Pur Beurre

Projet 8

Support technique

vivelegras.herokuapp.com

Back-end

Python
Django
PostgreSQL

Front-end

Bootstrap

Deployment

Heroku

Modules

Django

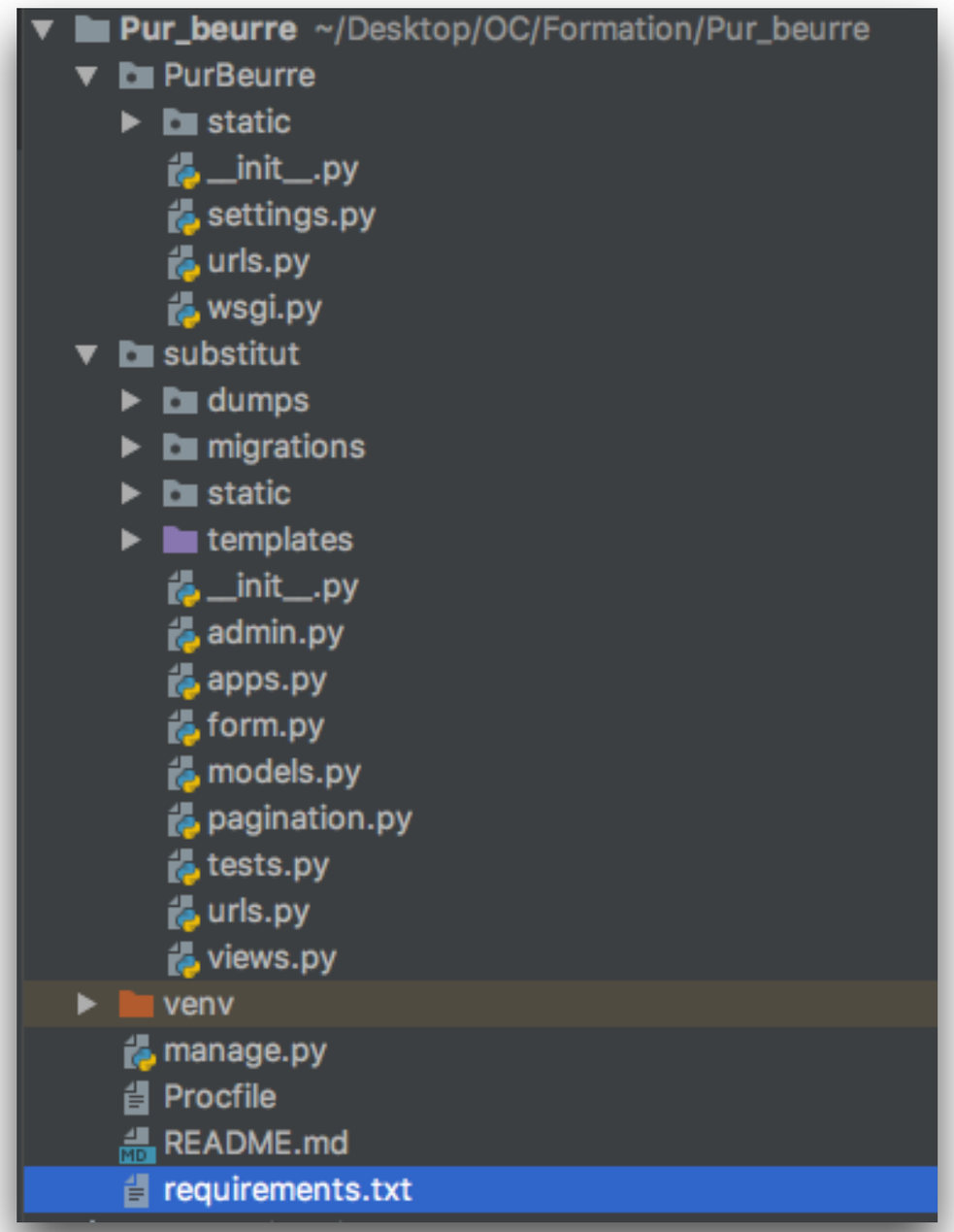
Requests

Json

Unittest

Psychopg2

Le projet



- Un projet : PurBeurre
- Une application : Substitut

Models

```python
from django.db import models
from django.contrib.postgres.fields import ArrayField

# Create your models here.
class Users(models.Model):
    email = models.EmailField(max_length=100)
    password = models.CharField(max_length=20)

    def __str__(self):
        return str(self.email)


class Products(models.Model):
    name = models.CharField(max_length=100)
    nutriscore = models.CharField(max_length=1)
    category = ArrayField(models.CharField(max_length=900),
                          blank=True,
                          size=None,
                          default=list)
    picture = models.URLField()
    url = models.URLField()

    def __str__(self):
        return self.name


class Saving(models.Model):
    contact = models.CharField(max_length=100, default='')
    product_key = models.CharField(max_length=9000, default='')
    date = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return str(self.contact)

    class Meta:
        ordering = ['date']
```

- 3 Tables
- Base de données PostgreSQL
- ORM de Django

Les vues

```python
import json
import requests
import random
from django.contrib.auth import login, authenticate, logout
from django.contrib.auth.models import User
from django.shortcuts import render, get_object_or_404

from .form import Connexion
from .models import Products, Users, Saving
from .pagination import customizePagination
```

- Django : login, logout, authenticate, User, render, get_object_or_404, models
- Python : json, requests, random
- + : form, pagination

```python
# Create your views here.
def index(request):...

def signup(request):...

def userlogin(request):...

def userlogout(request):...

def mentionslegales(request):...

def useraccount(request):...

def products(request):...

def userproducts(request):...

def search(request):...

def detail(request, product_id):...
```

- 10 vues composent l’application
- Elles sont liées à des templates

Views - SignUp

```python
def signup(request):
    """Signup function for user"""
    error = False

    if request.method == "POST":
        form = Connexion(request.POST)
        if form.is_valid():
            username = form.cleaned_data["username"]
            email = form.cleaned_data["email"]
            password = form.cleaned_data["password"]
            if email and password:
