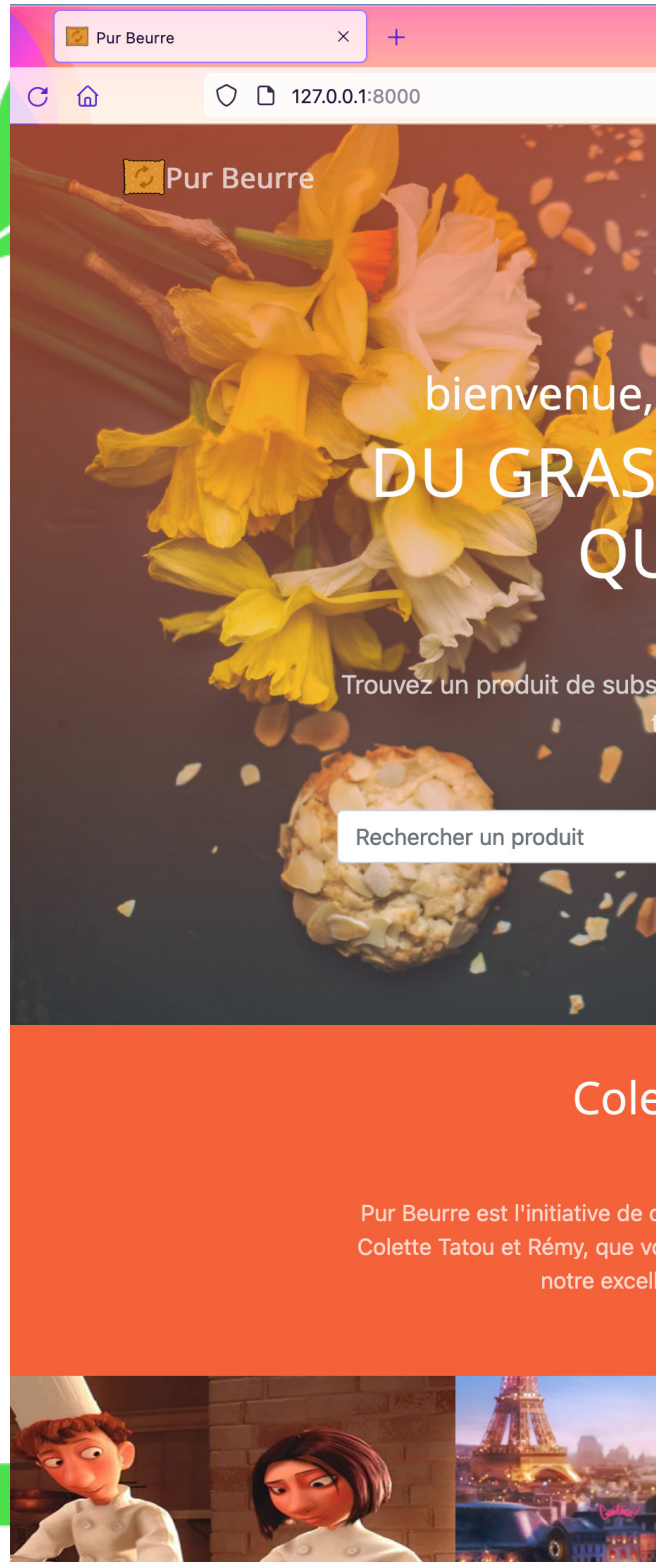
# OPEN FOOD

**Platform for Nutella lovers**

*Fat but good quality ...*



# SOMMAIRE



Nutella Fans

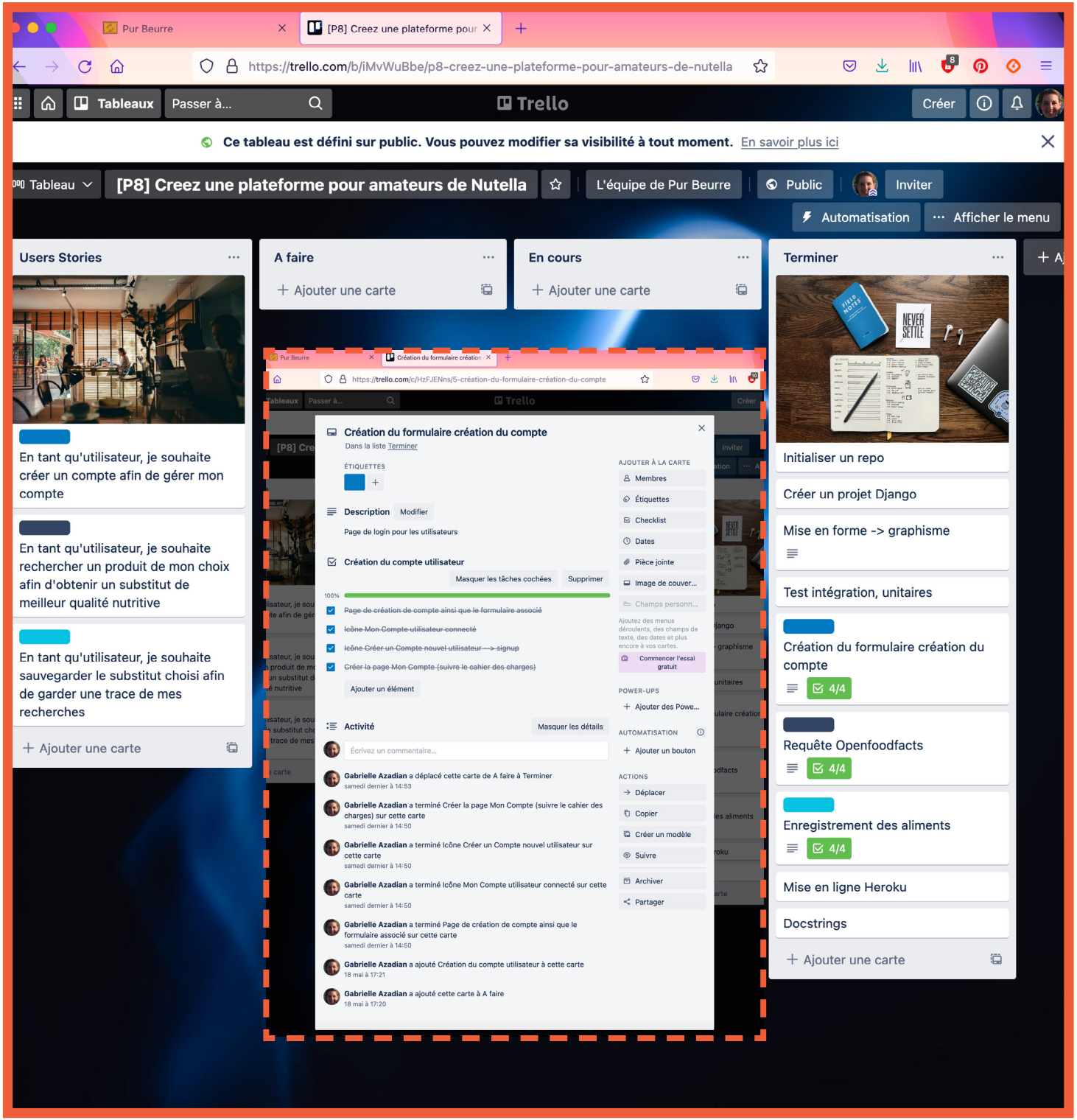
## Context

Pur Beurre aims to establish a platform for health-conscious individuals who seek high-quality products.
Our goal is to provide products that match the user's search criteria and have a superior nutriscore rating (A or B) in case they are looking for crisps or any other product.
Once the user is satisfied with the proposal, they can log in to their account and save it to their favourites.
Additionally, they can remove any product from their favourites if it no longer suits their preferences.

