

Projet 8 - Créez une plate-forme pour amateurs de Nutella

Parcours OpenClassrooms - Développeur d'application Python

Étudiant : Guillaume OJARDIAS Mentor : Erwan KERIBIN
Mentor évaluateur : Babacar SYLLA

Présentation

Objectif

- ▶ Trouver en un clic un produit de substitution

Cahier des charges

- ▶ Avoir un compte utilisateur
- ▶ Sauvegarder des substituts dans ses favoris
- ▶ Esquisses de disposition des pages
- ▶ Utilisation du template Creative proposé par StartBootstrap

Démarche de création

Etapes du projet

- ▶ Création du projet Django
- ▶ Mise en place de la gestion des utilisateurs
- ▶ Création des modèles de données (Product et Category)
- ▶ Insertion des produits depuis la base de données de l'Open Food Facts
- ▶ Mise en place du système de recherche des substituts
- ▶ Ajout de la fonctionnalité de mise en favoris

Code source

- ▶ Python + Django
- ▶ Github -> Pur-Beurre
- ▶ Découpage du projet

Arborescence du projet

```
|-- config          #--- Configuration du projet
|-- homepage        ##
|-- openfoodfacts   #
|-- product         #-- Applications
|-- search          #
|-- users           ##
|-- static          #--- Fichiers statiques
|-- templates       #--- Templates généraux
|-- tests           ##
    |-- homepage    #
    |-- openfoodfacts #
    |-- product     #-- Tests
    |-- search      #
    |-- users       ##
```


Algorithme de recherche d'un substitut

- Filtre pour le calcul des catégories en commun

```
q = Q(categories__in=product.categories.all()) & Q(
    nutriscore_grade__lt=product.nutriscore_grade))
```

- Requête pour obtenir les substituts

```
substitutes = (Product.objects.annotate(
    common_categories=Count("categories", filter=q))
    .order_by("-common_categories", "nutriscore_grade")
    .exclude(code=product.code)
    .exclude(name=product.name)[:30])
```

Tests 1/3

Test sur la classe Api de l'application openfoodfacts :

- Mock de la class Response :

```
class MockRequestsResponse:
    def __init__(self, json_data, status_code):
        self.json_data = json_data
        self.status_code = status_code

    def json(self):
        return self.json_data
```

Tests 2/3

Test sur la classe Api de l'application openfoodfacts :

- Mock de la méthode get de la class requests

```
class MockRequests:
    def __init__(self, data, status_code):
        self.data = data
        self.status_code = status_code

    def get(self, *args, **kwargs):
        return MockRequestsResponse(
            self.data,
            self.status_code)
```

Tests 3/3

Test sur la classe `Api` de l'application `openfoodfacts` :

► Utilisation du mock

```
def test_api_return_products(self):  
    data = { ... }  
    mock_requests_get = MockRequests(data, 200).get  
  
    # Patch de la méthode Get  
    with patch(  
        "openfoodfacts.api.requests.get", mock_requests_get):  
        products = Api().get_products()  
  
        self.assertEqual(products, data["products"])
```

Bilan du projet

Découpage du projet

- ▶ Perfectible
 - ▶ Changer le nom de homepage
 - ▶ Regrouper les templates
- ▶ Penser son découpage en amont
 - ▶ Changement en cours difficile
 - ▶ Facilité avec l'expérience

Tests

- ▶ **Coverage** est un très bon outil
 - ▶ Utile pour mettre en relief les zones non testées
 - ▶ Attention à avoir un esprit critique
- ▶ Le **Test Driven Development**
 - ▶ Indispensable
 - ▶ Difficile pour un débutant
 - ▶ Vient avec le temps et l'expérience

Django

- ▶ Nombreux outils disponibles *out of the box*
 - ▶ Users
 - ▶ Admin
 - ▶ API (non utilisé sur ce projet)
- ▶ Part de *magie* apprécié par certains
- ▶ Très puissant pour une utilisation poussée