

PROJET 8

CRÉEZ UNE PLATEFORME POUR AMATEURS DE NUTELLA

LOUDOT EMMANUEL
DÉVELOPPEUR D'APPLICATION PYTHON

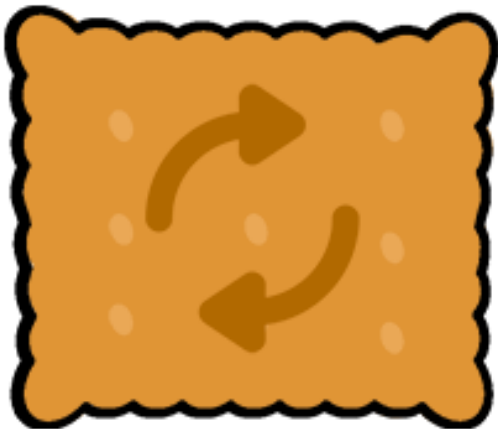


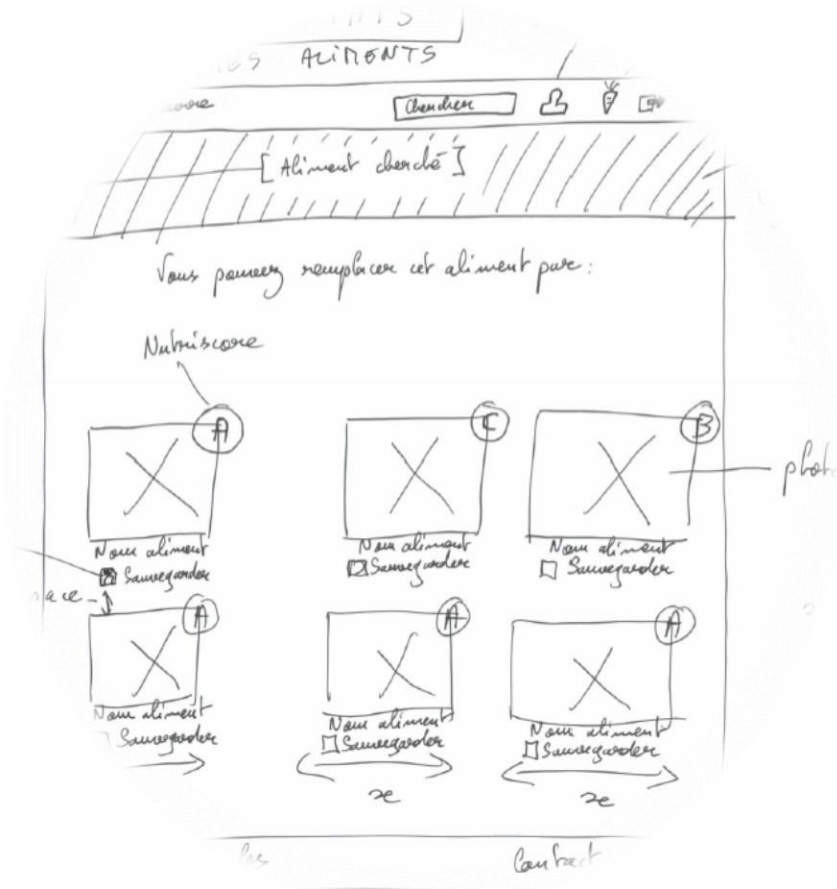
LE PROJET

Le projet consistait à développer une application web pour la société Pur Beurre.

Le site permet à quiconque de trouver un substitut sain à un aliment considéré comme "Trop gras, trop sucré, trop salé".

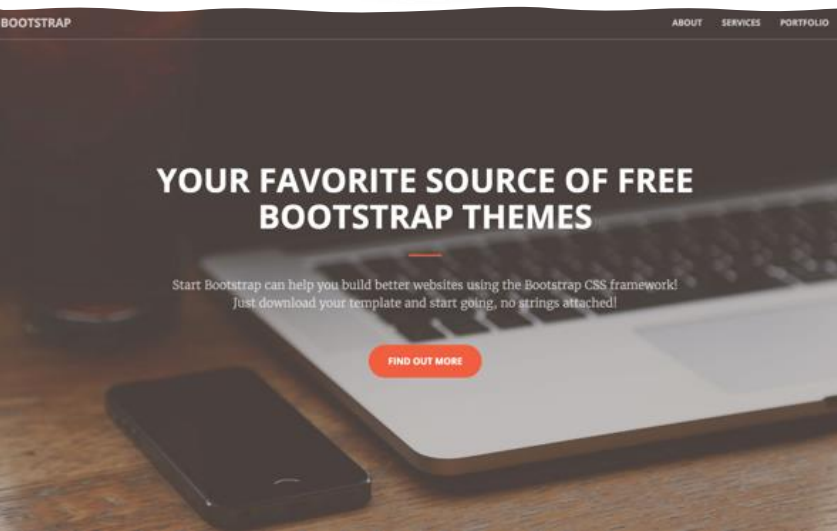
Il devait être développé avec le framework django et respecter le cahier des charges fournis.





CONSTRAINTES / FONCTIONNALITÉS

- Utilisez une méthodologie de projet agile pour travailler en mode projet.
- Testez votre projet
- Utilisez une base de données PostgreSQL
- Incluez une page "Mentions Légales"
- Suivez les bonnes pratiques de la PEP 8
- Poussez votre code régulièrement sur Git-Hub
- Code doit être intégralement écrit en anglais
- Affichage du champ de recherche dès la page d'accueil
- La recherche ne doit pas s'effectuer en AJAX
- Interface responsive
- Authentification de l'utilisateur :
création de compte en entrant un mail et un mot de passe, sans possibilité de changer son mot de passe pour le moment.
- Respecter le cahier des charges



PROJET : PUR BEURRE

- 1 – Planification du projet
- 2 – Structure du projet
- 3 – Points particuliers de l'application utilisateurs
- 4 – Points particuliers de l'application produits
- 5 – Front-end
- 6 – Tests
- 7 – Déploiement

1- PLANIFICATION DU PROJET

TABLEAU TRELLO

The screenshot shows a Trello board interface. At the top, there is a navigation bar with links for 'ACCUEIL', 'VISITE GUIDÉE', and 'BLOG'. The Trello logo is in the center, and on the right are buttons for 'S'inscrire' and 'Se connecter'. Below this is a yellow banner with the text 'Voulez-vous vous abonner à ces cartes ?' and a green button 'Inscrivez-vous gratuitement'. The board itself has a header with the name 'P8PurBeurre', a 'Public' status, and a user profile icon. A menu button 'Afficher le menu' is on the right. The board is divided into two main sections: 'A faire' (To Do) and 'En cours' (In Progress). The 'Fait' (Done) column on the right contains a list of tasks: 'Créer une base de donnée pour le projet', 'Initialiser un nouveau projet django', 'Ajouter deux applications une pour la gestion des utilisateurs et une pour la recherche des produits', 'Créer un template basique', 'Ajouter la création de compte', 'Tester la création de Compte', 'Ajouter l'authentification', 'Tester l'authentification', 'Ajouter la fonctionnalité pour peupler la base de donnée depuis OpenFoodFact', 'Tester le peuplement de la base de donnée', 'Créer la recherche de produit', 'Tester la recherche', 'Créer la recherche de substitut', 'Tester la recherche de substitut', and 'Ajouter l'enregistrement en favoris'.

ACCUEIL VISITE GUIDÉE BLOG

Trello

S'inscrire Se connecter

Voulez-vous vous abonner à ces cartes ? Inscrivez-vous gratuitement

P8PurBeurre Public E

A faire En cours

Fait

- Créer une base de donnée pour le projet
- Initialiser un nouveau projet django
- Ajouter deux applications une pour la gestion des utilisateurs et une pour la recherche des produits
- Créer un template basique
- Ajouter la création de compte
- Tester la création de Compte
- Ajouter l'authentification
- Tester l'authentification
- Ajouter la fonctionnalité pour peupler la base de donnée depuis OpenFoodFact
- Tester le peuplement de la base de donnée
- Créer la recherche de produit
- Tester la recherche
- Créer la recherche de substitut
- Tester la recherche de substitut
- Ajouter l'enregistrement en favoris

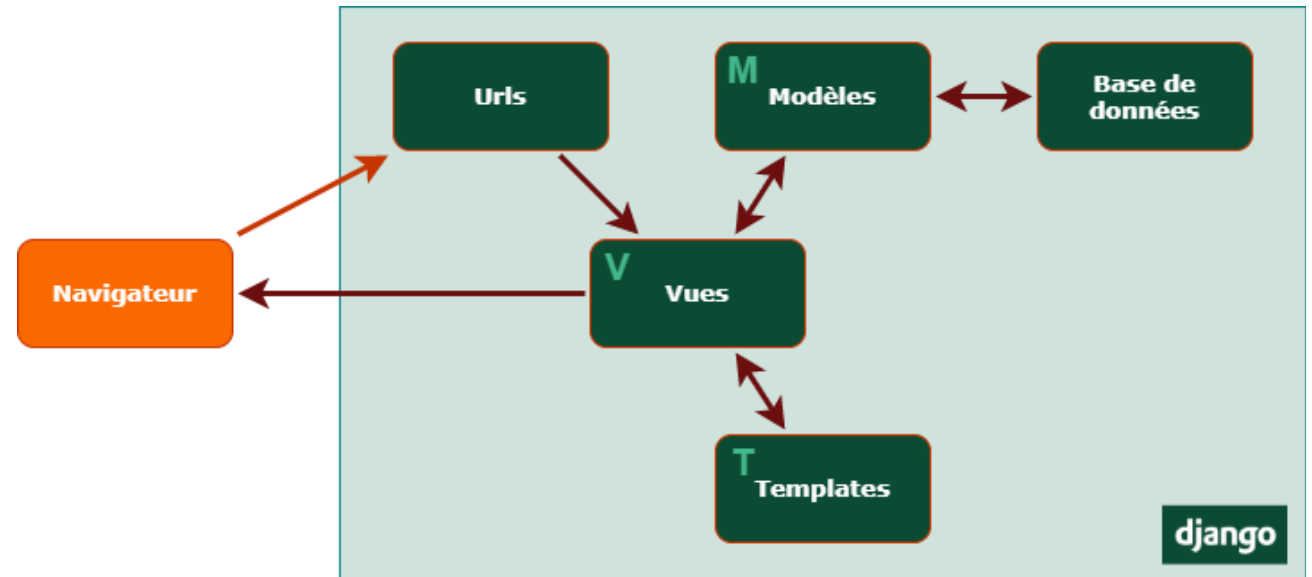
... Afficher le menu

2 - STRUCTURE DU PROJET

DJANGO - ARCHITECTURE MVT

Django est un Framework qui utilise l'architecture MVT:

- Le Modèle: permet d'interagir avec la base de données via un ORM
- La Vue : va traiter la requête HTTP puis utiliser les modèles et templates pour créer et renvoyer une réponse HTTP
- Le Template : un gabarit HTML utilisé par la vue.



L'utilisateur envoie une demande à Django via une URL.

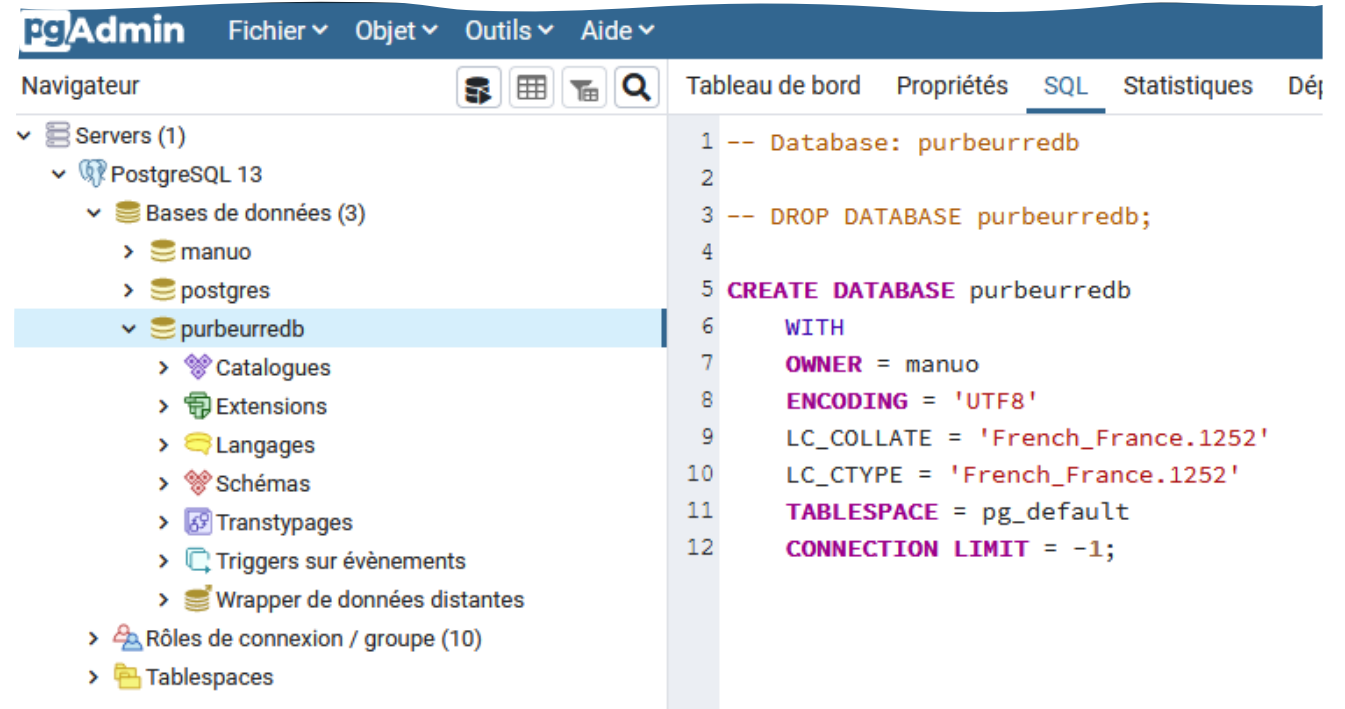
Si l'URL est liée à une vue, celle-ci est appelée.