Nutella Fans

# Trello

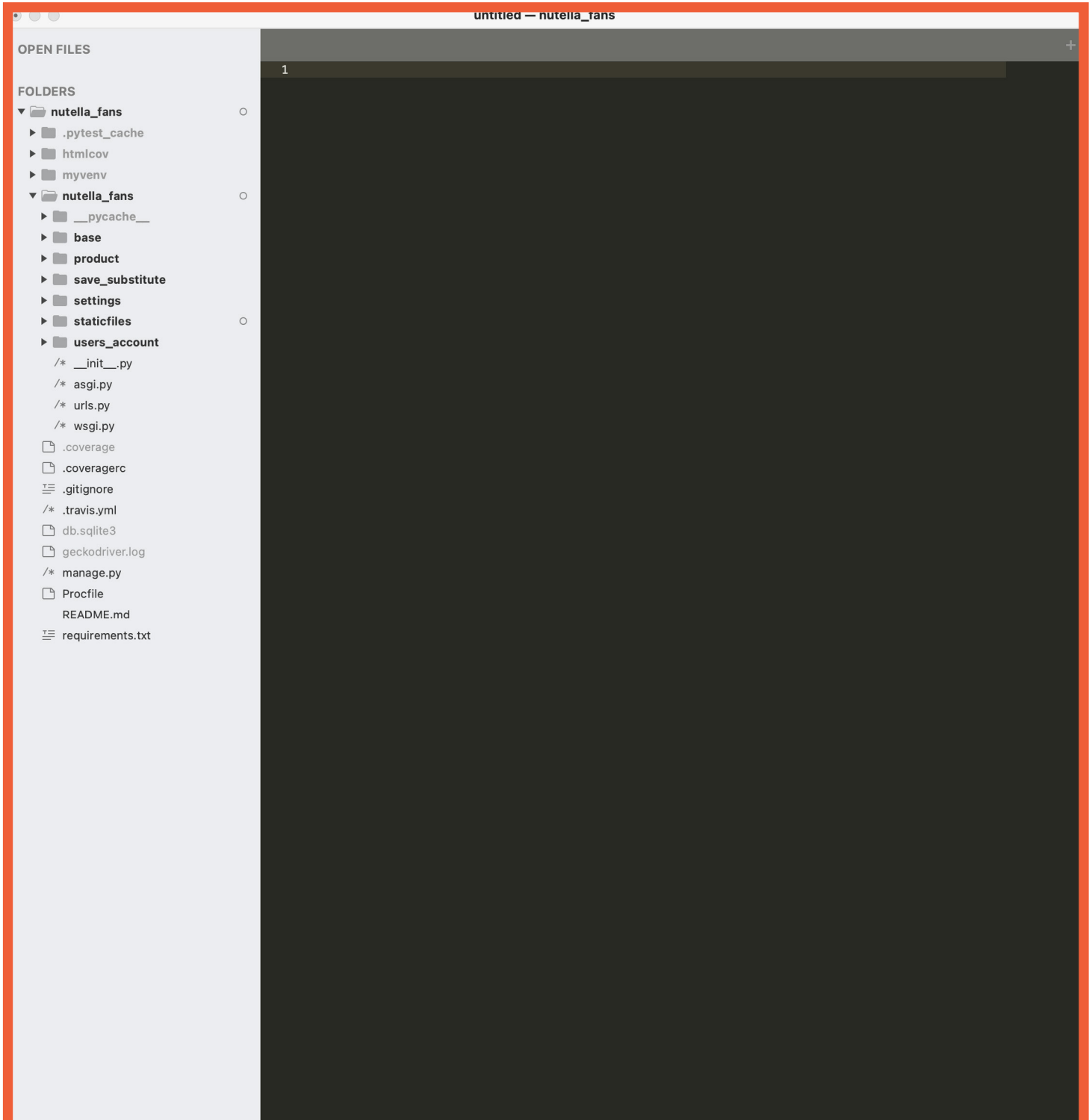
https://trello.com/b/iMvWuBbe/p8-creez-une-plateforme-pour-amateurs-de-nutella