                user = User.objects.create_user(username=username,
                                                email=email,
                                                password=password)
                user.save()
                users_db = Users.objects.create(email=email,
                                                password=password)
                users_db.save()
                user = authenticate(username=username,
                                    email=email,
                                    password=password)
                login(request, user)
            else:
                error = True
    else:
        form = Connexion()

    return render(request, 'substitut/signup.html', locals())
```

- Création d'un utilisateur
- Utilisation de la classe Form de Django

```python
class Connexion(forms.Form):
    username = forms.CharField(label="Nom d'utilisateur", max_length=30)
    email = forms.CharField(label="Mail", max_length=100)
    password = forms.CharField(label="Mot de passe", widget=forms.PasswordInput)
```

Views - Login, Logout

```python
def userlogin(request):
    """LogIn a registered user"""
    error = False

    if request.method == "POST":
        form = Connexion(request.POST)
        if form.is_valid():
            username = form.cleaned_data["username"]
            email = form.cleaned_data["email"]
            password = form.cleaned_data["password"]
            user = authenticate(username=username,
                                email=email,
                                password=password)
            login(request, user)
        else:
            error = True

    else:
        form = Connexion()

    return render(request, 'substitut/login.html', locals())
```

```python
def userlogout(request):
    """Logout a registered user"""
    logout(request)
    return render(request, 'substitut/logout.html', locals())
```

- Login : authentification/ connexion d'un utilisateur déjà enregistré en base de données

- Logout : déconnexion de l'utilisateur

Views - Search (partie 1)

```python
def search(request):
    """User's search return the result from query's form on index"""
    query = request.GET.get('query')
    nutriscore_number = {1: 'a',
                         2: 'b',
                         3: 'c',
                         4: 'd',
                         5: 'e'}

    if not query:
        product_list = Products.objects.all()
        product = customizePagination(request, product_list, 6)
        context = {"product": product}
```

- Récupération de la requête
- Si requête vide : affichage de tous les items présents en base