La vue interagira avec les modèles pour récupérer les données nécessaires dans la base de données.

La vue crée ensuite un contexte, l'intègre à un template puis retournent une réponse HTTP

BASE DE DONNÉE

Le projet utilise une base de données PostgreSQL que j'ai créé mais les tables et données sont entièrement générés par django



UN PROJET PURBEURRE

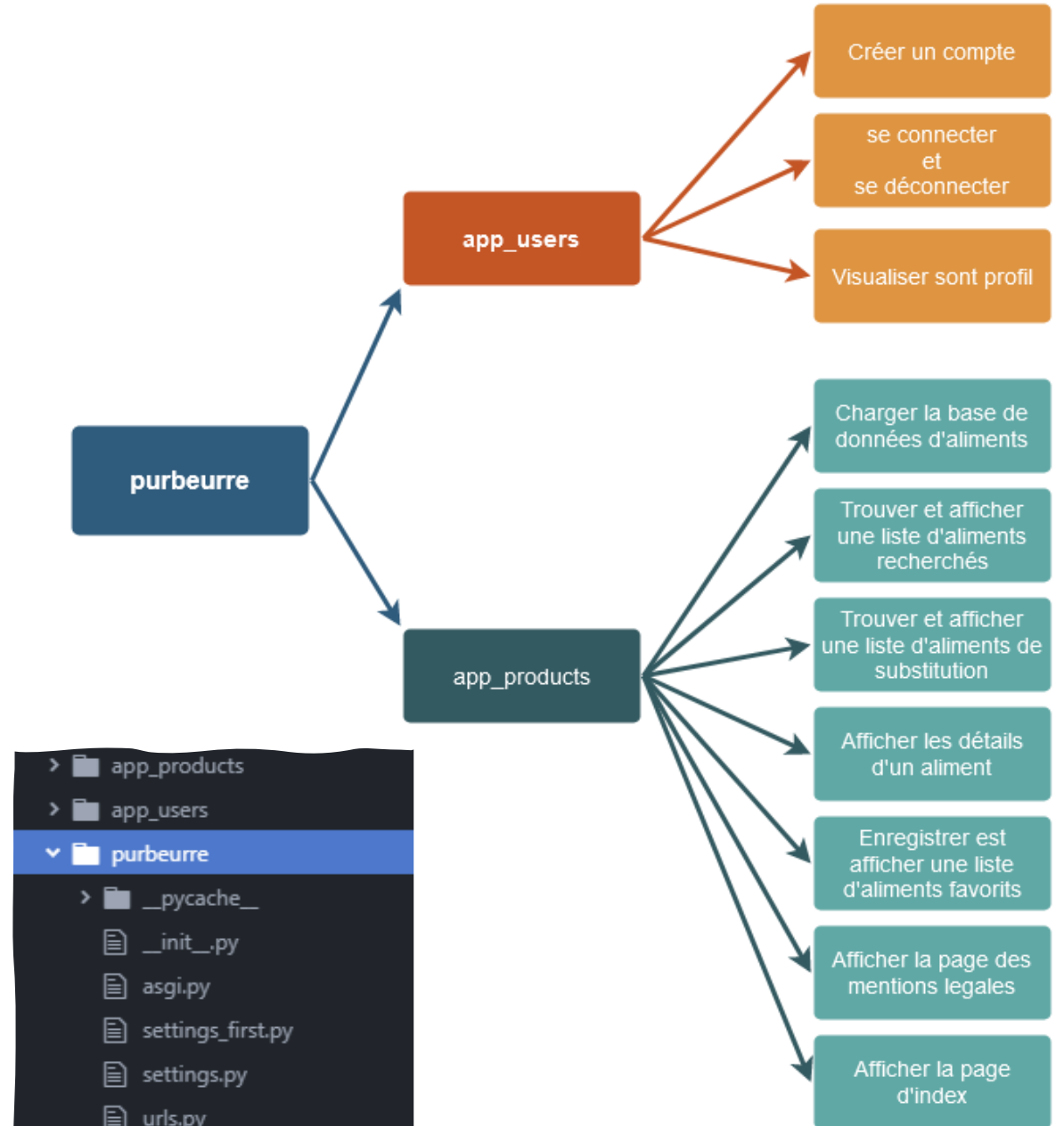
DEUX APPLICATIONS

Le projet est l'application Web, il contient principalement la configuration du projet

Les applications sont des paquets Python qui fournissent des ensemble de fonctionnalités.

Le projet purbeurre est séparé en deux applications

- app_users: pour la gestion des utilisateurs
- app_products: pour la recherche des



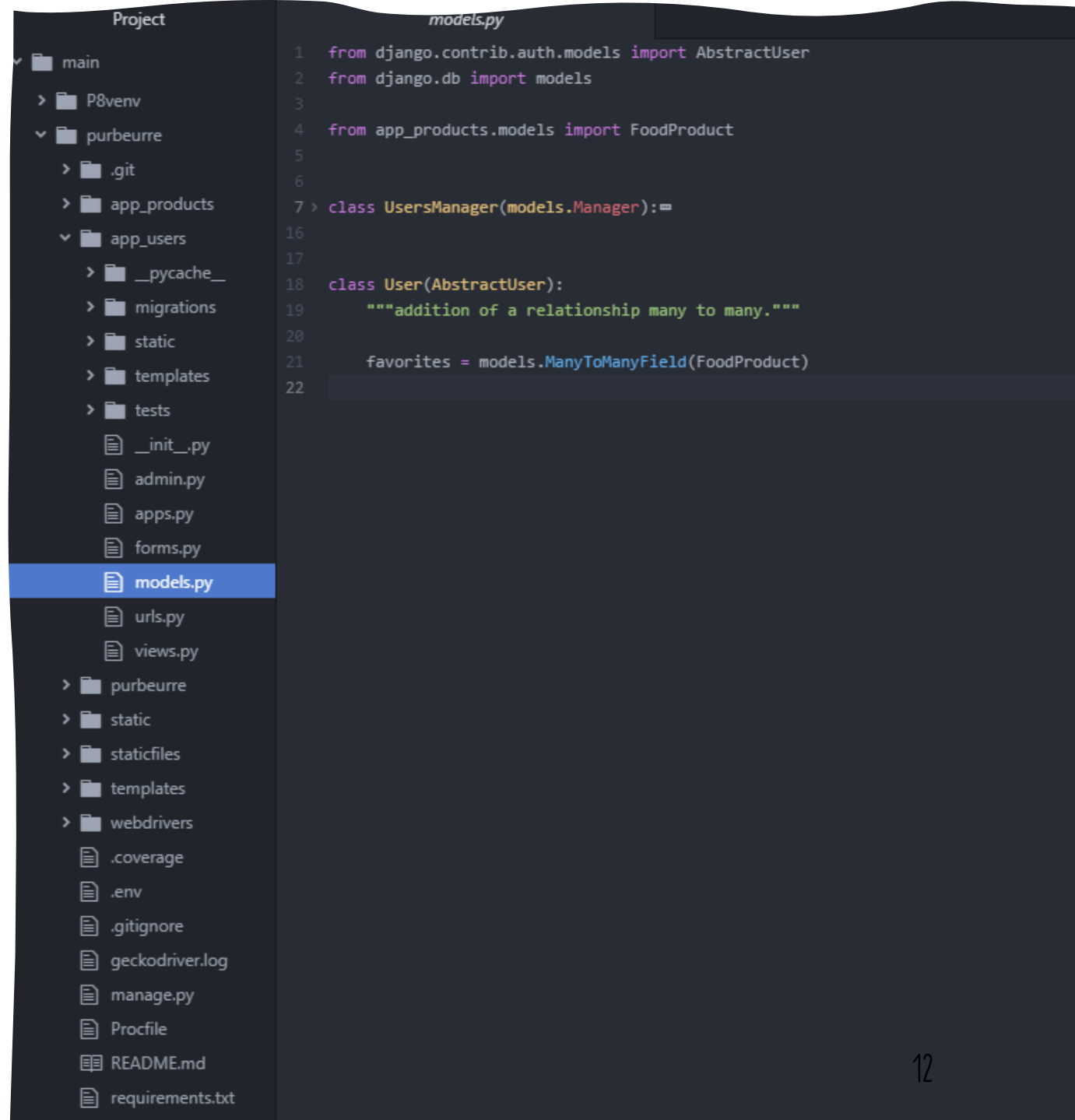
3 – POINTS PARTICULIERS DE L'APPLICATION

APP_USERS

MODÈLE D'UTILISATEUR PERSONNALISÉ

Pour permettre l'enregistrement des favoris il était nécessaire d'ajouter une relation plusieurs-à-plusieurs entre la table des utilisateurs et la table des produits alimentaires.

Il a donc été nécessaire de créer un modèle d'utilisateur personnalisé pour ajouter cette relation.



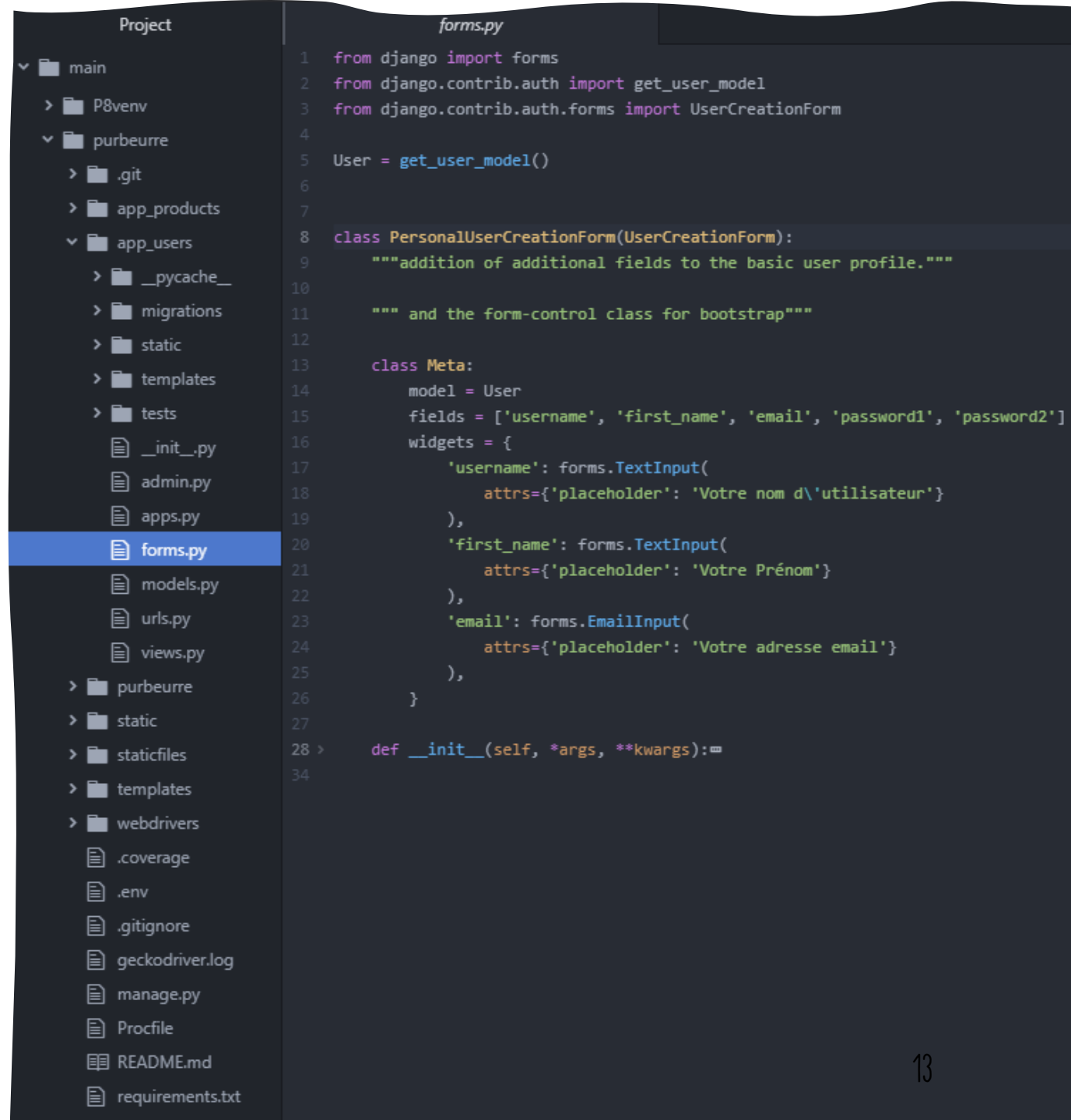
The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'main', 'P8venv', 'purbeurre', '.git', 'app_products', 'app_users', '__pycache__', 'migrations', 'static', 'templates', 'tests', and files like '__init__.py', 'admin.py', 'apps.py', 'forms.py', 'models.py', 'urls.py', and 'views.py'. The 'models.py' file is selected and highlighted in blue. The code editor on the right shows the content of 'models.py' with the following code:

```
1 from django.contrib.auth.models import AbstractUser
2 from django.db import models
3
4 from app_products.models import FoodProduct
5
6
7 > class UsersManager(models.Manager):=
16
17
18 class User(AbstractUser):
19     """addition of a relationship many to many."""
20
21     favorites = models.ManyToManyField(FoodProduct)
22
```