Nutella Fans

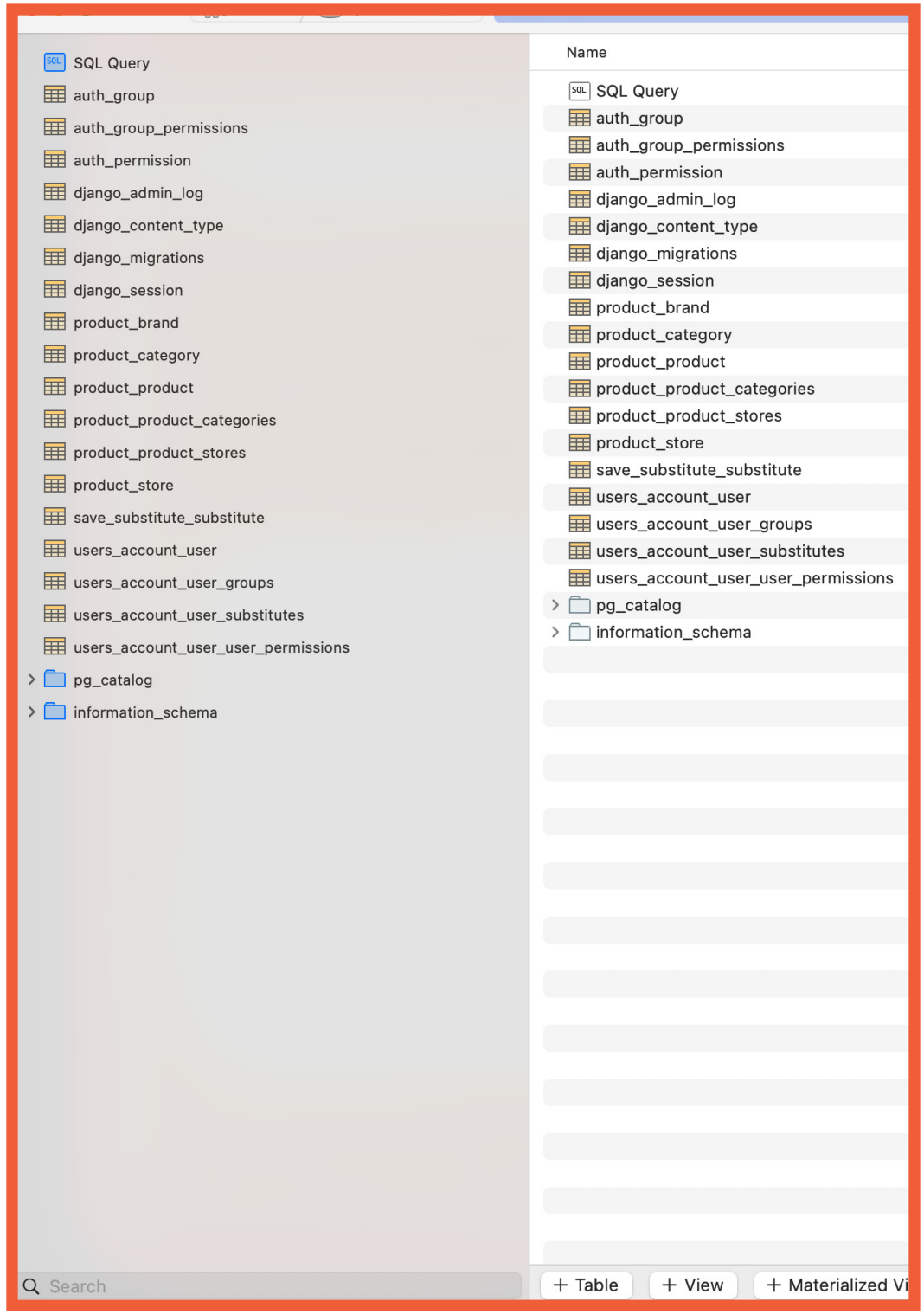
# Projet App

## Folders and app tree



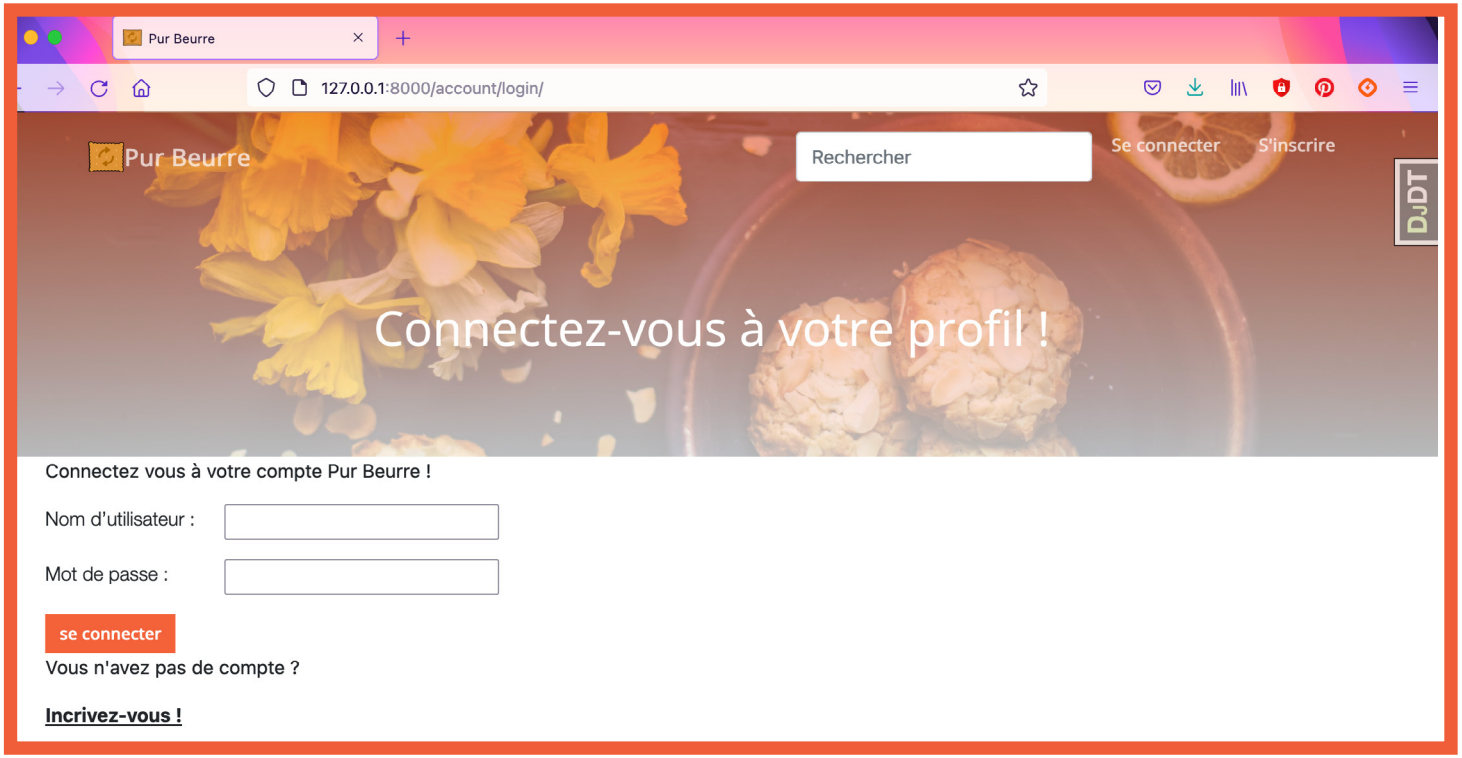
# Models

```
from django.db import models

class Brand(models.Model):
    name = models.CharField(max_length=200)
    def __str__(self):
        return self.name

class Category(models.Model):
    name = models.CharField(max_length=200)
    def __str__(self):
        return self.name

class Store(models.Model):
    name = models.CharField(max_length=200)
    def __str__(self):
        return self.name

class Product(models.Model):
    name = models.CharField(max_length=200)
    nutriscore = models.CharField(max_length=1, null=True)
    nova = models.IntegerField(null=True)
    url = models.URLField(max_length=200)
    barcode = models.CharField(max_length=200, unique=True)
    description = models.TextField(null=True)
    picture = models.URLField(null=True)
    ...
    brand = models.ForeignKey(
        Brand, on_delete=models.CASCADE)
    categories = models.ManyToManyField(Category)
    stores = models.ManyToManyField(Store)

    def get_substitutes(self):
        product_categories = self.categories.all()
        result = Product.objects.filter(categories__in=product_categories)\
            .filter(nutriscore__lt=self.nutriscore)\
            .exclude(pk=self.pk)\
            .order_by('nutriscore').distinct()
        return result
```



## users_account --> template

registration and login page




## FBV --> CBV

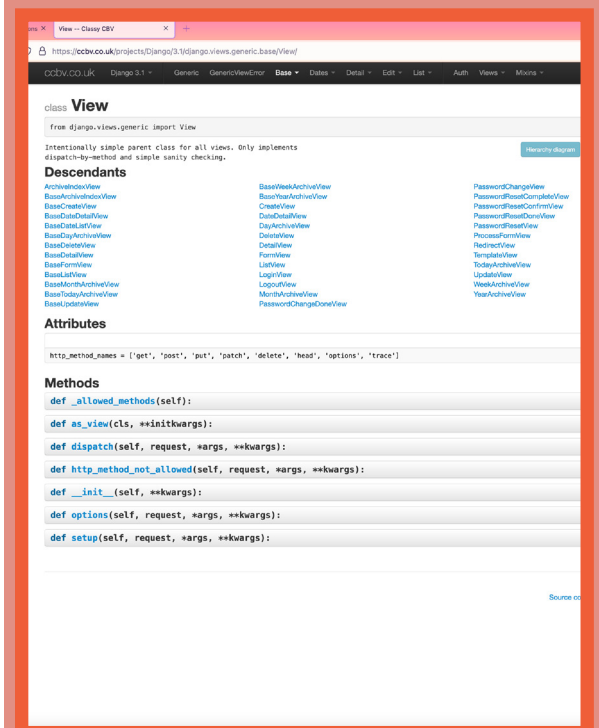
Function-Based Views --> Class-Based View

-> log in scripts FBV
- urls.py
- views.py

request.POST['username'] --> accès à des dictionnaires envoyés par leurs clés (par le formulaire)

-> scripts FBV
- urls.py
- views.py

Class generic LoginView




```
def login_request(request):
        if request.method == 'POST':
                username = request.POST['username']
                password = request.POST['password']
                user = authenticate(request,
                        username=username, password=password)
        if user is not None:
                login(request, user)
                messages.info(request, f»You are now logged in as {username}.»)
                return render(request, 'users_account/home.html')
        else:
        messages.error(request, «Invalid username or password.»)
                return render(request, 'users_account/registration/login.html')
```

Script Views pour le login --> FBV

```
urlpatterns = [
path('', views.base, name='base'),
path('sign_up/', views.sign_up, name='sign_up'),
 path('login/', views.login_request, name='login_request'),]
```

Script urls

```
class LoginView(View):

    «»»Clas Based View for user's login
    Attributes:
        template_name (str): template location
    «»»

    template_name = «registration/login.html»
```

