Pur Beurre

Repository : https://github.com/CoLoDot/Pur_beurre
Trello : https://trello.com/invite/b/dGsNJQKS/82d7aec6933a2f76934a6f996295cd2a/pur-beurre
Lien : https://vivelegras.herokuapp.com/

Pur Beurre est le huitième projet de la formation développeur d'application Python d'OpenClassrooms. La réalisation de ce projet permet de découvrir le framework Django et d'approfondir les notions préalablement acquises en front-end, notamment concernant Bootstrap. Pur Beurre est donc un projet web croisant toutes les étapes d'un projet professionnel full-stack d'application web basé sur une demande client avec un cahier des charges imposé.

1 - Mise en place du projet

Avant la réalisation en bonne et du forme du projet Pur Beurre, il fut nécessaire d'effectuer le cours « Découvrez le framework Django » d'OpenClassrooms. L'acquisition des notions fut agréable étant donné le fait que le projet précédent donnait lieu à la découverte du framework Flask.

Ainsi, j'ai pu approfondir des notions déjà vu et en acquérir de nouvelles plus rapidement (découverte des modules d'authentification, gestion d'une base de données par le biais d'un ORM, etc.).

La découverte de la notion d'ORM m'a permis d'approfondir la gestion d'une base de données et l'insertion de données externes récupérées au format JSON.

2 - Réalisation

Suite à la période d'acquisition des connaissances en Django, j'ai fait le choix de démarrer le projet en implémentant la partie front-end afin d'avoir un support visuel à mes premiers essais concernant l'algorithme de recherche des produits sur OpenFoodFacts et le résultat qui en ressortait.

J'ai donc fait en sorte de travailler de manière Agile en passant de la partie front au back, afin d'améliorer de concert les deux aspects du projet. J'ai donc commencé par travailler sur la vue Index (page d'accueil du site) qui contient le champ de recherche principal de l'application (le second étant visible en permanence dans la barre de navigation). Après la mise en place de cette première page, j'ai fait en sorte d'avoir une page avec un compte utilisateur donnant accès à des produits sauvegardés.

Cela m'a permis de travailler sur l'algorithme de recherche des produits dans OpenFoodFacts par le biais de l'API et de pouvoir visualiser directement sur mon application la manière dont s'articulait les données dans la charte graphique que j'avais suivis. A titre d'exemple, j'ai fait le choix d'utiliser les images de nutriscore d'OpenFoodFacts pour plus de lisibilité au lieu de simplement afficher la lettre correspondant au nutriscore, qui ne parle pas forcément à l'utilisateur en quête de substitut.

De manière plus générale, l'algorithme de recherche est ce qui a nécessité le plus de temps. De fait, il a fallut, dans un premier temps, travailler sur la récupération de la requête utilisateur et par la suite, vérifier si un produit correspondait à cette requête par le bien d'une comparaison de string disponible grâce à l'ORM de Django (*Model.objects.filter(name__icontains=marequête)*). Une fois cette recherche en table complète, c'est par le biais d'un If statement que mon algorithme de recherche s'actionne. Ainsi, si la requête utilisateur ne donne aucun résultat, l'algorithme envoie une requête (par le biais du module Requests) vers OpenFoodFacts en intégrant dans l'url de requête une string contenant le requête utilisateur. Une fois obtenue, la réponse est récupérée au format JSON, ensuite, l'utilisation d'une boucle for sur les index de produits dans un intervalle allant de 0 au total du nombre de produit obtenue dans la réponse permet de récupérer, au moyen des clefs à dispositions, les données nécessaires à l'insertion d'un nouveau produit en base de données (name, nutriscore, category, picture, url). Une fois ces données assignées à des variables, c'est encore via l'ORM de Django qu'elles sont insérées et qu'un nouvel item produit est ajoutée à la table.

En ce qui concerne la sauvegarde des produits par les utilisateurs, j'ai fait le choix de la simplicité en ne sauvegardant qu'un minimum d'information : le contact, la clé du produit et sa date de sauvegarde. Cela permet ensuite de filtrer les résultats et d'afficher tous les produits correspondant à un contact en particulier, si celui-ci est connecté à l'application. Si l'utilisateur n'est pas connecté, aucun produit ne s'affiche en page 'UserProducts'.

3 - Difficultés

Au cours de la réalisation du projet, plusieurs difficultés ont point. En premier lieu, le système d'authentification de Django, sur lequel j'étais néophyte, m'a posé quelques problèmes, notamment concernant la fonction login().

La seconde difficulté fut celle de la recherche par catégorie en base pour retourner des substituts dans l'application en fonction de la requête utilisateur. Je suis parvenue à régler ce problème en découvrant la fonction ArrayFields() de Django, qui permet d'associer à une colonne d'un modèle un champ qui peut accueillir une liste de données.

Ainsi, j'ai pu insérer les différentes catégories d'un produit issue de la réponse JSON sans difficultés. Par là suite, cela m'a permis d'utiliser le filtre « contained_by » de l'ORM qui effectue une recherche dans la liste de catégories de produit et recherche les catégories similaires dans les autres items dans le but de restituer au final plusieurs produits faisait partie de la même catégorie, et qui peuvent donc se substituer potentiellement au produit correspondant à la requête utilisateur.