EXTENSION DU FORMULAIRES D'AUTHENTIFICATION

Le formulaire standard de création d'utilisateur de django n'utilise que le nom d'utilisateur et le mot de passe.

Pour enregistrer aussi un email et un prénom il a fallut étendre ce formulaire avec un formulaire personnalisé héritant de la classe standard.



The image shows a code editor with a project structure on the left and the content of forms.py on the right.

Project Structure (Left Panel):

- Project
 - main
 - P8venv
 - purbeurre
 - .git
 - app_products
 - app_users
 - __pycache__
 - migrations
 - static
 - templates
 - tests
 - __init__.py
 - admin.py
 - apps.py
 - forms.py**
 - models.py
 - urls.py
 - views.py
 - purbeurre
 - static
 - staticfiles
 - templates
 - webdrivers
 - .coverage
 - .env
 - .gitignore
 - geckodriver.log
 - manage.py
 - Procfile
 - README.md
 - requirements.txt

forms.py (Right Panel):

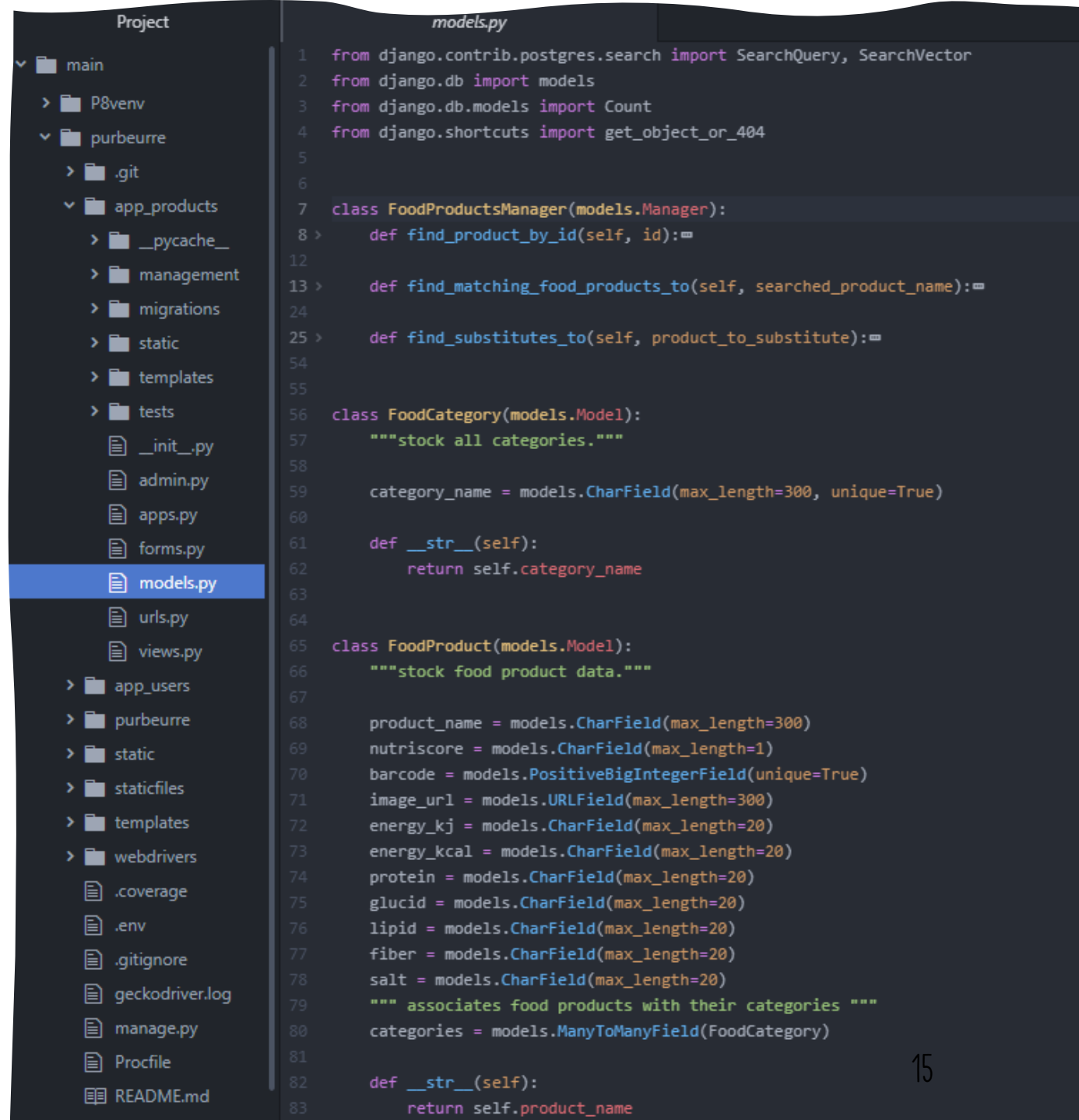
```
1 from django import forms
2 from django.contrib.auth import get_user_model
3 from django.contrib.auth.forms import UserCreationForm
4
5 User = get_user_model()
6
7
8 class PersonalUserCreationForm(UserCreationForm):
9     """addition of additional fields to the basic user profile."""
10
11     """ and the form-control class for bootstrap"""
12
13     class Meta:
14         model = User
15         fields = ['username', 'first_name', 'email', 'password1', 'password2']
16         widgets = {
17             'username': forms.TextInput(
18                 attrs={'placeholder': 'Votre nom d\'utilisateur'}
19             ),
20             'first_name': forms.TextInput(
21                 attrs={'placeholder': 'Votre Prénom'}
22             ),
23             'email': forms.EmailInput(
24                 attrs={'placeholder': 'Votre adresse email'}
25             ),
26         }
27
28 > def __init__(self, *args, **kwargs):=
34
```

4 - POINTS PARTICULIERS DE L'APPLICATION APP_PRODUCTS

MODÈLES

Uniquement 2 modèles les catégories d'aliments et les produits alimentaires avec leurs données.

L'association entre les deux tables est générée par django avec j'ajout du champ `ManyToManyField`



The image shows a code editor with a project file structure on the left and the `models.py` file on the right.

Project Structure:

- Project
 - main
 - P8venv
 - purbeurre
 - .git
 - app_products
 - __pycache__
 - management
 - migrations
 - static
 - templates
 - tests
 - __init__.py
 - admin.py
 - apps.py
 - forms.py
 - models.py
 - urls.py
 - views.py
 - app_users
 - purbeurre
 - static
 - staticfiles
 - templates
 - webdrivers
 - .coverage
 - .env
 - .gitignore
 - geckodriver.log
 - manage.py
 - Procfile
 - README.md

models.py:

```
1 from django.contrib.postgres.search import SearchQuery, SearchVector
2 from django.db import models
3 from django.db.models import Count
4 from django.shortcuts import get_object_or_404
5
6
7 class FoodProductsManager(models.Manager):
8     def find_product_by_id(self, id):
9
10
11
12
13     def find_matching_food_products_to(self, searched_product_name):
14
15
16
17
18     def find_substitutes_to(self, product_to_substitute):
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36 class FoodCategory(models.Model):
37     """stock all categories."""
38
39     category_name = models.CharField(max_length=300, unique=True)
40
41     def __str__(self):
42         return self.category_name
43
44
45
46
47
48
49
50
51
52
53
54
55
56 class FoodProduct(models.Model):
57     """stock food product data."""
58
59     product_name = models.CharField(max_length=300)
60     nutriscore = models.CharField(max_length=1)
61     barcode = models.PositiveBigIntegerField(unique=True)
62     image_url = models.URLField(max_length=300)
63     energy_kj = models.CharField(max_length=20)
64     energy_kcal = models.CharField(max_length=20)
65     protein = models.CharField(max_length=20)
66     glucid = models.CharField(max_length=20)
67     lipid = models.CharField(max_length=20)
68     fiber = models.CharField(max_length=20)
69     salt = models.CharField(max_length=20)
70     """ associates food products with their categories """
71     categories = models.ManyToManyField(FoodCategory)
72
73     def __str__(self):
74         return self.product_name
```

COMMANDES DJANGO-ADMIN PERSONNALISÉES

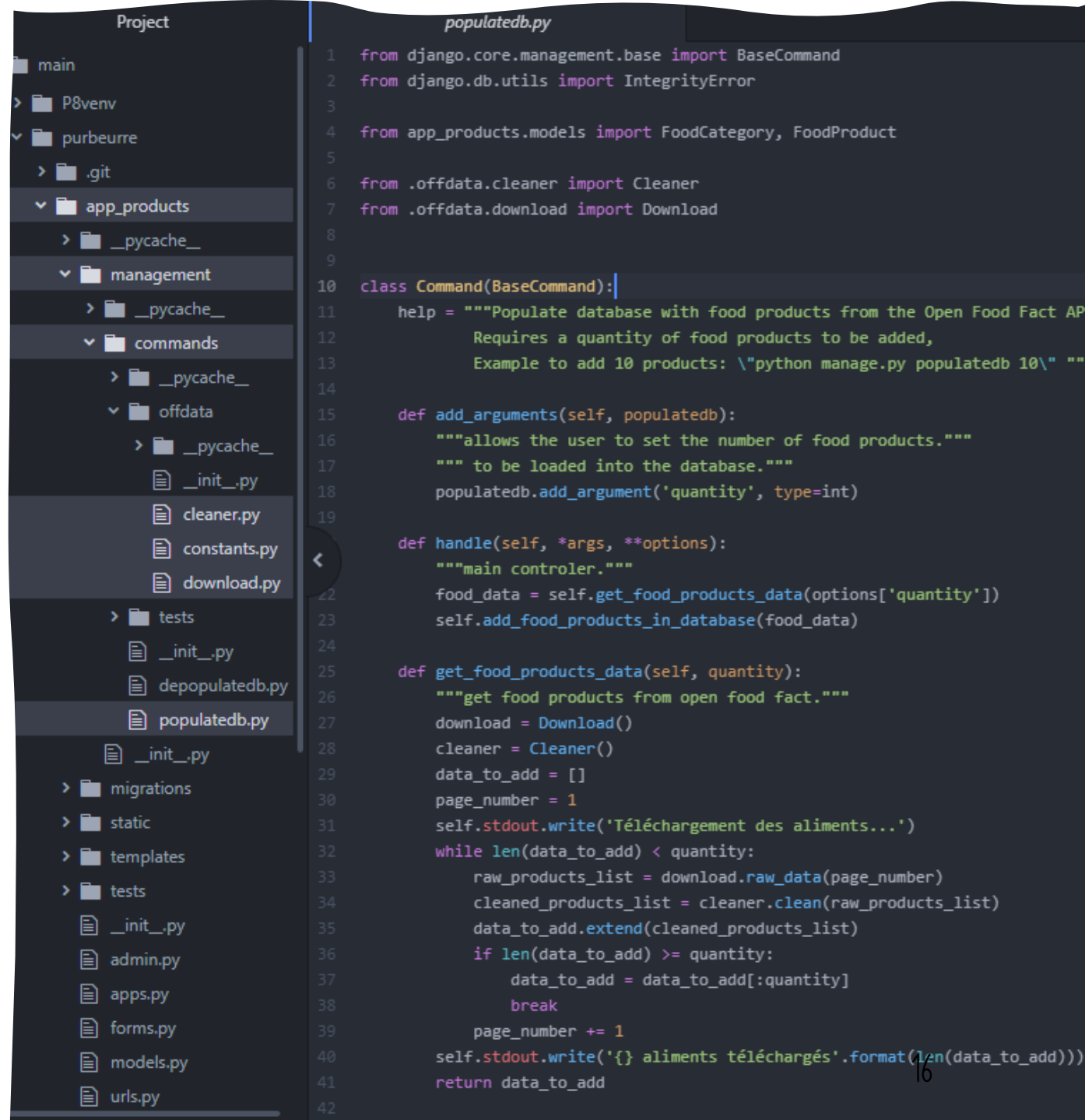
Le peuplement de la base de données des aliments passe par une commande django-admin personnalisée.