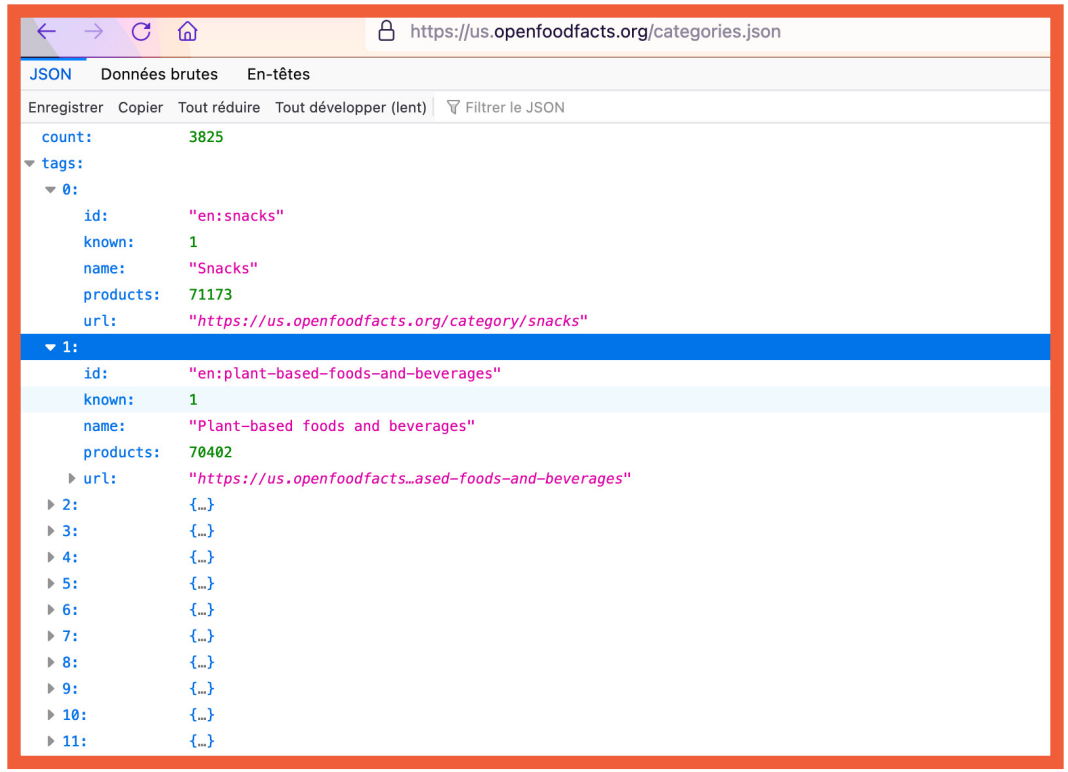
Script Views pour le login --> CBV

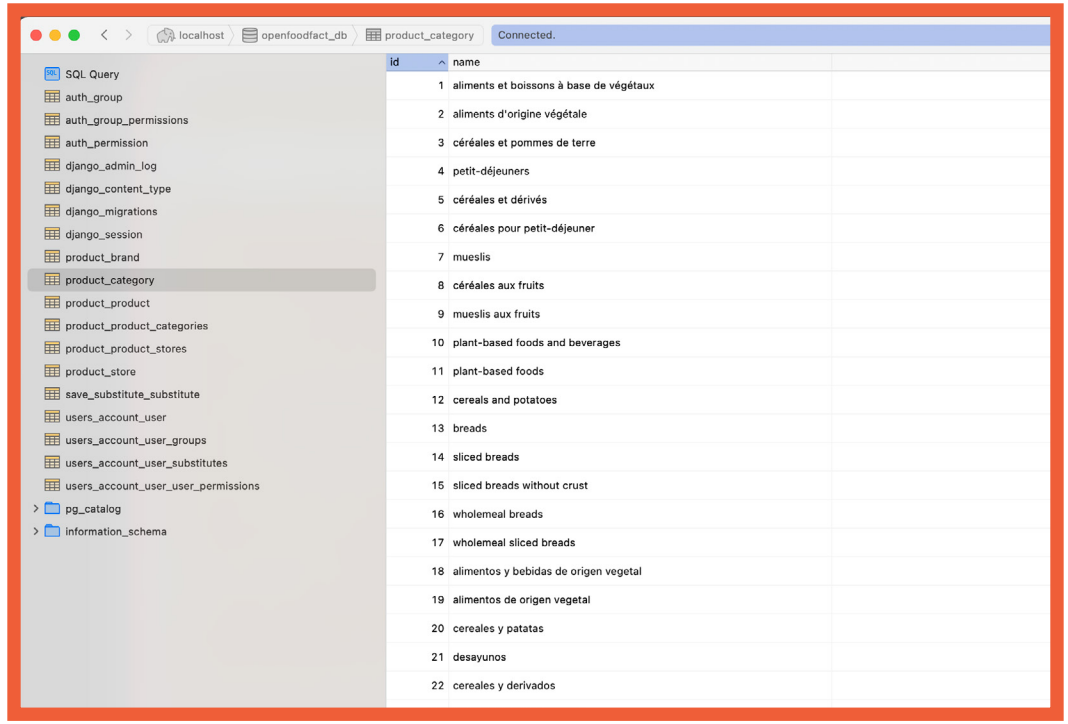
```
urlpatterns = [
path('sign_up/', SignupView.as_view(), name='sign_up'),
path('login/', LoginView.as_view(), name='login'),
path('logout/', views.logout_request, name='logout'),
path('profile/<int:pk>',
     UserDetailView.as_view(), name='profile'),]
```

Script urls

# product --> API and DataBase

```
def handle(self, *args, **options):
    category_list = self.get_category()
    for category_dict in category_list:
        products = self.get_products(category_dict)
        for product in products:
            p = self.create_product(product)
            if not p:
                continue
            else:
                category_names = product.get(
                    «categories»).lower().split(«,»)
                for category in category_names:
                    c, created = Category.objects.get_or_create(
                        name=category)
                    p.categories.add(c)
                if product.get(«stores»):
                    stores = product.get(«stores»).lower().split(«,»)
                    for store in stores:
                        s, created = Store.objects.get_or_create(
                            name=store)
                        p.stores.add(s)

def get_category(self):
    «»»response of the request to extract data from API

    Returns:
        LIST: list of categories
    «»»
    response = requests.get(«https://fr.openfoodfacts.org/categories.json»)
    result_category = response.json()
    data_category = result_category.get('tags')
    categories = [data.get('name') for data in data_category
                  if data.get(«name»)]
    category_name = categories[0:10]
    return category_name
```



## product --> test mock

```
class GetCategory(TestCase):

@mock.patch('requests.get')
def test_get_categories(self, mock_get):
    mock_response = mock.Mock()
    mock_api = {
        'tags': [{
            'id': 'en:plant-based-foods-and-beverages',
            'known': 1,
            'name': 'Aliments et boissons à base de végétaux',
            'products': 107490,
            'url': 'https://fr.openfoodfacts…ssons-a-base-de-vegetaux'
        }
        ]
    }

    mock_response.json.return_value = mock_api
    mock_response.status_code = 200
    mock_get.return_value = mock_response

    result = Command().get_category()
    self.assertEquals(result,
                      [«Aliments et boissons à base de végétaux»]
                      )
```

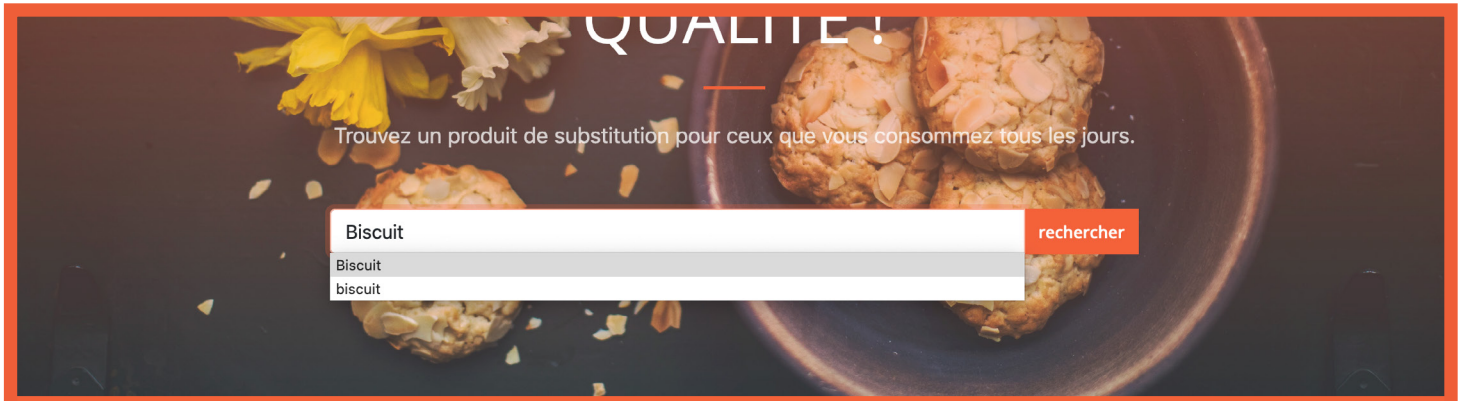
Openfoodfact's API request call test and response simulation.

mock_get.return_value = mock_response --> simulated values return

## product --> templates

app --> base
--> base.html

app --> product
--> product_list.html
--> substitute_list.html




Colette et Rémi

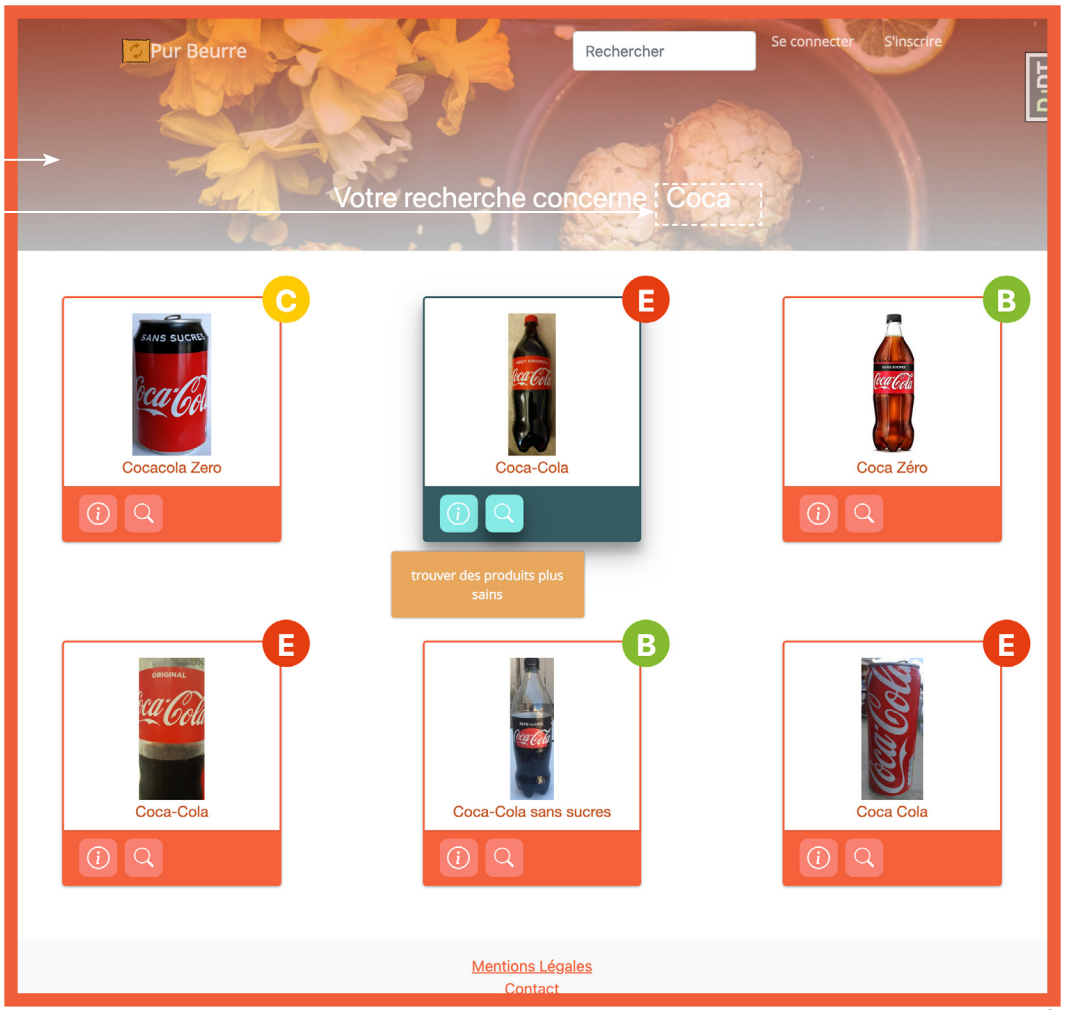


Nutella Fans

# Gestion des produits --> django

```
<div class=»col-lg-8 align-self-baseline»>
        <p class=»text-white-75 mb-5»> Find a substitute product.</p>
        <form  method=»GET» action=»{% url ‘product_list’ %}»>
            <div class=»input-group mb-3»>
                <input type=»text» class=»form-control»
placeholder=»Rechercher un produit» name=»search_product» value=’{{request.GET.
search_product}}’>
                <input type=»submit»  class=»btn btn-primary»
value=»Rechercher»>
            </div>
        </form>
</div>
```

