Etapes

1 - Organiser son travail

Initialisation du repo sur Github

Ecrire des user stories pour faire des taches et sous taches.

user_stories.md

Créer un tableau agile et mettre des deadlines.

https://trello.com/b/NFvfd67Q/ocdapythonpr5

Ecrire la documentation pour faire du DDD.

README.md

2 - Construire la base de données

Analyse de l'API OpenFoodFacts

Ecrire le modèle physique de données.

Document de conception fonctionnelle

Avoir un script pour la création de la base.

Ecrire un script qui récupère les données de PenFoodFacts pour les mettre dans notre base.

Intégré dans le module database.

> create_database()

> fill_in_database()

> _fill_with_off_data()

Un script permet de lancer ces méthodes et de réinitialiser les données.

createdb.py

3 - Construire le programme

https://github.com/Zepmanbc/oc_dapython_pr5

```
pur_beurre.py
config.py
createdb.py
    app/
        database/
            database.py
```

```
        gui/
            Créer gui.py
        static/
            categories.json
            dboff.sql
```

4 - Interagir avec la base de données

Les requêtes sont dans le module *database*.

La gestion des fenêtres sont dans le module *gui*.

Difficultés rencontrées

Structure du main pour la logique de séquençage des fenêtres

La fenêtre de détail du substitut pouvant arriver par 2 chemins possible, je ne pouvais pas savoir quel était la fenêtre précédente (la liste des substituts proposé ou la liste des substituts sauvegardés). J'ai donc décidé de passer par une liste *current_screen* afin d'empiler les fenêtres. La fenêtre active étant la dernière, elle est "dépilée" quand on revient en arrière.

Cette méthode me permet également de revenir à la page sur laquelle l'utilisateur se situait avant dans passer à l'écran suivant.

Si l'on souhaite rajouter des fonctionnalités, il sera également possible de "dépiler" toutes les fenêtres pour revenir à la page d'accueil.

L'appel des procédures qui faisaient planter le curseur

J'ai choisi d'utiliser des procédures pour les requêtes un peu plus "complexes" que juste l'interrogation d'une table.

J'ai une procédure *get_better_product* qui me permet uniquement à partir du product_id de récupérer la catégorie et le grade du produit d'origine et de faire une recherche sur ces éléments. Cette procédure faisant appel à des requêtes imbriquées, le retour dans le curseur est un itérable. J'ai mis du temps à m'en rendre compte...

la seconde procédure *show_detail* permet une double requête sur les éléments et de vérifier si le binôme existe déjà dans la table Substitute.

J'ai également eu recours à une vue *V_Substitute* pour me simplifier le résultat de la liste des substituts sauvegardés.

L'injection du fichier SQL

Je n'ai pas trouvé de résultat satisfaisant pour utiliser le même fichier à injecter et celui à parser.

Il y a un plantage au niveau du DELIMITER |

j'ai proposé une version sur stackoverflow pour injecter directement le fichier dans la base mais cette solution ne me convient pas, de plus j'utilise une base dans un container docker donc cela rend la commande beaucoup trop spécifique.

Gestion des pages et de l'ordre des éléments

L'affichage des produits doit être affiché aléatoirement mais le passage d'une page à l'autre ne peut pas relancer une requête aléatoire, on risque de ne jamais retrouver un produit. J'ai donc opté pour enregistrer le résultat de la requête aléatoire dans une liste découpé par page de 9 produits.

La gestion des listes de produits, de substituts et de substituts sauvegardés sont géré de la même manière avec une liste. Celle-ci est vidée lorsque l'on revient en arrière dans l'interface.