Views - Search (partie 2)

```python
if not query:...
else:
    product_list = Products.objects.filter(name__icontains=query).order_by('nutriscore')

    if not product_list:
        search_one_product = requests.get("https://fr.openfoodfacts.org/cgi/search.pl?action=process&search_terms="
                                          + str(query) + "&sort_by=unique_scans_n&page_size=20&json=1")
        response = json.loads(search_one_product.text)
        products_created = 0
        for product_index in range(0, int(response['count'])):
            if response['products'][product_index]['states_hierarchy'][1] == 'en:complete':
                try:
                    get_name = response['products'][product_index]['product_name']
                except KeyError:
                    get_name = ''
                try:
                    get_url = response['products'][product_index]['url']
                except KeyError:
                    get_url = ''
                try:
                    get_img = response['products'][product_index]['image_front_url']
                except KeyError:
                    get_img = ''
                try:
                    get_nutriscore = response['products'][product_index]['nutrition_grades']
                    for key, value in nutriscore_number.items():
                        if get_nutriscore == value:
                            get_nutriscore = key
                except KeyError:
                    get_nutriscore = ''
                try:
                    categories_tags = response['products'][product_index]['categories_hierarchy'][:]
                    listing_categories = []
                    for c in categories_tags:
                        cleaned_cat = c.split(':')
                        listing_categories.append(cleaned_cat[1])
                    get_cat = listing_categories
                except KeyError:
                    get_cat = ''

                Products.objects.create(name=get_name,
                                        nutriscore=get_nutriscore,
                                        category=get_cat,
                                        picture=get_img,
                                        url=get_url)
                products_created += 1
                if products_created < 30:
                    break
```

- Récupération de la requête
- Si un produit correspond en base : le retourne
- Sinon : envoie une requête sur OpenFoodFacts et créer le produit en base

Views - Search (Partie 3)

```
    filteringbycategory = Products.objects.filter(name__icontains=query)
    cat = []
    for i in filteringbycategory:
        cat.extend(i.category)
    cat = list(set(cat))

    try:
        product_list = Products.objects.filter(category__contained_by=cat) \
            .order_by('nutriscore', 'name')
        pc = []
        for p in filteringbycategory:
            pc.append(p.picture)
        product = customizePagination(request, product_list, 6)
        try:
            context = {"product": product,
                       "urlp": query,
                       "name": query,
                       "picture": pc[0]}
        except:
            context = {"product": product,
                       "urlp": query,
                       "name": query}
        print('Products.objects.filter(category__contained_by=cat)')
    except:
        product_list = Products.objects.filter(name__icontains=query) \
            .order_by('nutriscore', 'name')
        pc = []
        for p in filteringbycategory:
            pc.append(p.picture)
        product = customizePagination(request, product_list, 6)
        try:
            context = {"product": product,
                       "urlp": query,
                       "name": query,
                       "picture": pc[0]}
        except:
            context = {"product": product,
                       "urlp": query,
                       "name": query}
        print('Products.objects.filter(name__icontains=query)')

return render(request, 'substitut/search.html', context)
```

- Récupère l'objet en base, l'image, la catégorie
- Filtre les résultats selon la catégorie de la requête
- Retourne la liste de produit selon la catégorie du produit de la requête sur le template

Views - Detail

```python
def detail(request, product_id):
    """Details for a product"""
    try:
        product_detail = get_object_or_404(Products, pk=product_id)
        user = request.user.email
        saving = request.POST.get('saving')
        if saving:
            Saving.objects.create(contact=user,
                                  product_key=product_detail.pk)
            return redirect('substitut:userproducts')

        print(product_detail.category)
        context = {'name': product_detail.name,
                   'nutriscore': product_detail.nutriscore,
                   'picture': product_detail.picture,
                   'url': product_detail.url}
    except:
        product_detail = get_object_or_404(Products, pk=product_id)
        context = {'name': product_detail.name,
                   'nutriscore': product_detail.nutriscore,
                   'picture': product_detail.picture,
                   'url': product_detail.url}

    return render(request, 'substitut/detail.html', context)
```

- Affiche la page de détail d'un produit
- nom, nutriscore, image et url vers OpenFoodFacts
- Bouton pour sauvegarder le produit si l'utilisateur est connecté
- Redirection vers les produits sauvegardés le bouton est actionné

Views - Userproducts

```python
def userproducts(request):
    """Return saved products of a user"""
    try:
        user = request.user.email
        products_to_display = Saving.objects.filter(contact=user)
        keys_list = []

        for item in products_to_display:
            keys_list.append(item.product_key)

        product_filter = Products.objects.filter(pk__in=keys_list)
        product = customizePagination(request, product_filter, 6)
        context = {"product": product}
    except:
        product = None
        context = {"product": product}

    return render(request, 'substitut/userproducts.html', context)
```