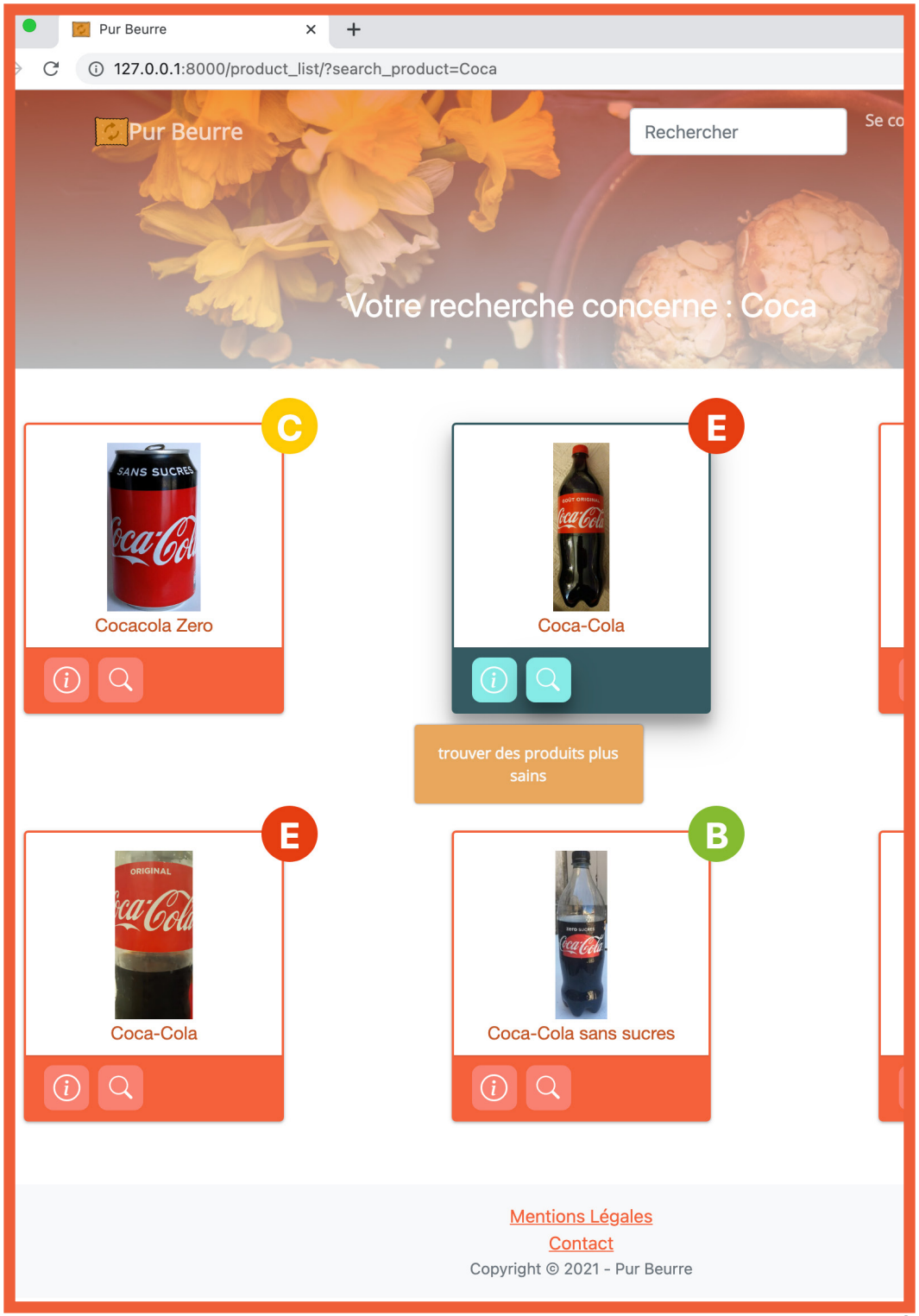
```
class ProductListView(ListView):

    template_name = 'product_list.html'
    model = Product

    def get_queryset(self):
        «»» List of products according to user's research
        queryset --> get products filter by 'search_product' from base.html form

        Returns:
            Dict: list of products model queryset
        «»»
        search = self.request.GET.get('search_product', '').strip()
        qs = super().get_queryset()
        if search:
            return qs.filter(name__icontains=search)
        else:
            return qs
```


Nutella Fans

## Gestion  des produits -->html

```
    <div class=»row»>
        {% if product_list %}
            {% for product in product_list %}
                <div class=»col-lg-4 col-md-6 col-sm-12 d-flex col-card justify-content-center»>
                    <div class=»card product-card d-flex flex-column «>
                        <img src=»{{product.picture}}» class=»card-img-top product-img img-fluid rounded d-block align-m»>
                        <p class=»card-title»>{{product.name}}</p>
                        <div class=»card-footer d-flex product align-content-around mt-auto»>
                            <a  class=»d-flex justify-content-center align-items-center btn btn-detail» name=»detail_product»  type=»button» href=»{% url 'detail_product' product.id  %}»
                                data-toggle=»tooltip» data-placement=»bottom»>
                            <span>Afficher les détails du produit choisi</span>
                            <i class=»bi bi-info-circle»></i>
                            </a>
                            <a class=»d-flex justify-content-center align-items-center btn btn-detail» name=»search_substitute» type=»button» href=»{% url 'substitute_list' product_id=product.id %}» value='{{request.GET.search_substitute}}' data-toggle=»tooltip» data-placement=»bottom»>
                                <span>Trouver des produits plus sains</span>
                                <i class=»bi bi-search»></i>
                            </a>
                        </div>
                        <div class=»card-score card-score-color-{{ product.nutriscore }} text-center» data-bs-toggle=»tooltip» data-bs-placement=»right»
                            title=»Nutriscore, de A (très bon) à ''E" (Très mauvais)»>{{product.nutriscore}}
                        </div>
                    </div>
                </div>
            {% endfor %}
    {% else %}
        <div class=»container h-100»>
            <div class=»row justify-content-center h-100»>
                <div class=»col-lg-4 col-md-6 col-sm-12 d-flex text-center»>
                    <h2 class=»text-center no-favoris»>Le produit n'existe pas  !</h2>
                </div>
            </div>
```



## Tests --> test_views

```
from django.test import TestCase, RequestFactory, Client


class BaseTest(TestCase):
    def setUp(self):
            self.product_url = reverse('product_list')
            self.client = Client()
            self.factory = RequestFactory()
            return super().setUp()


class ProductListTest(BaseTest):

    def test_get_queryset(self):
        request = self.factory.get(self.product_url)
        view = ProductListView()
        view.request = request

        qs = view.get_queryset()

        self.assertQuerysetEqual(qs, Product.objects.all())

    def test_status_code(self):
        response = self.client.get(self.product_url)
        self.assertEquals(response.status_code, 200)
```

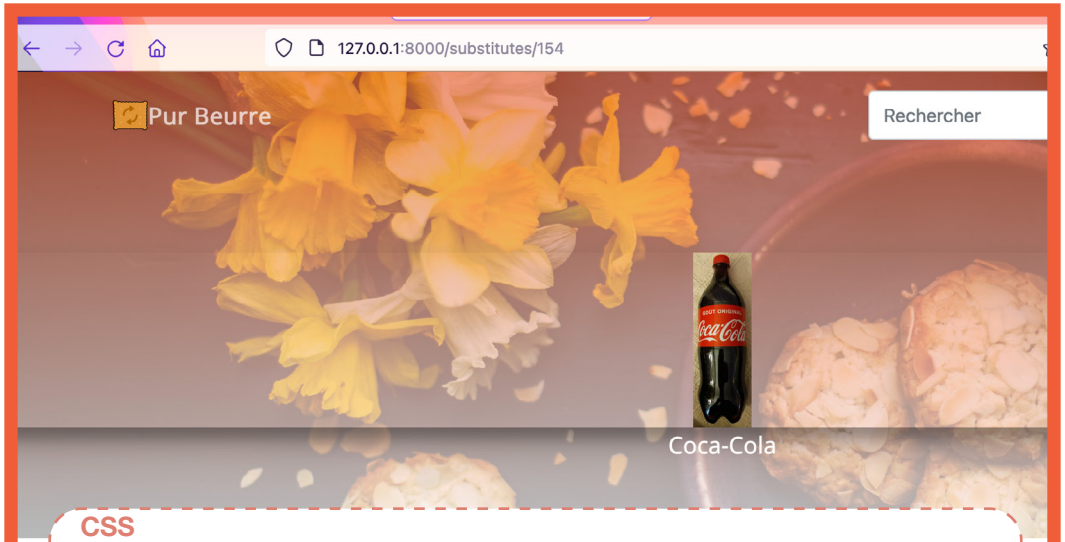
RequestFactory returns a ***request***, Client returns a ***response***.
RequestFactory --> this is a factory for creating request objects. Client simulates a complete ***request-response*** cycle.
Client executes views, tests views, inspects response.