Lors du déploiement il suffit d'utiliser la commande :

manage.py populatedb 2000

Elle permet de spécifier le nombre de produits insérés, ce projet étant déployé sur heroku en version gratuite le nombre de lignes est limité.

Les données sont donc téléchargées depuis OpenFoodFact, nettoyées puis insérées.

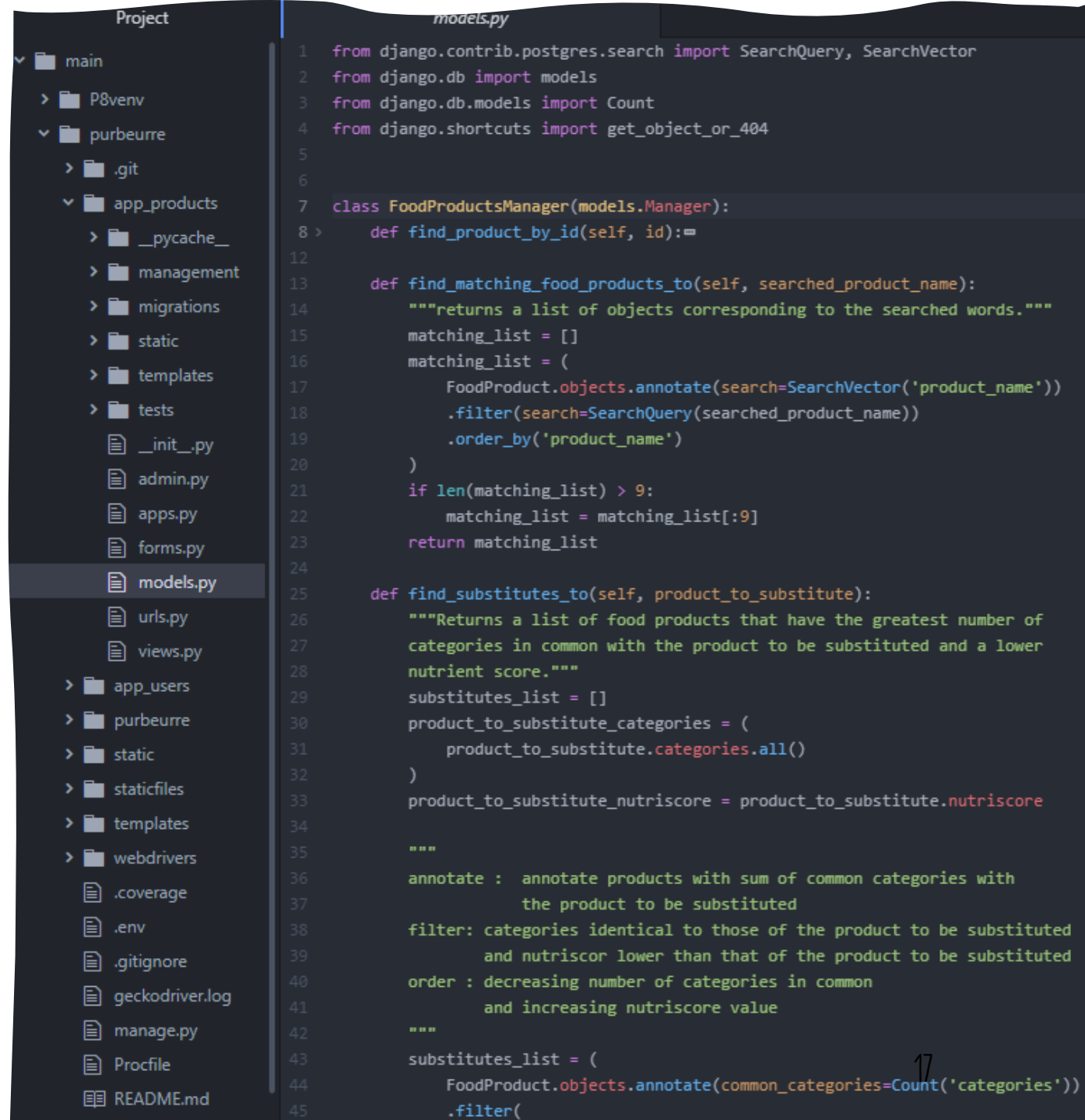


```
Project
├── main
├── P8venv
├── purbeurre
├── .git
├── app_products
├── _pycache_
├── management
├── _pycache_
├── commands
├── _pycache_
├── offdata
├── _pycache_
├── _init_.py
├── cleaner.py
├── constants.py
├── download.py
├── tests
├── _init_.py
├── depopulatedb.py
├── populatedb.py
├── _init_.py
├── migrations
├── static
├── templates
├── tests
├── _init_.py
├── admin.py
├── apps.py
├── forms.py
├── models.py
├── urls.py

populatedb.py
1 from django.core.management.base import BaseCommand
2 from django.db.utils import IntegrityError
3
4 from app_products.models import FoodCategory, FoodProduct
5
6 from .offdata.cleaner import Cleaner
7 from .offdata.download import Download
8
9
10 class Command(BaseCommand):
11     help = """Populate database with food products from the Open Food Fact AP
12         Requires a quantity of food products to be added,
13         Example to add 10 products: \"python manage.py populatedb 10\" """
14
15     def add_arguments(self, populatedb):
16         """allows the user to set the number of food products."""
17         """ to be loaded into the database."""
18         populatedb.add_argument('quantity', type=int)
19
20     def handle(self, *args, **options):
21         """main controller."""
22         food_data = self.get_food_products_data(options['quantity'])
23         self.add_food_products_in_database(food_data)
24
25     def get_food_products_data(self, quantity):
26         """get food products from open food fact."""
27         download = Download()
28         cleaner = Cleaner()
29         data_to_add = []
30         page_number = 1
31         self.stdout.write('Téléchargement des aliments...')
32         while len(data_to_add) < quantity:
33             raw_products_list = download.raw_data(page_number)
34             cleaned_products_list = cleaner.clean(raw_products_list)
35             data_to_add.extend(cleaned_products_list)
36             if len(data_to_add) >= quantity:
37                 data_to_add = data_to_add[:quantity]
38                 break
39             page_number += 1
40         self.stdout.write('{} aliments téléchargés'.format(len(data_to_add)))
41         return data_to_add
42
```


GESTIONNAIRES PERSONNALISÉS

Ajout d'un gestionnaire personnalisé pour que tous les appels en base soient bien effectués dans le modèle et non dans les vues.



The image shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'main' folder containing 'P8venv', 'purbeurre', and 'app_products'. The 'app_products' folder contains '.__pycache__', 'management', 'migrations', 'static', 'templates', 'tests', '.__init__.py', 'admin.py', 'apps.py', 'forms.py', 'models.py' (selected), 'urls.py', and 'views.py'. The code editor shows the 'models.py' file with the following code:

```
1 from django.contrib.postgres.search import SearchQuery, SearchVector
2 from django.db import models
3 from django.db.models import Count
4 from django.shortcuts import get_object_or_404
5
6
7 class FoodProductsManager(models.Manager):
8     def find_product_by_id(self, id):
9
10
11
12     def find_matching_food_products_to(self, searched_product_name):
13         """returns a list of objects corresponding to the searched words."""
14         matching_list = []
15         matching_list = (
16             FoodProduct.objects.annotate(search=SearchVector('product_name'))
17             .filter(search=SearchQuery(searched_product_name))
18             .order_by('product_name')
19         )
20         if len(matching_list) > 9:
21             matching_list = matching_list[:9]
22         return matching_list
23
24     def find_substitutes_to(self, product_to_substitute):
25         """Returns a list of food products that have the greatest number of
26         categories in common with the product to be substituted and a lower
27         nutrient score."""
28         substitutes_list = []
29         product_to_substitute_categories = (
30             product_to_substitute.categories.all()
31         )
32         product_to_substitute_nutriscore = product_to_substitute.nutriscore
33
34         """
35         annotate : annotate products with sum of common categories with
36                     the product to be substituted
37         filter: categories identical to those of the product to be substituted
38                 and nutriscore lower than that of the product to be substituted
39         order : decreasing number of categories in common
40                 and increasing nutriscore value
41         """
42         substitutes_list = (
43             FoodProduct.objects.annotate(common_categories=Count('categories'))
44             .filter(
45
```

LA RECHERCHE DE SUBSTITUTS

La sélection des aliments de substitution passe par un enchainement de filtres et utilise `annotate()`.

Les filtres sélectionnes déjà les aliments qui ont des catégories présentes dans la liste des catégories de l'aliment à substituer (`__in`) et qui ont un nutri-score inférieur (`__lt`), puis ces produits sont annotés avec la quantité de catégories qu'ils ont en commun avec le produit à substituer.

Le QuerySet est finalement trié et retourne en premier les aliments qui ont le plus de catégories en commun puis les nutri-score les plus faibles.

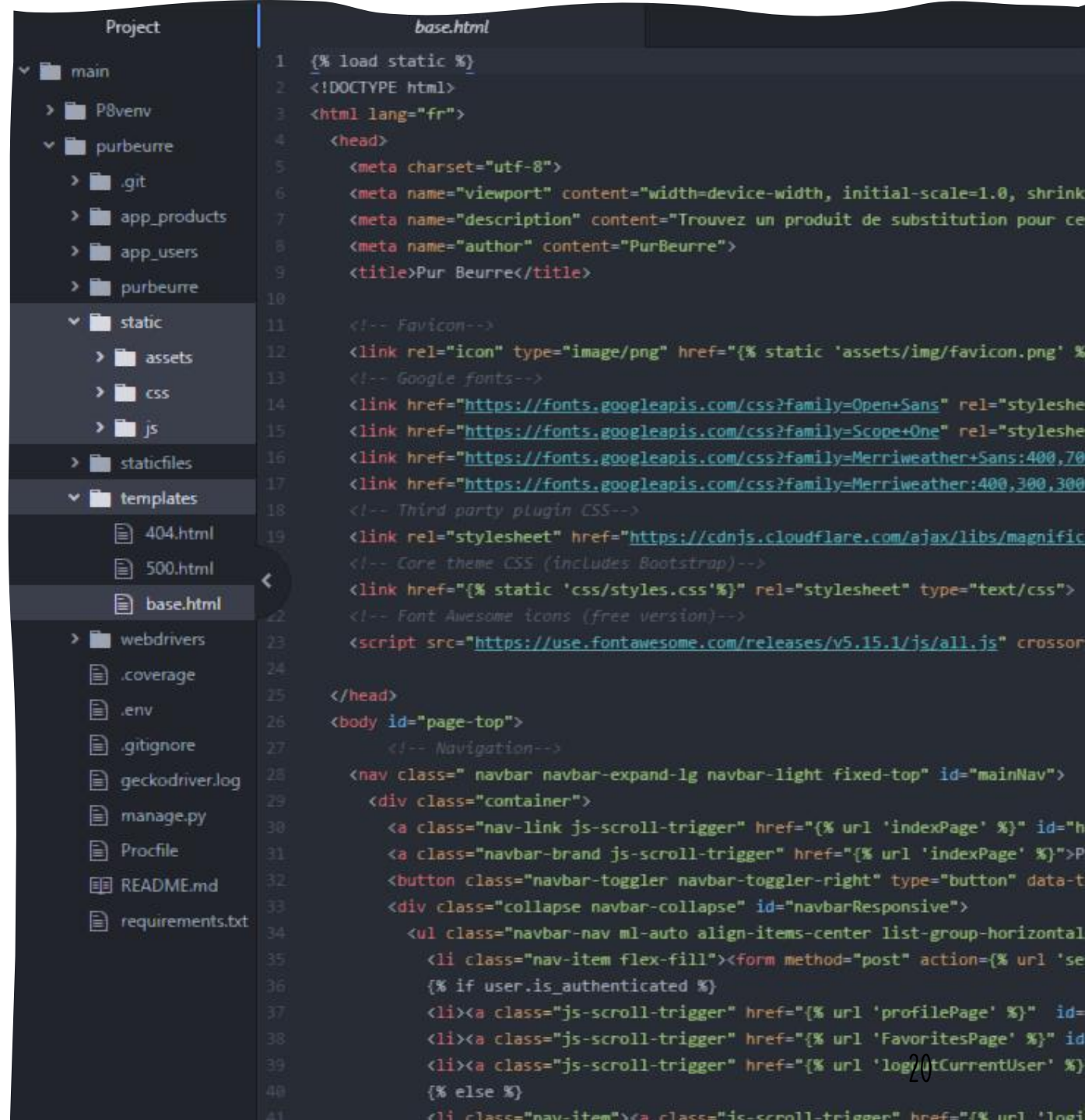
```
Project
├── main
│   ├── P8venv
│   ├── purbeurre
│   │   ├── .git
│   │   └── app_products
│   │       ├── __pycache__
│   │       ├── management
│   │       ├── migrations
│   │       ├── static
│   │       ├── templates
│   │       ├── tests
│   │       ├── __init__.py
│   │       ├── admin.py
│   │       ├── apps.py
│   │       ├── forms.py
│   │       └── models.py
│   ├── urls.py
│   └── views.py
├── app_users
├── purbeurre
├── static
├── staticfiles
├── templates
├── webdrivers
├── .coverage
├── .env
├── .gitignore
├── geckodriver.log
├── manage.py
├── Procfile
└── README.md

models.py
1  from django.contrib.postgres.search import SearchQuery, SearchVector
2  from django.db import models
3  from django.db.models import Count
4  from django.shortcuts import get_object_or_404
5
6
7  class FoodProductsManager(models.Manager):
8  >   def find_product_by_id(self, id):=
12
13 >   def find_matching_food_products_to(self, searched_product_name):=
24
25   def find_substitutes_to(self, product_to_substitute):
26       """Returns a list of food products that have the greatest number of
27       categories in common with the product to be substituted and a lower
28       nutrient score."""
29       substitutes_list = []
30       product_to_substitute_categories = (
31           product_to_substitute.categories.all()
32       )
33       product_to_substitute_nutriscore = product_to_substitute.nutriscore
34
35       """
36       annotate :   annotate products with sum of common categories with
37                   the product to be substituted
38       filter: categories identical to those of the product to be substituted
39               and nutriscore lower than that of the product to be substituted
40       order : decreasing number of categories in common
41               and increasing nutriscore value
42       """
43       substitutes_list = (
44           FoodProduct.objects.annotate(common_categories=Count('categories'))
45           .filter(
46               categories__in=product_to_substitute_categories,
47               nutriscore__lt=product_to_substitute_nutriscore,
48           )
49           .order_by('-common_categories', 'nutriscore')
50       )
51       if len(substitutes_list) > 8:
52           substitutes_list = substitutes_list[:9]
53       return substitutes_list
54
55
```

