (par exemple) et le DocDrivenDevelopment devraient être des automatismes que je ne possède pas encore, probablement du fait que ces méthodes me sont relativement nouvelles. Néanmoins, il est évident que l’utilisation de ces méthodes auraient probablement permis une diminution significative du temps de développement et une amélioration sensible de l’ergonomie du programme.

L’autre problème rencontré a été une mauvaise approche lors de l’utilisation de l’API. Lors du début du projet, il a été décidé de partir de l’endpoint de la liste totale des produits (*reference*) pour, d’abord, récupérer les produits et ensuite leur associer des catégories. Cette méthode a prouvé être largement inefficace ou impossible du fait de la structure de l’API (les catégories ne sont pas organisées de façon hiérarchique, rendant l’automatisation de la création de sous catégories impossible ou trop imprécises pour être pertinentes).  
 L’approche préférée dans ce système est de, tout d’abord, récupérer un échantillon de catégories répondant a certains critères pour limiter les catégories composites et éviter un maximum les doublons et les overlaps. Cet échantillon de catégorie est ensuite utilisé pour récupérer les produits associés à ces catégories. On obtient comme cela la catégorie associée à chaque produit de manière évidente et cela permet la comparaison du produit.

Lien du projet sur GitHub : <https://github.com/Psemp/OC_P5>