RequestFactory 's Test --> **get_queryset**
Creating a request object using a request factory
Instantiate the view
I attach the request to the view
I call the method and compare it with the result

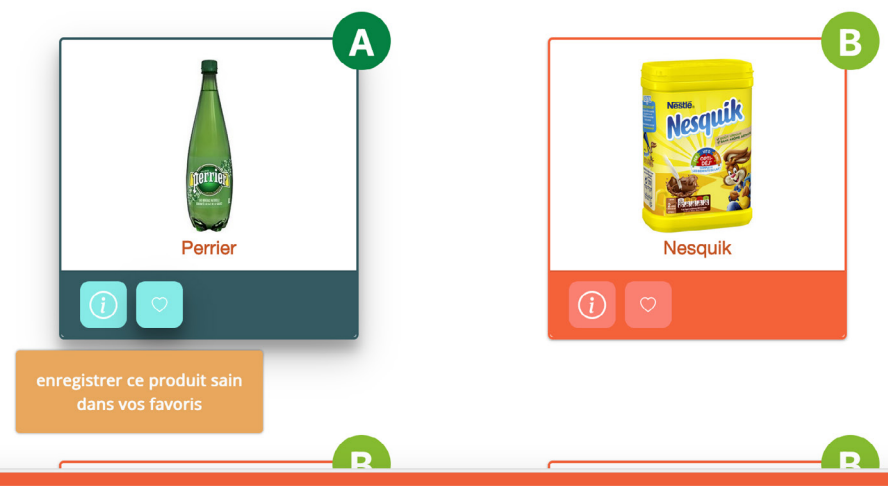
response Test HTTP --> status_code

# Substituts --> templates, html, CSS

```
{% for substitute in product_list %}
        <div class=»col-lg-4 col-md-6 col-sm-12 d-flex col-card justify-content-center»>
            <div class=»card product-card d-flex flex-column «>
                <div class=»card-score card-score-color-{{ substitute.nutriscore }} text-center» data-bs-toggle=»tooltip» data-bs-placement=»right»
                    title=»Nutriscore, de A (très bon) à ‘’E” (Très mauvais)»>{{substitute.nutriscore}}
                </div>
                <img src=»{{substitute.picture}}» class=»card-img-top product-img img-fluid rounded d-block align-m»>
                <p class=»card-title»>{{substitute.name}}</p>
                <div class=»card-footer d-flex favoris align-content-around mt-auto»>
                    <a  class=»d-flex justify-content-center align-items-center btn btn-detail» name=»detail_product»  type=»button» href=»{% url ‘detail_product’ substitute.id %}»
                        data-toggle=»tooltip» data-placement=»bottom»>
                        <span>Afficher les détails du produit choisi</span>
                        <i class=»bi bi-info-circle»></i>
                    </a>
                {% if user.is_authenticated %}
                <form  method=»post» action=»{% url ‘substitute_save’ %}»>
                {% csrf_token %}
                <input type=»hidden» name=»product_id» value=»{{ product.id }}»>
                <input type=»hidden» name=»substitute_id» value=»{{ substitute.id }}»>
                <button type=»submit» class=»d-flex justify-content-center align-items-center btn btn-link btn-choice»
                    data-bs-toggle=»tooltip» data-bs-placement=»bottom» >
                    <span>Enregistrer ce produit sain dans vos favoris</span>
                    <i class=»bi bi-heart»></i>
                    </button>
                    </form>
                {% endif %}
                </div>
            </div>
        </div>
    {% endfor %}
```




```
CSS
.product-card {
  position: relative;
  width: 65%;
  margin-bottom: 3.5rem;
  margin-top: 3rem;
  border-radius: 0.25rem;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
  transition: all 0.3s cubic-bezier(.25,.8,.25,1);
}
.product-card:hover {
 box-shadow: 0 14px 28px rgba(0,0,0,0.25), 0 10px 10px rgba(0,0,0,0.22);
 border: 2px solid #345a61;
}
```



# Substituts --> django

```
class SubstituteListView(ListView):

    template_name = 'product/substitute_list.html'
    model = Product
    context_object_name = 'product_list'

    def get_context_data(self):
        context = super().get_context_data()
        product = Product.objects.get(pk=self.kwargs.get('product_id'))
        context['product'] = product
        return context

    def get_queryset(self, *args, **kwargs):
        product_id = self.kwargs.get('product_id')
        product = Product.objects.get(pk=product_id)
        return product.get_substitutes()
```

**context_object_name** --> variable name returned by Django to display lists. I renamed it to product_list.
In template_name = 'product/substitute_list.html'
{% for **substitute** in **product_list** %}

**def get_context_data** --> get the data from the url using the attribute **kwargs**
product = Product.objects.get(pk=**self. kwargs.get('product_id'))**

**def get_queryset** --> retrieve the product id from the url,
make a query to retrieve the id from the database, then apply the **get_substitutes()** function to display the best products.