5 - FRONT-END

INTÉGRATION DU THÈME BOOTSTRAP

Le thème désiré par le client à été inséré et un template de base, commun à toutes les pages web a été ajouté pour ne pas répéter des lignes inutilement.



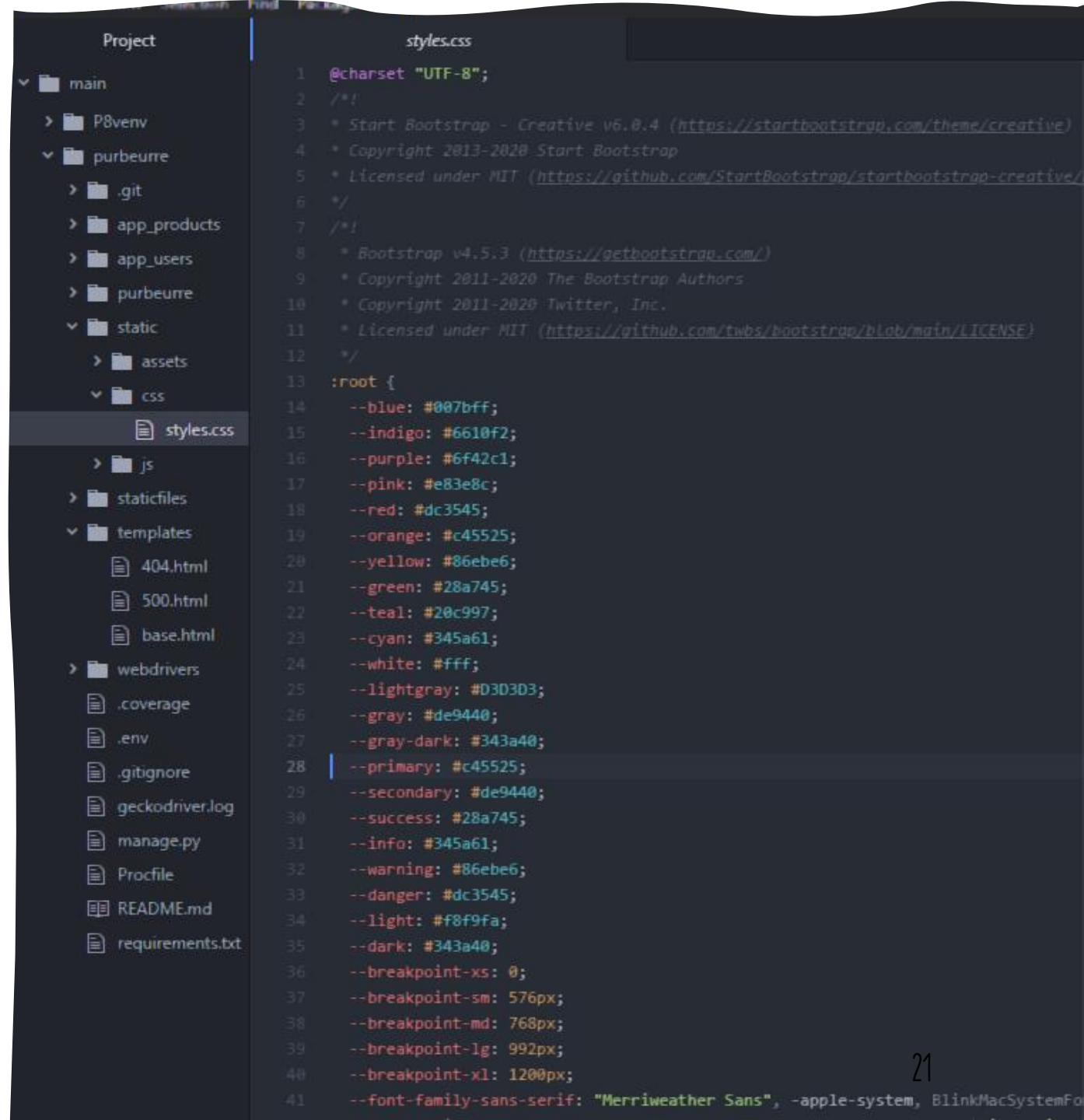
```
Project
├── main
│   ├── P8venv
│   ├── purbeurre
│   │   ├── .git
│   │   ├── app_products
│   │   ├── app_users
│   │   └── purbeurre
│   ├── static
│   │   ├── assets
│   │   ├── css
│   │   └── js
│   ├── staticfiles
│   ├── templates
│   │   ├── 404.html
│   │   ├── 500.html
│   │   └── base.html
│   ├── webdrivers
│   ├── .coverage
│   ├── .env
│   ├── .gitignore
│   ├── geckodriver.log
│   ├── manage.py
│   ├── Procfile
│   ├── README.md
│   └── requirements.txt
└── ...

base.html
1  {% load static %}
2  <!DOCTYPE html>
3  <html lang="fr">
4      <head>
5          <meta charset="utf-8">
6          <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">
7          <meta name="description" content="Trouvez un produit de substitution pour ce produit.">
8          <meta name="author" content="PurBeurre">
9          <title>Pur Beurre</title>
10
11      <!-- Favicon -->
12      <link rel="icon" type="image/png" href="{% static 'assets/img/favicon.png' %}">
13
14      <!-- Google fonts -->
15      <link href="https://fonts.googleapis.com/css?family=Open+Sans" rel="stylesheet">
16      <link href="https://fonts.googleapis.com/css?family=Scope+One" rel="stylesheet">
17      <link href="https://fonts.googleapis.com/css?family=Merriweather+Sans:400,700" rel="stylesheet">
18      <link href="https://fonts.googleapis.com/css?family=Merriweather:400,300,300italic" rel="stylesheet">
19
20      <!-- Third party plugin CSS -->
21      <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/magnific-popup.js/1.1.0/magnific-popup.css">
22
23      <!-- Core theme CSS (includes Bootstrap) -->
24      <link href="{% static 'css/styles.css' %}" rel="stylesheet" type="text/css">
25
26      <!-- Font Awesome icons (free version) -->
27      <script src="https://use.fontawesome.com/releases/v5.15.1/js/all.js" crossorigin="anonymous"></script>
28
29  </head>
30  <body id="page-top">
31      <!-- Navigation -->
32      <nav class="navbar navbar-expand-lg navbar-light fixed-top" id="mainNav">
33          <div class="container">
34              <a class="nav-link js-scroll-trigger" href="{% url 'indexPage' %}" id="home">Accueil</a>
35              <a class="nav-link js-scroll-trigger" href="{% url 'indexPage' %}" id="home">Pur Beurre</a>
36              <button class="navbar-toggler navbar-toggler-right" type="button" data-toggle="collapse" data-target="#navbarResponsive">
37                  <span>☰</span>
38              </button>
39              <div class="collapse navbar-collapse" id="navbarResponsive">
40                  <ul class="navbar-nav ml-auto align-items-center list-group-horizontal">
41                      <li class="nav-item flex-fill"><form method="post" action="{% url 'search' %}">
42                          <input type="text" value="Rechercher">
43                      </li>
44                      <li class="nav-item"><a class="js-scroll-trigger" href="{% url 'profilePage' %}" id="profile">Profil</a>
45                      <li class="nav-item"><a class="js-scroll-trigger" href="{% url 'FavoritesPage' %}" id="favorites">Favoris</a>
46                      <li class="nav-item"><a class="js-scroll-trigger" href="{% url 'logoutCurrentUser' %}" id="logout">Déconnexion</a>
47                      <li class="nav-item"><a class="js-scroll-trigger" href="{% url 'login' %}" id="login">Connexion</a>
48                  </ul>
49              </div>
50          </div>
51      </nav>
```

MODIFICATION DES FEUILLES DE STYLE EN CASCADE (.CSS)

