Maxime J.

Créez une plateforme pour les amateurs de Nutella – Livrables

- Lien vers votre site en production, entièrement fonctionnel : https://nutella-stop.herokuapp.com/
- Document écrit expliquant votre démarche de création, les difficultés rencontrées et la manière dont vous les avez résolues. Incluez-y le lien vers votre board Trello ou Pivotal Tracker, le lien vers votre repo Github et le site en production. Le document doit être en format pdf et ne pas excéder 2 pages A4. Il peut être rédigé en anglais ou en français, au choix, mais prenez bien en considération que les fautes d'orthographe et de grammaire seront évaluées !

I. Généralités

La startup Pur Beurre souhaite faire évoluer les habitudes alimentaires des Français en proposant à quiconque de trouver un substitut sain à un aliment considéré comme « Trop gras, trop sucré, trop salé ».

Afin de répondre au mieux cahier des charges dont Pur Beurre nous a fait part, nous avons créé un programme codé en HTML, CSS, JavaScript et Python sous Visual Studio Code 1.35.1. Le front-end est développé grâce au Framework Bootstrap permettant la gestion du responsive design, et le back-end grâce au Framework Django. Le programme fait appel à l'API d'OpenFoodFacts, ceci afin d'insérer et récupérer en base de données PostgreSQL les informations des produits des catégories suivantes : Huile d'olive d'Aix-en-Provence, Sauces au roquefort, Œufs, Beurres, Muffins au chocolat, Bœuf, Beurres de cacahuètes, Saumons, Croissants au beurre.

Le programme est hébergé sur GitHub : https://github.com/Maximedu13/plateforme-pour-amateurs-de-Nutella ainsi que sur le serveur Heroku : https://nutella-stop.herokuapp.com/. Le site web est fonctionnel.

II. Parcours utilisateur et démarche de création

En synthétisant le cahier des charges, on peut en déduire le parcours de l'utilisateur utilisant l'application :

- Celui-ci peut naviguer sur les pages d'accueil, des mentions légales, des produits cherchés et substituts renvoyés sans être connecté.
- Celui-ci doit pouvoir consulter les pages produits (score nutritionnel, énergie etc.)
- Celui-ci doit pouvoir s'inscrire et s'authentifier sur l'application via un formulaire. Il doit pouvoir aussi se déconnecter.
- Celui-ci peut ajouter un substitut sain dans ses favoris s'il est connecté.
- Celui-ci peut consulter ses pages de profil et de produits favoris s'il est connecté.
- Celui-ci doit retrouver la charte graphique donnée. Pour la photo en fond de la page d'accueil, nous avons, parmi les trois photos proposées, choisi la numéro une.

Afin de répondre au mieux aux besoins précédemment cités, nous avons choisi d'adopter une approche Test Driven Development dite TDD, qui consiste à déterminer respectivement la liste des fonctionnalités à fournir, rédiger la documentation, coder, vérifier si la documentation est bien respectée et enfin itérer.

Nous avons donc commencé par créer un tableau Trello, disponible à cette adresse : https://trello.com/b/F9RQBizK/cr%C3%A9ez-une-plateforme-pour-amateurs-de-nutella.

Nous avons découpé le programme en user stories et en tâches, puis nous avons créé le repository de votre application sur GitHub. Ensuite, nous avons commencé à nous interroger sur la structure

de votre base de données PostgreSQL : Quelles informations doivent y figurer ? Afin de garder en tête une vue globale de l'application des relations entre les tables, nous avons créé un diagramme SQL (diagramme de classes) via l'outil draw.io. Nous avons commencé par créer le projet Django et avons attaque le front-end en y intégrant la librairie Bootstrap. Tout en testant notre application, nous nous sommes intéressés à l'API OpenFoodFacts et avons construit un programme qui insère en base de données les produits, puis qui les affiche lorsque l'utilisateur les recherche. En fonction du nombre de résultats correspondant, il se peut qu'il y ait plusieurs pages. La pagination est assurée par le module pagination de Django. Chaque produit a une page spécifique, qui affiche différentes informations nutritionnelles et géographiques. Qui dit produit, dit substitut, nous avons donc crée un algorithme permettant d'afficher les substituts plus sains suivant la recherche. Tout de suite après, nous avons géré les formulaires, l'authentification en Django pour que l'utilisateur puisse créer un compte ou se connecter. Nous avons également géré les actions de l'utilisateur connecté (Mes favoris + Mon compte + déconnexion) et les messages flash associes à ces actions. Il a aussi fallu gérer les erreurs, par un mot de passe erroné, une recherche n'aboutissant à aucun résultat etc… Enfin, nous nous sommes penchés sur la fonctionnalité permettant à l'utilisateur de sauvegarder un produit en favoris, ainsi que sur la page où l'utilisateur les retrouve, et puis nous avons déployé votre site web sur Heroku tout en nous assurant que notre code respectait la PEP8.