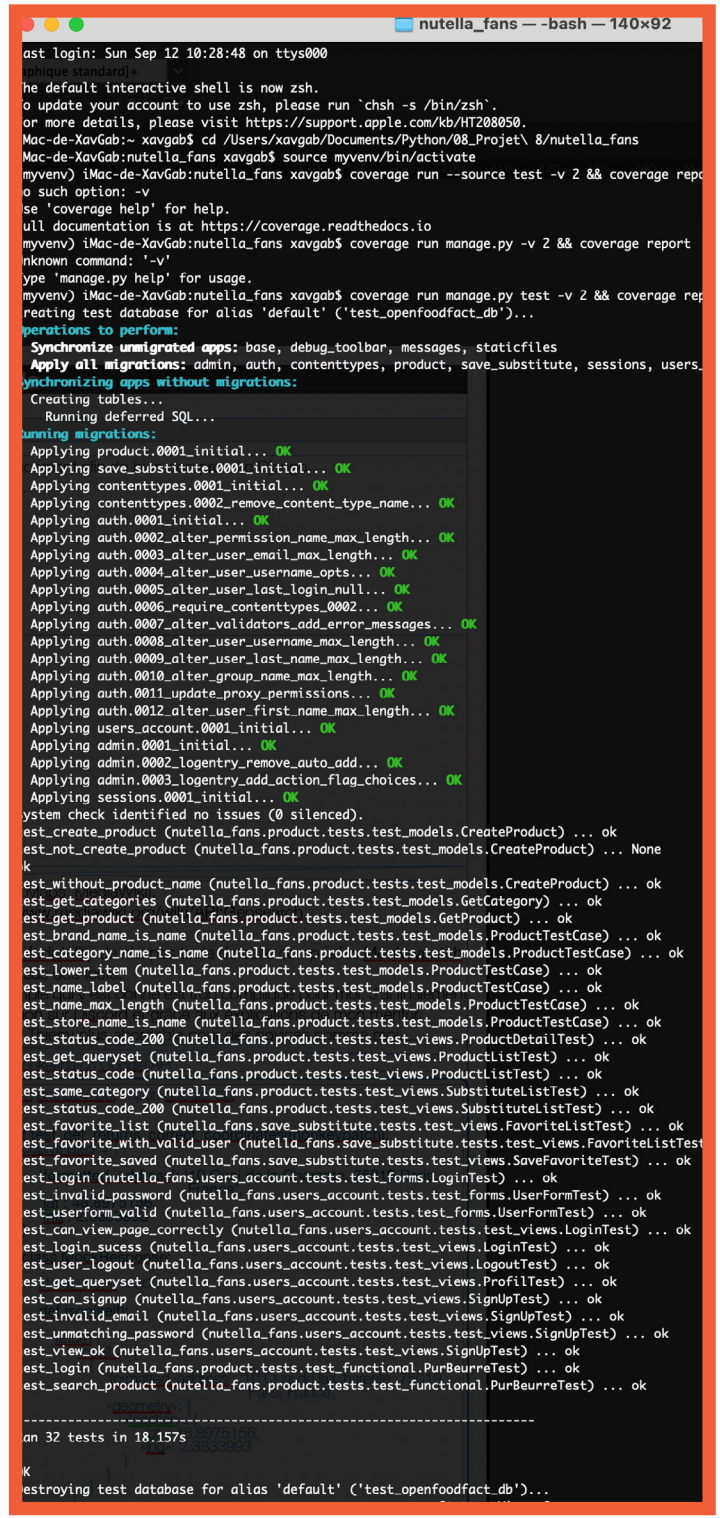
```
class SubstituteListTest(TestCase):
def test_same_category_better_nutriscore(self):
        brand_p1 = Brand.objects.create(name='ferrero').pk
        brand_p2 = Brand.objects.create(name='Funky Veggie').pk
        brand_p3 = Brand.objects.create(name='La Vache qui rit').pk
        category = Category.objects.create(name='Produits à tartiner')
        other_category = Category.objects.create(name='Produits laitiers')
        product = Product.objects.create(name='Nutella - Ferrero - 400 g',
                                         nutriscore='e', nova=4,
                                         url='https://fr.openfoodfacts.org/produit/'
                                         '3017620422003/nutella-ferrero',
                                         barcode=3017620422003,
                                         description=»Sucre, huile de palme, noisettes 13%»
                                         « lait écrémé en poudre 8,7%»
                                         «cacao maigre 7,4%, émulsifiants: lécithines [soja] ; vanilline. Sans gluten»,
                                         picture='https://images.openfoodfacts.org/images/'
                                         ' products/301/762/042/2003/front_en.288.400.jpg',
                                         brand_id=brand_p1)
        product_better = Product.objects.create(name='OUF! La pâte à tartiner Cacao Noisettes - Funky Veggie - 200 g',
                                                nutriscore='b', nova=3,
                                                url='https://fr.openfoodfacts.org/produit/'
                                                '3770008009653/ouf-la-pate-a-tartiner-cacao-noisettes-funky-veggie',
                                                barcode=3770008009653,
                                                description=»Purée de haricots rouges* 30% (eau, farine de haricots rouges*),»
                                                «sucre de fleur de coco*,»
                                                «eau, purée de noisettes torréfiées* 17,5%, poudre de cacao 4,8%.»
                                                «ingrédients issus de l'agriculture biologique»,
                                                picture='https://images.openfoodfacts.org/images/'
                                                ' products/377/000/800/9653/front_fr.63.400.jpg',
                                                brand_id=brand_p2)
        product_worst = Product.objects.create(name='La Vache qui rit 32 portions - 535 g',
                                               nutriscore='d', nova=4,
                                               url='https://fr.openfoodfacts.org/produit/'
                                               '3073781070309/la-vache-qui-rit-32-portions',
                                               barcode=3073781070309,
                                               description=»LAIT écrémé réhydraté (origine : France),»
                                               «FROMAGES, BEURRE, protéines de LAIT,»
                                               «concentré des minéraux du LAIT,»
                                               «sels de fonte : polyphosphates, concentré LACTIQUE LAITIER.»,
                                               picture='https://images.openfoodfacts.org/images/'
                                               ' products/307/378/107/0309/front_fr.64.400.jpg',
                                               brand_id=brand_p3)
        product.categories.add(category)
        product_same_category.categories.add(category)
        product_other_category.categories.add(other_category)
        response = product.get_substitutes()
        assert product_better in response
        assert product_worst not in response
```

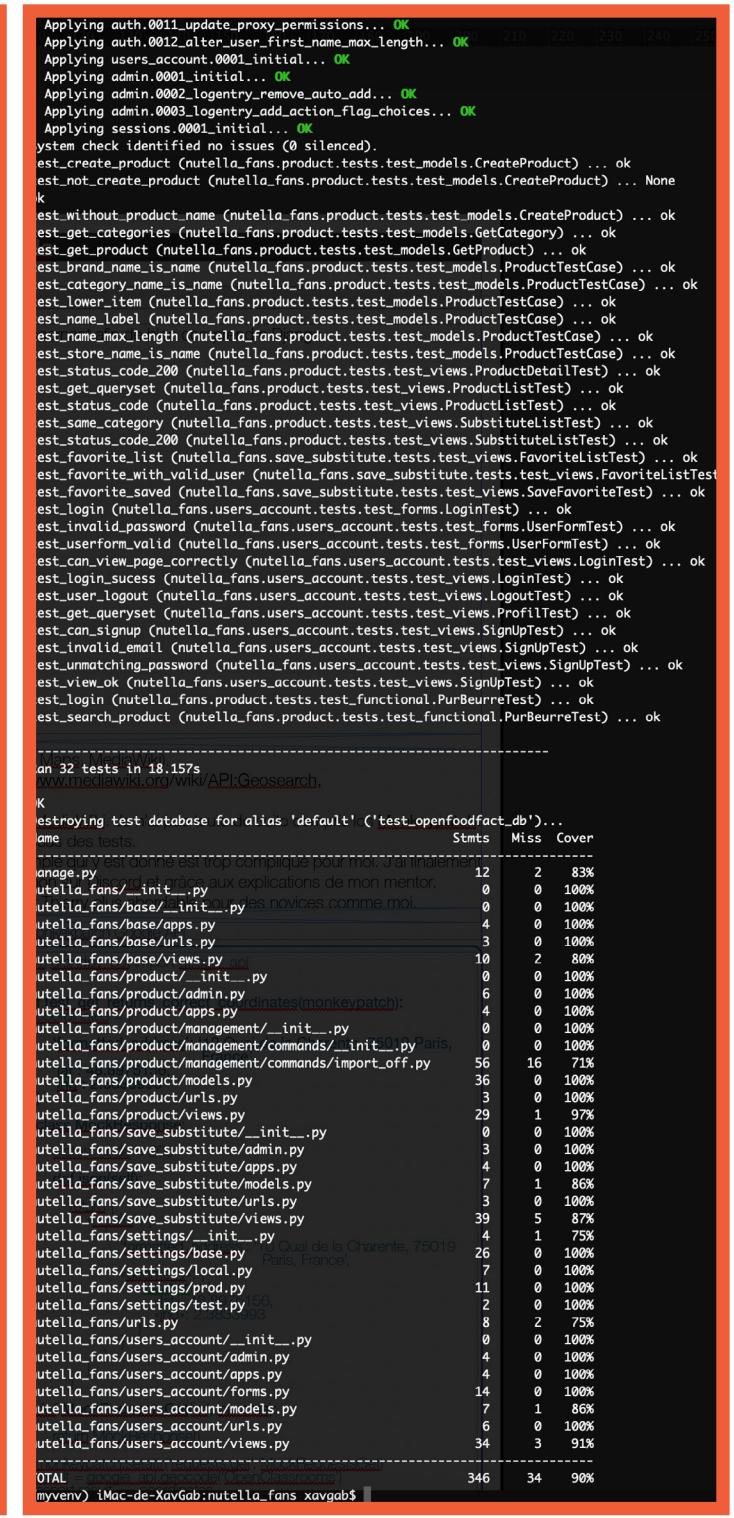
# Coverage --> launch

The Nutella lover's platform product has a test coverage of 90%.

python3 manage.py test test.users_account.tests.test_views.SignUpTest.test_can_signup

coverage run  manage.py test -v 2 && covert report && html



# Coverage --> report html

The Nutella lover's platform product has a test coverage of 90%


| dule ↑ | statements | missing | excluded | coverage |
|---|---|---|---|---|
| age.py | 12 | 2 | 0 | 83% |
| ella_fans/__init__.py | 0 | 0 | 0 | 100% |
| ella_fans/base/__init__.py | 0 | 0 | 0 | 100% |
| ella_fans/base/apps.py | 4 | 0 | 0 | 100% |
| ella_fans/base/urls.py | 3 | 0 | 0 | 100% |
| ella_fans/base/views.py | 10 | 2 | 0 | 80% |
| ella_fans/product/__init__.py | 0 | 0 | 0 | 100% |
| ella_fans/product/admin.py | 6 | 0 | 0 | 100% |
| ella_fans/product/apps.py | 4 | 0 | 0 | 100% |
| ella_fans/product/management/__init__.py | 0 | 0 | 0 | 100% |
| ella_fans/product/management/commands/__init__.py | 0 | 0 | 0 | 100% |
| ella_fans/product/management/commands/import_off.py | 55 | 16 | 0 | 71% |
| ella_fans/product/models.py | 36 | 0 | 0 | 100% |
| ella_fans/product/urls.py | 3 | 0 | 0 | 100% |
| ella_fans/product/views.py | 29 | 1 | 0 | 97% |
| ella_fans/save_substitute/__init__.py | 0 | 0 | 0 | 100% |
| ella_fans/save_substitute/admin.py | 3 | 0 | 0 | 100% |
| ella_fans/save_substitute/apps.py | 4 | 0 | 0 | 100% |
| ella_fans/save_substitute/models.py | 7 | 1 | 0 | 86% |
| ella_fans/save_substitute/urls.py | 3 | 0 | 0 | 100% |
| ella_fans/save_substitute/views.py | 39 | 5 | 0 | 87% |
| ella_fans/settings.py | 29 | 0 | 0 | 100% |
| ella_fans/urls.py | 5 | 0 | 0 | 100% |
| ella_fans/users_account/__init__.py | 0 | 0 | 0 | 100% |
| ella_fans/users_account/admin.py | 4 | 0 | 0 | 100% |
| ella_fans/users_account/apps.py | 4 | 0 | 0 | 100% |
| ella_fans/users_account/forms.py | 14 | 0 | 0 | 100% |
| ella_fans/users_account/models.py | 7 | 1 | 0 | 86% |
| ella_fans/users_account/urls.py | 6 | 0 | 0 | 100% |
| ella_fans/users_account/views.py | 34 | 3 | 0 | 91% |
| **tal** | **321** | **31** | **0** | **90%** |