- Affiche les produits enregistrés par un utilisateur

Pagination

```python
from django.core.paginator import Paginator, PageNotAnInteger, EmptyPage


def customizePagination(request, to_paginate, product_by_page):
    """Function to paginate results"""
    paginator = Paginator(to_paginate, product_by_page)
    page = request.GET.get('page')

    try:
        product = paginator.page(page)
    except PageNotAnInteger:
        product = paginator.page(1)
    except EmptyPage:
        product = paginator.page(paginator.num_pages)

    return product
```

- Création d’une fonction pour la pagination des produit dans l’application

- Module de Django : Paginator

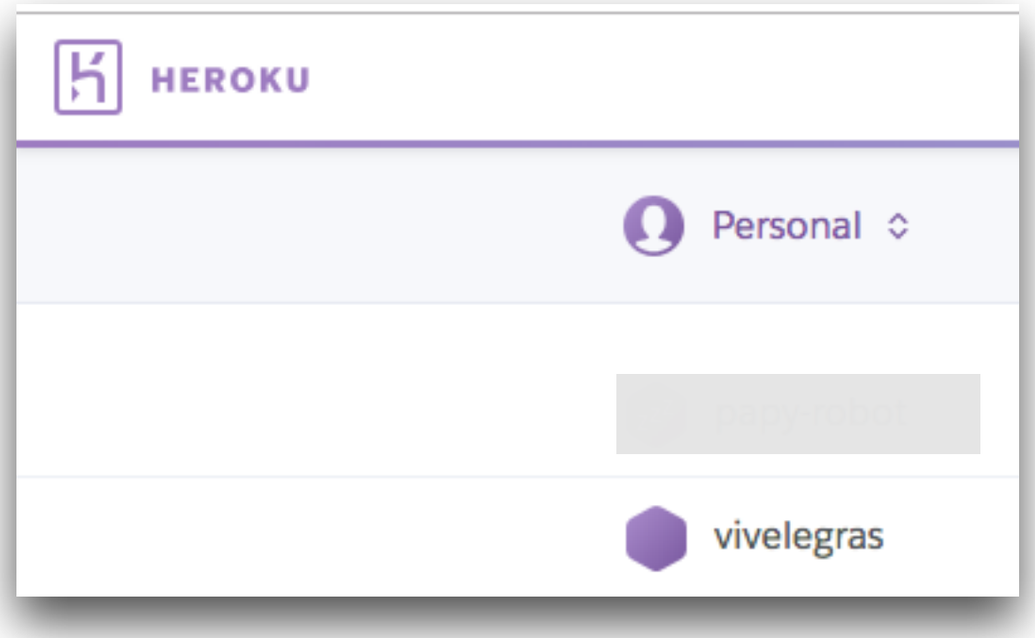
Tests

```python
class PageTestCase(TestCase):
    def test_index_page(self):...
    def test_mentionlegales_page(self):...
    def test_search_page(self):...
    def test_signup_page(self):...
    def test_login_page(self):...
    def test_logout_page(self):...
    def test_user_account_page(self):...
    def test_user_products_page(self):...

class DetailPageTestCase(TestCase):
    def setUp(self):...
    def test_detail_page_returns_200(self):...
    def test_detail_page_returns_404(self):...

class SignupPageTestCase(TestCase):
    def setUp(self):...
    def test_signup_page_returns_200(self):...
    def test_form_is_valid(self):...
    def test_form_not_valid(self):...
```

```python
class LoginLogoutTestCase(TestCase):
    def setUp(self):...
    def test_user_login(self):...
    def test_user_logout(self):...

class SavingPageTestCase(TestCase):
    def setUp(self):...
    def test_saving_product_is_registered(self):...
    def test_saving_product_not_registered(self):...
```

- Tests unitaires
- Permettent de tester une fonctionnalité (si un formulaire est valide, un code de statut de page, etc.)

Déploiement



- Heroku
- Variables d’environnement : ENV, SECRET_KEY
- Migration de la base de données
- Dump de la base de données

Méthodologie