III. Études algorithmiques

Dans cette partie, est développé quelques algorithmes utilisés dans le programme.

L'algorithme de création de modèles et d'insertion des produits en base de données :

Cet algorithme ressemble sensiblement à celui utilise précédemment lors du projet 5 Openclassrooms *Utilisez les données publiques de l'OpenFoodFacts*. La méthode gérant cela dans l'application est insert(). Elle se trouve dans le fichier database.py. Tout d'abord, nous construisons une liste en Python avec à l'intérieur toutes les catégories que nous souhaitons insérer en base de données. Pour chaque produit présent dans cette liste on crée un objet Category et on appelle via le module request l'API d'OpenFoodFacts. Pour chaque requête, suivant le nombre de produits, on essaie par un bloc try/except d'assigner les valeurs qui nous intéressent aux valeurs JSON d'OpenFoodFacts. On crée les objets Product. S'il y a une erreur, le produit n'est pas inséré.

L'algorithme d'auto complétion en AJAX :

Il s'agit pour l'utilisateur d'avoir une liste de produits (10) qu'il recherche et de choisir celui qu'il veut, et cela avant de lancer la recherche de substitut. L'utilisateur doit taper minimum deux lettres dans le champ prévu à cet effet. La méthode associé, autocomplete() s'assure tout d'abord que la requête est de type AJAX. Si c'est le cas, on récupère la recherche effectuée qui va rechercher parmi tous les produits ceux dont le nom est semblable, tout ceci dans une liste et un dictionnaire. On se limite à 10 résultats.

On appelle cette fonction dans index.html par le biais d'une balise <script>.

L'algorithme de recherche de substituts plus sains :

La méthode permettant de trouver des substituts a un produit et de les afficher est substitute(). Elle se charge de charger le Template associé à la fonction et récupère la requête demandée via request.GET.get('query'). Le filtre Django se charge de trouver, pour le produit recherché, des substituts qui appartiennent à la catégorie de ce dernier et qui ont un score nutritionnel A. S'il n'y en a aucun, alors on cherche des substituts qui ont un score nutritionnel B etc.

Enfin, on gère la pagination.

IV. Difficultés rencontrées et solutions trouvées

De nombreuses erreurs de codage et avertissement ont été des obstacles au bon fonctionnement du programme :

✗ Django – getting Error "Reverse for 'id' with no arguments not found. 1 pattern(s) tried.

? Django ne trouve pas l'url ou l'id associé à la requête.

✓ Dans notre cas, le lien comportait une erreur. Il s'agissait, pour être redirige vers la page du produit d'écrire `<a href="{% url 'catalog:detail' product_id=product.id %}">` et non `<a href="{% url 'catalog:detail product.id ">`

✗ Django: TemplateSyntaxError: Could not parse the remainder.

? Syntaxe particulière du moteur de templates Jinja. Dans notre projet par exemple il n'est pas possible d'écrire directement dans le fichier HTML : {% for i in range(1, 100} %}

✓ Créer une fonction dans views.py et l'appeler directement dans le fichier HTML.

Nous avons également éprouvé quelques difficultés lors de la gestion de la pagination sous Django, mais en procédant de façon logique, nous avons réussi à permettre à vos futurs utilisateurs de naviguer entre les pages de résultats plus confortablement.

V. Améliorations envisageables

Le site web réalisé répond aux attentes de notre client, il reste améliorable, et peut être complété par les fonctionnalités suivantes :

- L'ajout d'un formulaire de contact. Pour le moment, les utilisateurs de l'application souhaitant entrer en contact avec la startup doivent se rendre sur la page d'accueil et cliquer sur leur adresse mail. Cela les redirige vers leur boite de messagerie par défaut. C'est assez contraignant, voire dissuasif. Un formulaire de contact est une façon plus rapide, et efficace pour un utilisateur de contacter Pur Beurre.
- La possibilité pour les utilisateurs de personnaliser leurs profils, par exemple en ajoutant le choix d'une photo de profil.
- La possibilité pour les utilisateurs de changer leurs mots de passe et leurs adresses-mail.
- La possibilité pour les utilisateurs de laisser un commentaire sur un produit.