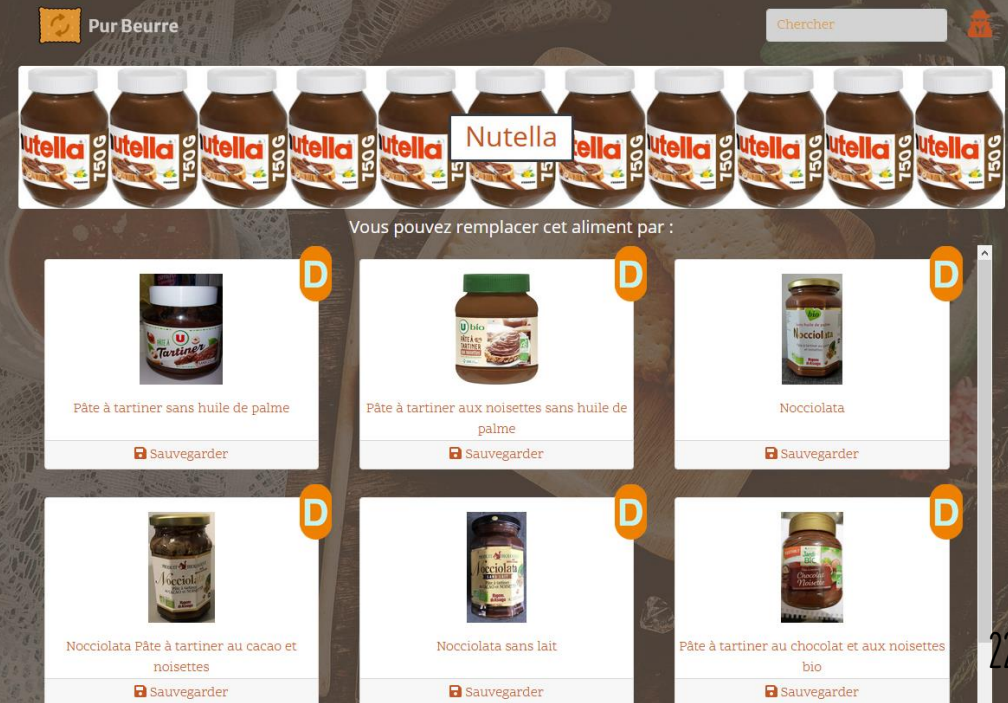
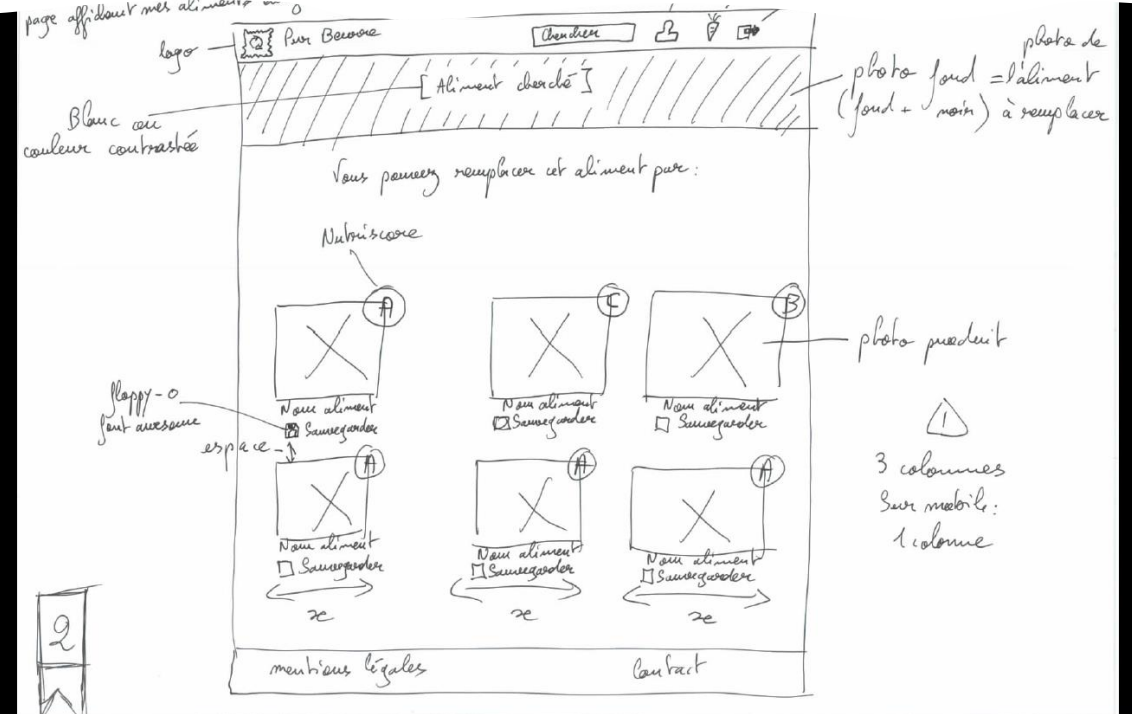
Le fichiers css du thème étant déjà très conséquent il n'a pas été nécessaire d'ajouter beaucoup d'éléments.

Pour respecter la chartre graphique, j'ai en majorité modifié les couleurs utilisées, les police de caractère et deux classes .



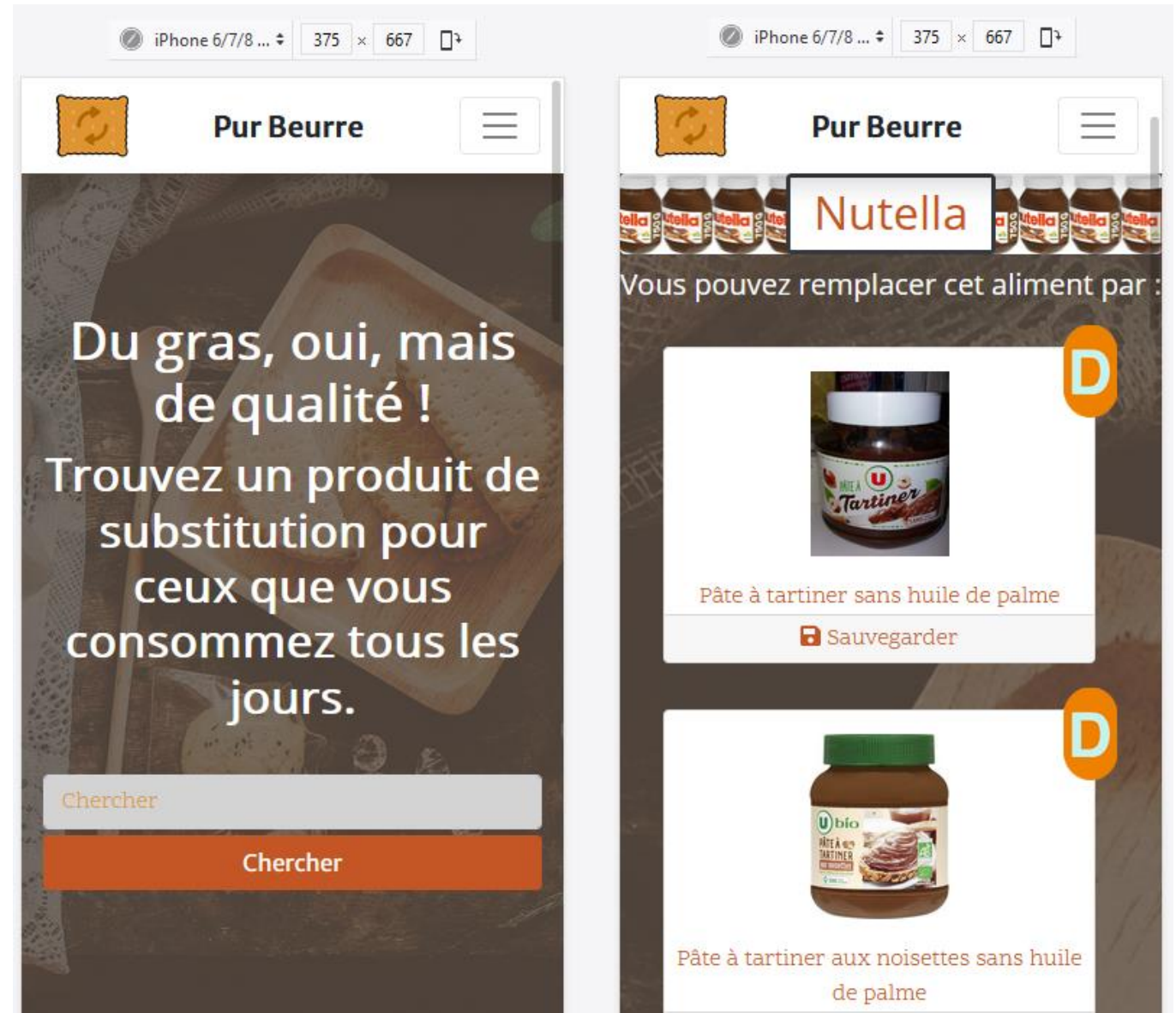
APPARENCE DES PAGES WEB CONFORMES AUX ESQUISSES

Les pages web ont été
construites pour correspondre à
la charte graphique et aux
attentes clients.



UN SITE RESPONSIVE

Bootstrap a permis de rendre le site responsive



PAGE MENTIONS LÉGALES

J'ai utilisé le site
<https://www.economie.gouv.fr>
pour lister les éléments
obligatoires à ajouter



Pur Beurre

Edition

Le présent site est la propriété de la Startup Pur Beurre:

Pur Beurre
Montmartre
75018 Paris
01 42 92 81 00

Capital social de 1,00 EURO
RCS 443061841

Pour tout renseignement relatif au site Pur Beurre, vous pouvez envoyer un courriel à l'adresse suivante :

lebeurre@largentdubeurre.fr

Crédits

Template : [StartBootstrap](#)

Image d'arrière-plan : [Kawin Harasai](#)

Icones : [Font Awesome](#)

Hébergement : [Heroku](#)

Données sur les produits alimentaires : [Open Food Facts](#)

Développeur : Emmanuel OUDOT

6 - TESTS

PLAN DE TEST

Plan de test

Définition du périmètre des tests :

-Tests unitaires :

app_products :
management/commands/offdata/cleaner.py
 Toutes les méthodes de la classe Cleaner
views.py
 Test des méthodes : index, search et substitutes

-Tests d'intégrations :

app_products :
 Toutes les vues et les urls

app_users :
 Toutes les vues et les urls

-Tests fonctionnels :

app_products :
 Test l'affichage de la page d'index
 Test si la recherche retourne un résultat
 Test si la recherche de substitut retourne un résultat

app_users :
 Test si l'utilisateur peut se connecter et se déconnecter
 Test si l'utilisateur peut afficher son profil
 Test si l'utilisateur peut créer un compte

Limites du périmètre de test :

Ne sera pas testé :

- Tests de performances
- Tests de charge
- Tests de sécurité

Outils de tests :

- unittest et django.test
- unittest.mock pour les mocks et patches
- selenium pour les tests fonctionnels
- coverage pour mesurer la couverture

Organisation des tests :

```
app_products/  
  __init__.py  
  management/  
    __init__.py  
    commands/  
      __init__.py  
      tests  
        __init__.py  
        test_cleaner.py  
  
tests/  
  __init__.py  
  fonctionnal/  
    __init__.py  
    test_firefox.py  
  integration/  
    __init__.py  
    test_models.py  
    test_urls.py  
    test_views.py  
  unit/  
    __init__.py  
    test_views.py  
  
app_users/  
  __init__.py  
  tests/  
    __init__.py  
    fonctionnal/  
      __init__.py  
      test_firefox.py  
    integration/  
      __init__.py  
      test_urls.py  
      test_views.py
```

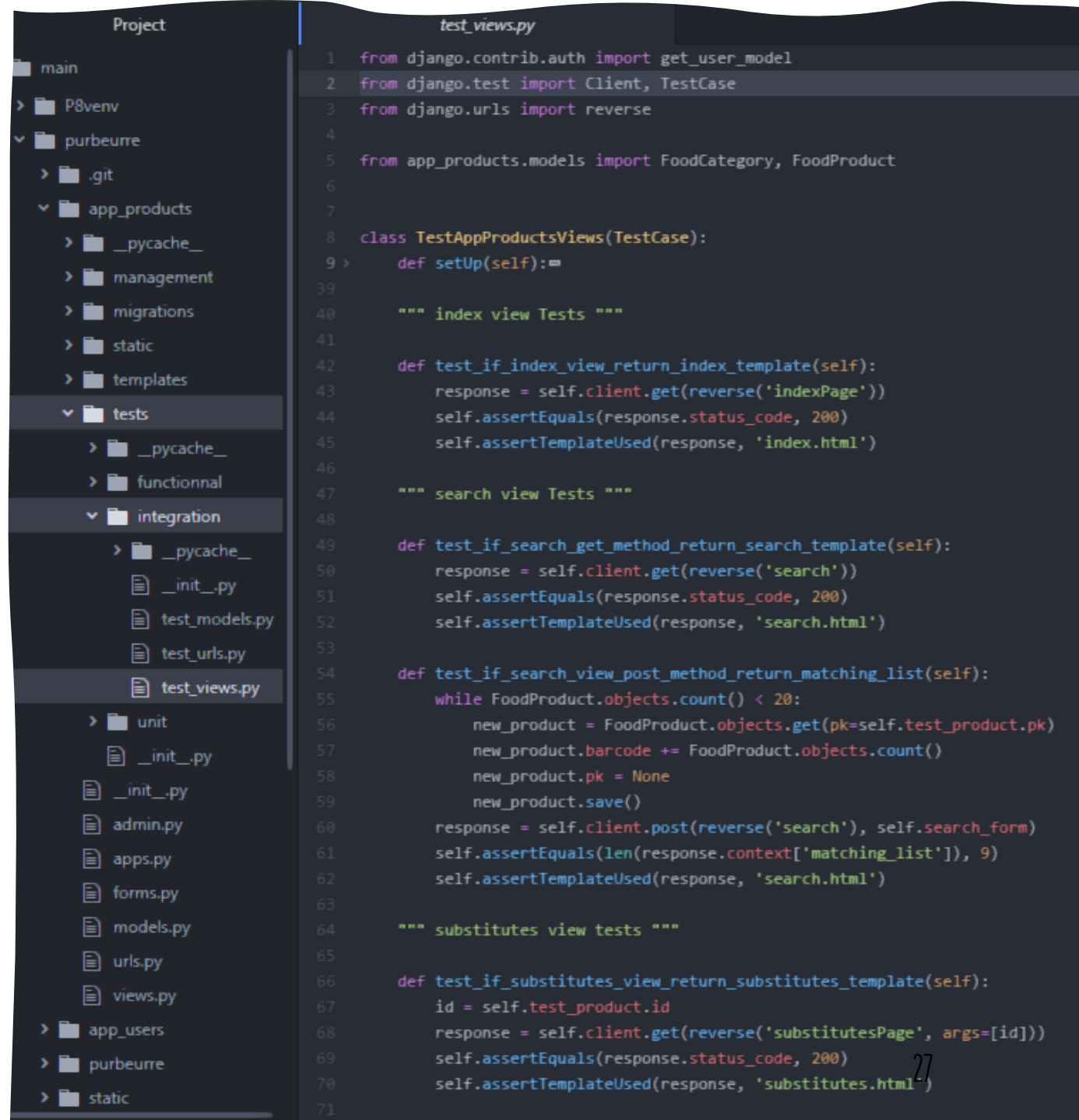
Exécution des tests :

Les tests seront écrits et exécutés après chaque nouvelle fonctionnalité ajoutée.
Puis exécuté à nouveau en cas de modification et avant déploiement.

1. Définition du périmètre des tests
2. Limites du périmètre de test
3. Outils de tests
4. Organisation des tests
5. Exécution des tests

TESTS D'INTÉGRATION

Les tests d'intégrations ont été ajoutés au fur et à mesure de l'ajout des fonctionnalités et couvrent une grande partie du code et utilisent le module unittest avec django.test



The image shows a code editor with a project structure on the left and a test file on the right.

Project Structure (Left Panel):

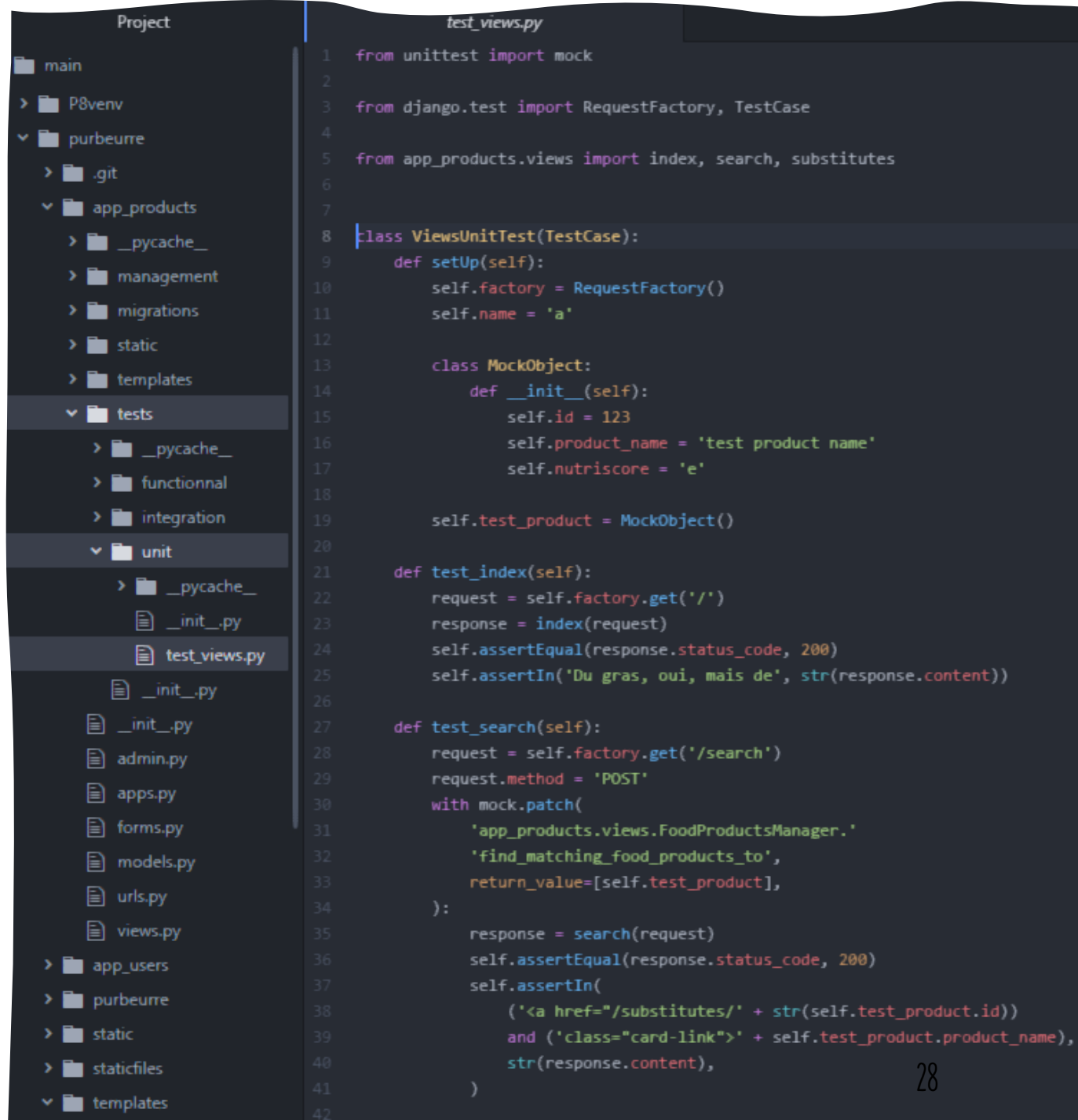
- Project
 - main
 - P8venv
 - purbeurre
 - .git
 - app_products
 - __pycache__
 - management
 - migrations
 - static
 - templates
 - tests
 - __pycache__
 - functionnal
 - integration
 - __pycache__
 - __init__.py
 - test_models.py
 - test_urls.py
 - test_views.py
 - unit
 - __init__.py

test_views.py (Right Panel):

```
1 from django.contrib.auth import get_user_model
2 from django.test import Client, TestCase
3 from django.urls import reverse
4
5 from app_products.models import FoodCategory, FoodProduct
6
7
8 class TestAppProductsViews(TestCase):
9     def setUp(self):
10
11
12     """ index view Tests """
13
14     def test_if_index_view_return_index_template(self):
15         response = self.client.get(reverse('indexPage'))
16         self.assertEqual(response.status_code, 200)
17         self.assertTemplateUsed(response, 'index.html')
18
19     """ search view Tests """
20
21     def test_if_search_get_method_return_search_template(self):
22         response = self.client.get(reverse('search'))
23         self.assertEqual(response.status_code, 200)
24         self.assertTemplateUsed(response, 'search.html')
25
26     def test_if_search_view_post_method_return_matching_list(self):
27         while FoodProduct.objects.count() < 20:
28             new_product = FoodProduct.objects.get(pk=self.test_product.pk)
29             new_product.barcode += FoodProduct.objects.count()
30             new_product.pk = None
31             new_product.save()
32         response = self.client.post(reverse('search'), self.search_form)
33         self.assertEqual(len(response.context['matching_list']), 9)
34         self.assertTemplateUsed(response, 'search.html')
35
36     """ substitutes view tests """
37
38     def test_if_substitutes_view_return_substitutes_template(self):
39         id = self.test_product.id
40         response = self.client.get(reverse('substitutesPage', args=[id]))
41         self.assertEqual(response.status_code, 200)
42         self.assertTemplateUsed(response, 'substitutes.html')
```

TESTS UNITAIRES

Les tests unitaires quand à eux ont été ajoutés pour tester des parties plus spécifiques comme par exemple les recherches de substituts ou les méthodes de Cleaner. J'ai utilisé unittest.mock.patch pour patcher.

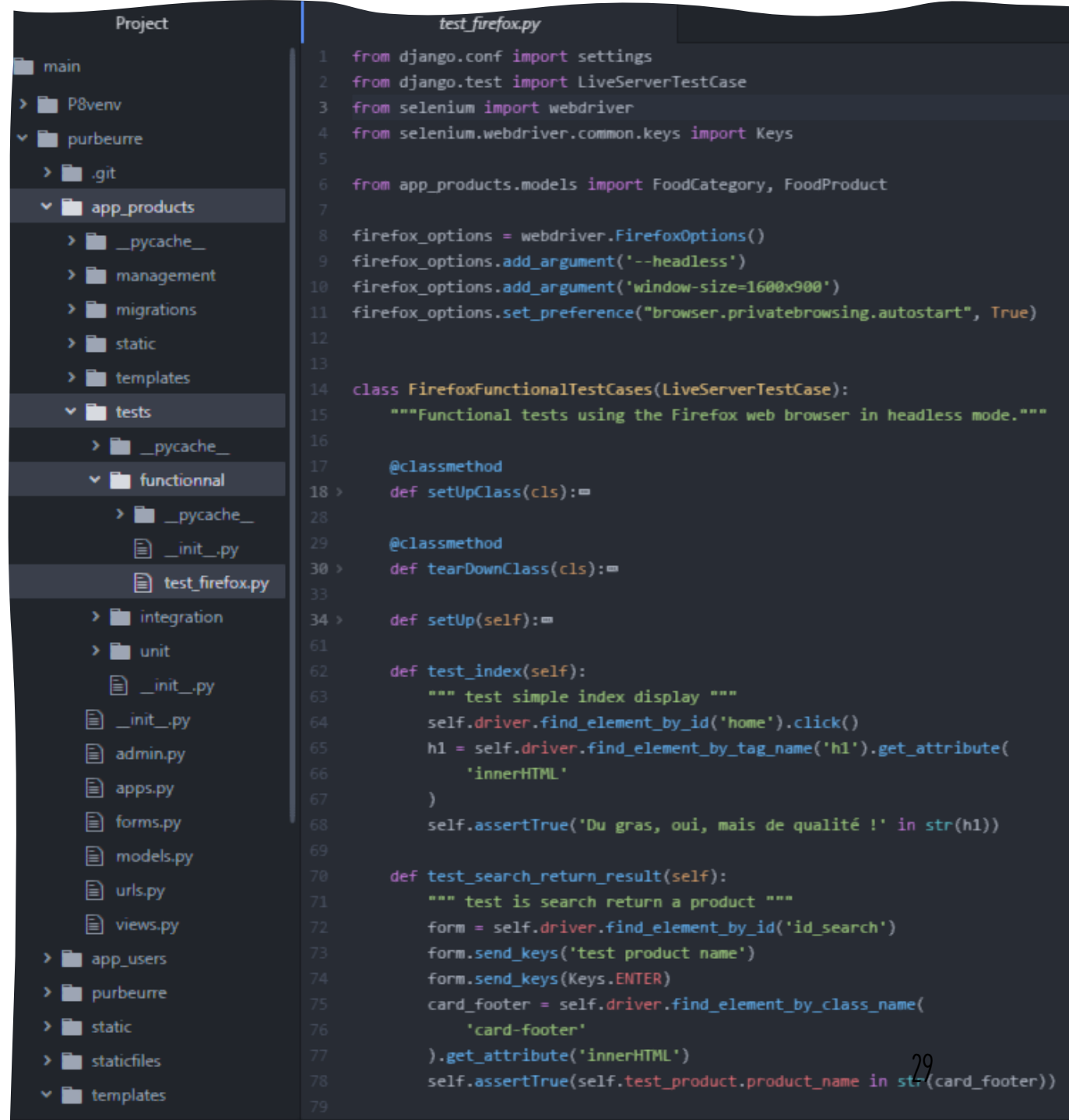


```
Project
├── main
├── P8venv
├── purbeurre
├── .git
├── app_products
│   ├── __pycache__
│   ├── management
│   ├── migrations
│   ├── static
│   └── templates
├── tests
│   ├── __pycache__
│   ├── fonctionnal
│   ├── integration
│   └── unit
│       ├── __pycache__
│       ├── __init__.py
│       └── test_views.py
│           ├── __init__.py
│           ├── admin.py
│           ├── apps.py
│           ├── forms.py
│           ├── models.py
│           ├── urls.py
│           └── views.py
├── app_users
├── purbeurre
├── static
├── staticfiles
└── templates

test_views.py
1  from unittest import mock
2
3  from django.test import RequestFactory, TestCase
4
5  from app_products.views import index, search, substitutes
6
7
8  class ViewsUnitTest(TestCase):
9      def setUp(self):
10         self.factory = RequestFactory()
11         self.name = 'a'
12
13         class MockObject:
14             def __init__(self):
15                 self.id = 123
16                 self.product_name = 'test product name'
17                 self.nutriscore = 'e'
18
19         self.test_product = MockObject()
20
21     def test_index(self):
22         request = self.factory.get('/')
23         response = index(request)
24         self.assertEqual(response.status_code, 200)
25         self.assertIn('Du gras, oui, mais de', str(response.content))
26
27     def test_search(self):
28         request = self.factory.get('/search')
29         request.method = 'POST'
30         with mock.patch(
31             'app_products.views.FoodProductsManager.'
32             'find_matching_food_products_to',
33             return_value=[self.test_product],
34         ):
35             response = search(request)
36             self.assertEqual(response.status_code, 200)
37             self.assertIn(
38                 ('<a href="/substitutes/' + str(self.test_product.id))
39                 and ('class="card-link">' + self.test_product.product_name),
40                 str(response.content),
41             )
42
```

TESTS FONCTIONNELS

Les tests fonctionnels ont été réalisés à l'aide de selenium.webdriver pour le navigateur Firefox.



```
Project
├── main
├── P8venv
├── purbeurre
├── .git
├── app_products
│   ├── __pycache__
│   ├── management
│   ├── migrations
│   ├── static
│   ├── templates
│   └── tests
│       ├── __pycache__
│       ├── fonctionnal
│       │   ├── __pycache__
│       │   ├── __init__.py
│       │   └── test_firefox.py
│       ├── integration
│       ├── unit
│       │   ├── __init__.py
│       │   ├── admin.py
│       │   ├── apps.py
│       │   ├── forms.py
│       │   ├── models.py
│       │   ├── urls.py
│       │   └── views.py
│       ├── app_users
│       ├── purbeurre
│       ├── static
│       ├── staticfiles
│       └── templates
└── test_firefox.py

test_firefox.py
1  from django.conf import settings
2  from django.test import LiveServerTestCase
3  from selenium import webdriver
4  from selenium.webdriver.common.keys import Keys
5
6  from app_products.models import FoodCategory, FoodProduct
7
8  firefox_options = webdriver.FirefoxOptions()
9  firefox_options.add_argument('--headless')
10 firefox_options.add_argument('window-size=1600x900')
11 firefox_options.set_preference("browser.privatebrowsing.autostart", True)
12
13
14 class FirefoxFunctionalTestCases(LiveServerTestCase):
15     """Functional tests using the Firefox web browser in headless mode."""
16
17     @classmethod
18     def setUpClass(cls):
19
20
21     @classmethod
22     def tearDownClass(cls):
23
24
25     def setUp(self):
26
27
28     def test_index(self):
29         """ test simple index display """
30         self.driver.find_element_by_id('home').click()
31         h1 = self.driver.find_element_by_tag_name('h1').get_attribute(
32             'innerHTML'
33         )
34         self.assertTrue('Du gras, oui, mais de qualité !' in str(h1))
35
36
37     def test_search_return_result(self):
38         """ test is search return a product """
39         form = self.driver.find_element_by_id('id_search')
40         form.send_keys('test product name')
41         form.send_keys(Keys.ENTER)
42         card_footer = self.driver.find_element_by_class_name(
43             'card-footer'
44         ).get_attribute('innerHTML')
45         self.assertTrue(self.test_product.product_name in str(card_footer))
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
```

RAPPORT DE L'EXÉCUTION DES TESTS

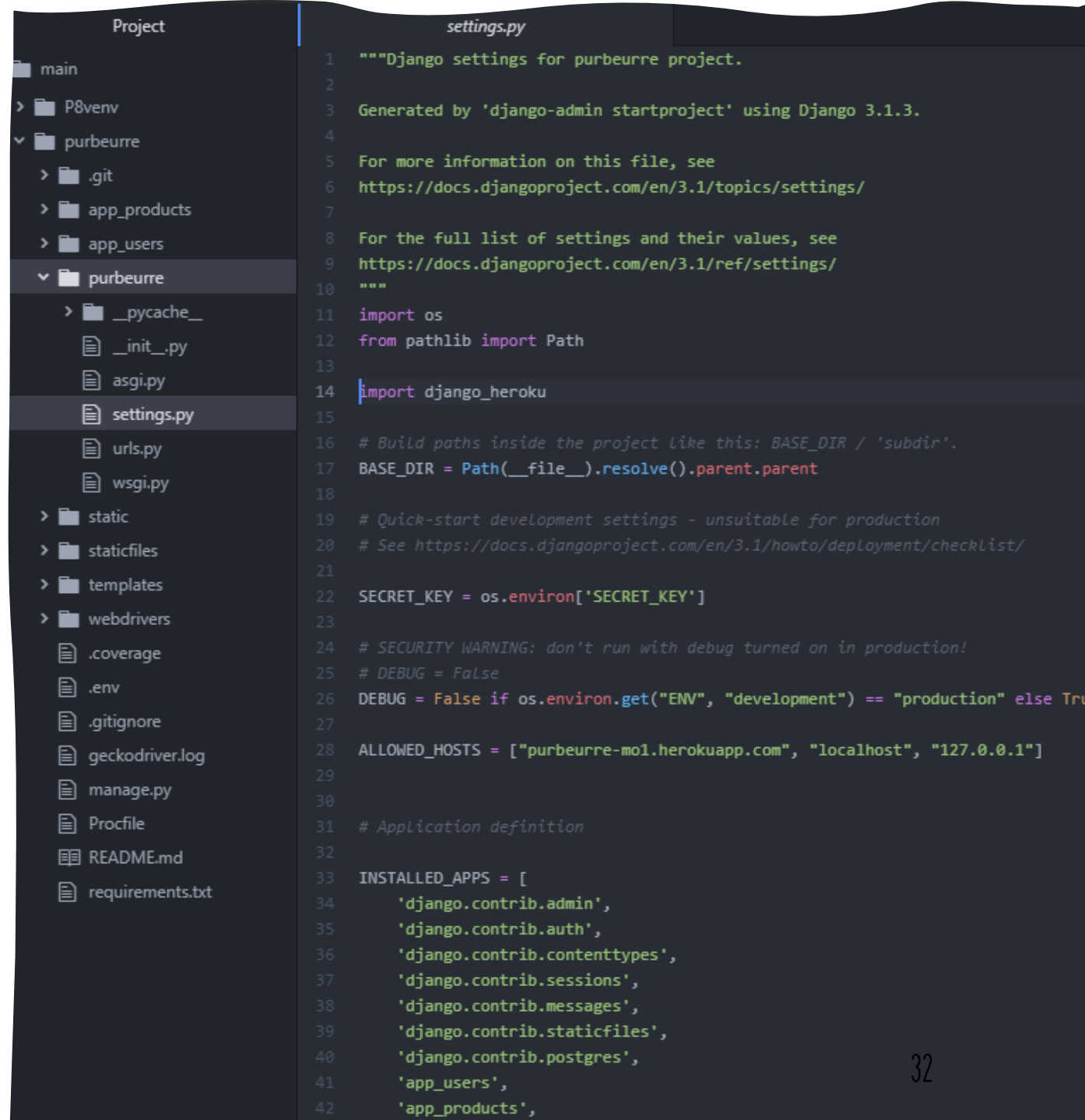
Enfin j'ai utiliser coverage pour avoir un rapport de l'exécution des tests et connaitre le taux de couverture, donc 87% pour ce projet.

| | Miss | Cover |
|--|------|-------|
| app_products__init__.py | 0 | 100% |
| app_products\admin.py | 0 | 100% |
| app_products\apps.py | 3 | 0% |
| app_products\forms.py | 14 | 100% |
| app_products\management__init__.py | 0 | 100% |
| app_products\management\commands__init__.py | 0 | 100% |
| app_products\management\commands\depopulatedb.py | 10 | 0% |
| app_products\management\commands\offdata__init__.py | 0 | 100% |
| app_products\management\commands\offdata\cleaner.py | 52 | 94% |
| app_products\management\commands\offdata\constants.py | 6 | 100% |
| app_products\management\commands\offdata\download.py | 23 | 0% |
| app_products\management\commands\populatedb.py | 49 | 0% |
| app_products\management\commands\tests__init__.py | 0 | 100% |
| app_products\management\commands\tests\constants_test.py | 3 | 100% |
| app_products\management\commands\tests\test_cleaner.py | 43 | 100% |
| app_products\migrations\0001_initial.py | 5 | 100% |
| app_products\migrations__init__.py | 0 | 100% |
| app_products\models.py | 41 | 100% |
| app_products\tests__init__.py | 0 | 100% |
| app_products\tests\functionnal__init__.py | 0 | 100% |
| app_products\tests\functionnal\test_firefox.py | 48 | 100% |
| app_products\tests\integration__init__.py | 0 | 100% |
| app_products\tests\integration\test_models.py | 10 | 100% |
| app_products\tests\integration\test_urls.py | 25 | 100% |
| app_products\tests\integration\test_views.py | 101 | 100% |
| app_products\tests\unit__init__.py | 0 | 100% |
| app_products\tests\unit\test_views.py | 33 | 100% |
| app_products\urls.py | 3 | 100% |
| app_products\views.py | 43 | 100% |
| app_users__init__.py | 0 | 100% |
| app_users\admin.py | 4 | 100% |
| app_users\apps.py | 3 | 0% |
| app_users\forms.py | 15 | 100% |
| app_users\migrations\0001_initial.py | 8 | 100% |
| app_users\migrations\0002_user_favorites.py | 4 | 100% |
| app_users\migrations__init__.py | 0 | 100% |
| app_users\models.py | 11 | 100% |
| app_users\tests__init__.py | 0 | 100% |
| app_users\tests\functionnal__init__.py | 0 | 100% |
| app_users\tests\functionnal\test_firefox.py | 51 | 100% |
| app_users\tests\integration__init__.py | 0 | 100% |
| app_users\tests\integration\test_urls.py | 16 | 100% |
| app_users\tests\integration\test_views.py | 61 | 100% |
| app_users\urls.py | 3 | 100% |
| app_users\views.py | 42 | 100% |
| manage.py | 12 | 83% |
| urbeurre__init__.py | 4 | 100% |
| urbeurre\asgi.py | 4 | 0% |
| urbeurre\settings.py | 28 | 100% |
| urbeurre\urls.py | 3 | 100% |
| urbeurre\wsgi.py | 4 | 0% |
| ebdrivers__init__.py | 0 | 100% |
| TOTAL | 785 | 87% |

7 - DÉPLOIEMENT

DJANGO ET HEROKU

- Heroku et django simplifient le déploiement avec par exemple:
- L'installation automatisé de la base de données avec l'add-on postgres
- Le paquet django_heroku qui configure le projet pour être déployé sur heroku
- Le push avec git pour charger les fichiers sur Heroku
- Et enfin les commandes django pour créer les tables, peupler la base de donnée etc...



The image shows a code editor with a dark theme. On the left, a file explorer pane displays the project structure. The 'purbeurre' directory is expanded, showing files like __pycache__, __init__.py, asgi.py, settings.py (highlighted), urls.py, wsgi.py, static, staticfiles, templates, webdrivers, .coverage, .env, .gitignore, geckodriver.log, manage.py, Procfile, README.md, and requirements.txt. The main editor area shows the content of settings.py, which includes project configuration, imports, and application settings.

```
settings.py
1  """Django settings for purbeurre project.
2
3  Generated by 'django-admin startproject' using Django 3.1.3.
4
5  For more information on this file, see
6  https://docs.djangoproject.com/en/3.1/topics/settings/
7
8  For the full list of settings and their values, see
9  https://docs.djangoproject.com/en/3.1/ref/settings/
10 """
11 import os
12 from pathlib import Path
13
14 import django_heroku
15
16 # Build paths inside the project like this: BASE_DIR / 'subdir'.
17 BASE_DIR = Path(__file__).resolve().parent.parent
18
19 # Quick-start development settings - unsuitable for production
20 # See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/
21
22 SECRET_KEY = os.environ['SECRET_KEY']
23
24 # SECURITY WARNING: don't run with debug turned on in production!
25 # DEBUG = False
26 DEBUG = False if os.environ.get("ENV", "development") == "production" else True
27
28 ALLOWED_HOSTS = ["purbeurre-mo1.herokuapp.com", "localhost", "127.0.0.1"]
29
30 # Application definition
31
32 INSTALLED_APPS = [
33     'django.contrib.admin',
34     'django.contrib.auth',
35     'django.contrib.contenttypes',
36     'django.contrib.sessions',
37     'django.contrib.messages',
38     'django.contrib.staticfiles',
39     'django.contrib.postgres',
40     'app_users',
41     'app_products',
42 ]
```


DIFFICULTÉS ET AXES D'AMÉLIORATIONS

DIFFICULTÉS

Bien que largement conseillé je n'utilisais pas suffisamment les documentations et ce projet avec django, très structuré, m'a vraiment montré l'aspect indispensable d'utiliser les documentations.

Le plus compliqué a vraiment été de trouver comment utiliser et personnaliser toutes les fonctionnalités proposées par Django comme les formulaires ou les modèles par exemple.

Ce framework oblige aussi à une rigueur dans la structure du projet ce qui m'a permis de comprendre que je suis encore loin de savoir structurer correctement un projet mais en connaissant mieux cet outil on peut vraiment faire un projet propre.

AXES D'AMÉLIORATIONS

Je pense que ce projet n'est pas suffisamment découpé, l'application `app_products` pourrait être scindée en trois applications pour séparer l'affichage des pages d'index et des mention légales, le peuplement de la base de données et la partie sur les recherches et affichages des produits alimentaires.

Je pourrais aussi couvrir plus de code avec des tests unitaires et de fonctionnement.

Et j'ai choisi la facilité pour la modification du fichiers css vu le peu de changements nécessaires mais je ne devrais pas modifier le css et plutôt ajouter un custom css par exemple.