*erage.py v5.5, created at 2021-08-23 15:52 +0200*


```python
from django.views.generic import CreateView
from django.contrib.auth.mixins import LoginRequiredMixin
from django.views.generic import ListView
from django.views.generic.edit import DeleteView
from django.urls import reverse_lazy
from django.shortcuts import redirect
from django.contrib import messages


from nutella_fans.save_substitute.models import Substitute, Product
from nutella_fans.users_account.models import User


class FavorateListView(ListView):
    template_name = 'favorites_list.html'
    model = Substitute
    context_object_name = 'favorite_list'

    def get_queryset(self):
        queryset = super().get_queryset()
        return queryset.filter(user=self.request.user)


class SubtituteSaveView(LoginRequiredMixin, CreateView):
    template_name = 'substitute_list.html'
    model = Substitute
    fields = ['product', 'substitute']

    def post(self, request, *args, **kwargs):
        product = Product.objects.get(
            pk=self.request.POST.get('product_id'))
        substitute = Substitute.objects.filter(
            product_id=product.id, substitute_id=self.request.POST.get('substitute_id'))
        user_id = request.user
        user = User.objects.get(id=user_id.id)
        if not substitute.exists():
            substitute = Substitute.objects.create(product_id=product.id,
                                                   substitute_id=self.request.POST.get('substi
            user.substitutes.add(substitute)
            messages.success(
                request, 'Le produit est bien dans votre liste de favoris')
        else:
            messages.warning(
                request, 'Le produit existe déja dans votre liste de favoris')
        return redirect('favorites_list')


class FavoriteDeleteView(DeleteView):
    model = Substitute
    success_url = reverse_lazy('favorites_list')

    def get(self, request, *args, **kwargs):
        substitute_id = self.kwargs.get('fav.id')
        delete = self.post(request, Substitute.objects.filter(
            substitute_id=substitute_id).delete())
        messages.success(request, 'Le substitut a bien été supprimé')
        return delete
```

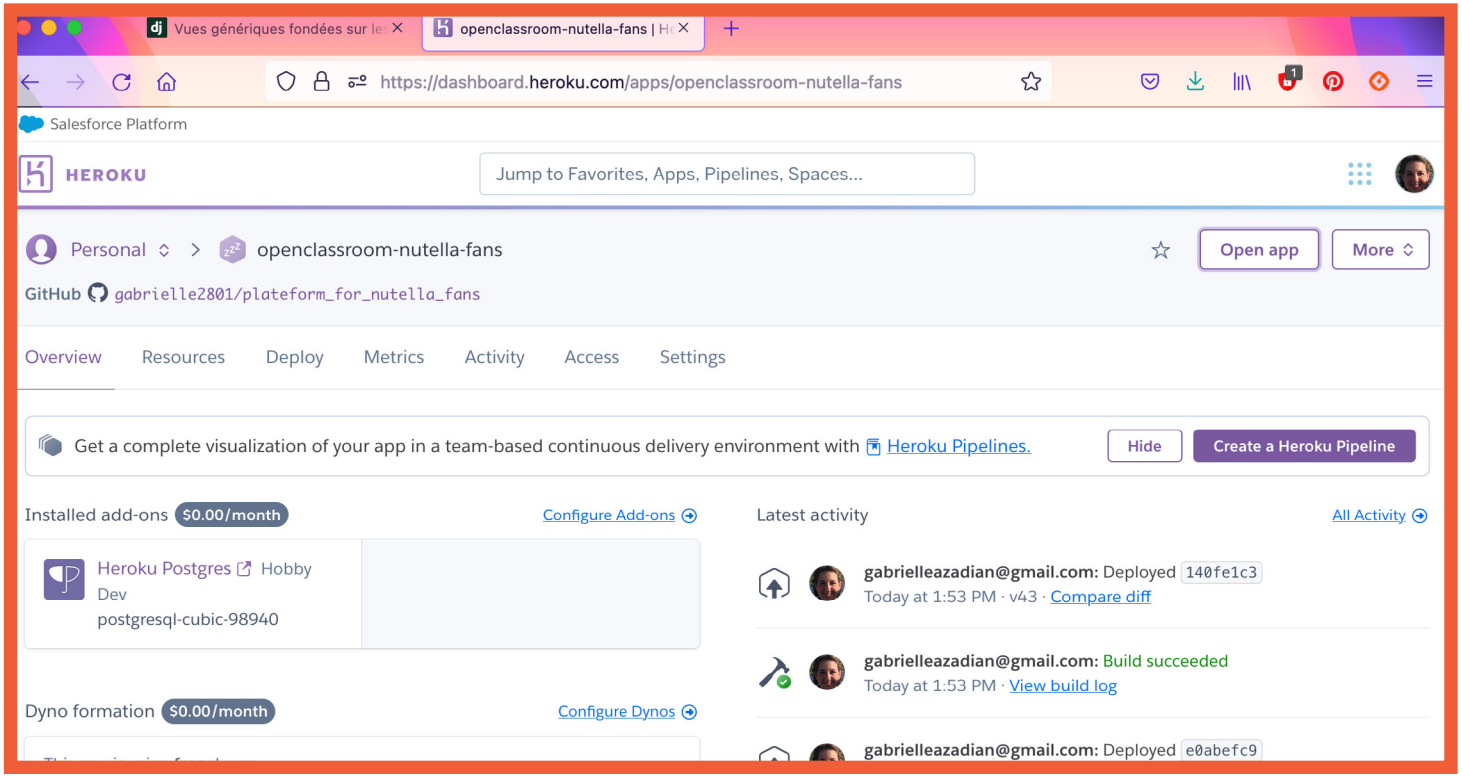
```
16/Sep/2021 16:12:09] "GET /static/assets/img/bg-purbeurre.jpg HTTP/1.1" 200 1933650
16/Sep/2021 16:12:09] "GET /static/assets/favicon.ico HTTP/1.1" 200 679
16/Sep/2021 16:12:14] "GET / HTTP/1.1" 200 21002
16/Sep/2021 16:12:14] "GET /static/css/styles.css HTTP/1.1" 200 203956
16/Sep/2021 16:12:14] "GET /static/js/scripts.js HTTP/1.1" 200 1779
16/Sep/2021 16:12:14] "GET /static/debug_toolbar/css/toolbar.css HTTP/1.1" 200 11461
16/Sep/2021 16:12:14] "GET /static/debug_toolbar/js/toolbar.js HTTP/1.1" 200 10452
16/Sep/2021 16:12:14] "GET /static/debug_toolbar/js/utils.js HTTP/1.1" 200 2988
16/Sep/2021 16:12:14] "GET /static/assets/img/logo_pur_beurre.png HTTP/1.1" 200 11846
16/Sep/2021 16:12:14] "GET /static/assets/img/portfolio/thumbnails/remi-11.png HTTP/1.1" 200 276313
16/Sep/2021 16:12:14] "GET /static/assets/img/portfolio/thumbnails/colette-6.jpg HTTP/1.1" 200 160090
16/Sep/2021 16:12:14] "GET /static/assets/img/portfolio/thumbnails/remy-2.jpg HTTP/1.1" 200 198168
16/Sep/2021 16:12:14] "GET /static/assets/img/bg-purbeurre.jpg HTTP/1.1" 200 1933650
16/Sep/2021 16:12:14] "GET /static/debug_toolbar/css/print.css HTTP/1.1" 200 43
16/Sep/2021 16:12:15] "GET /product_list/?search_product=Biscuit HTTP/1.1" 200 35454
16/Sep/2021 16:12:15,101] - Broken pipe from ('127.0.0.1', 57687)
```

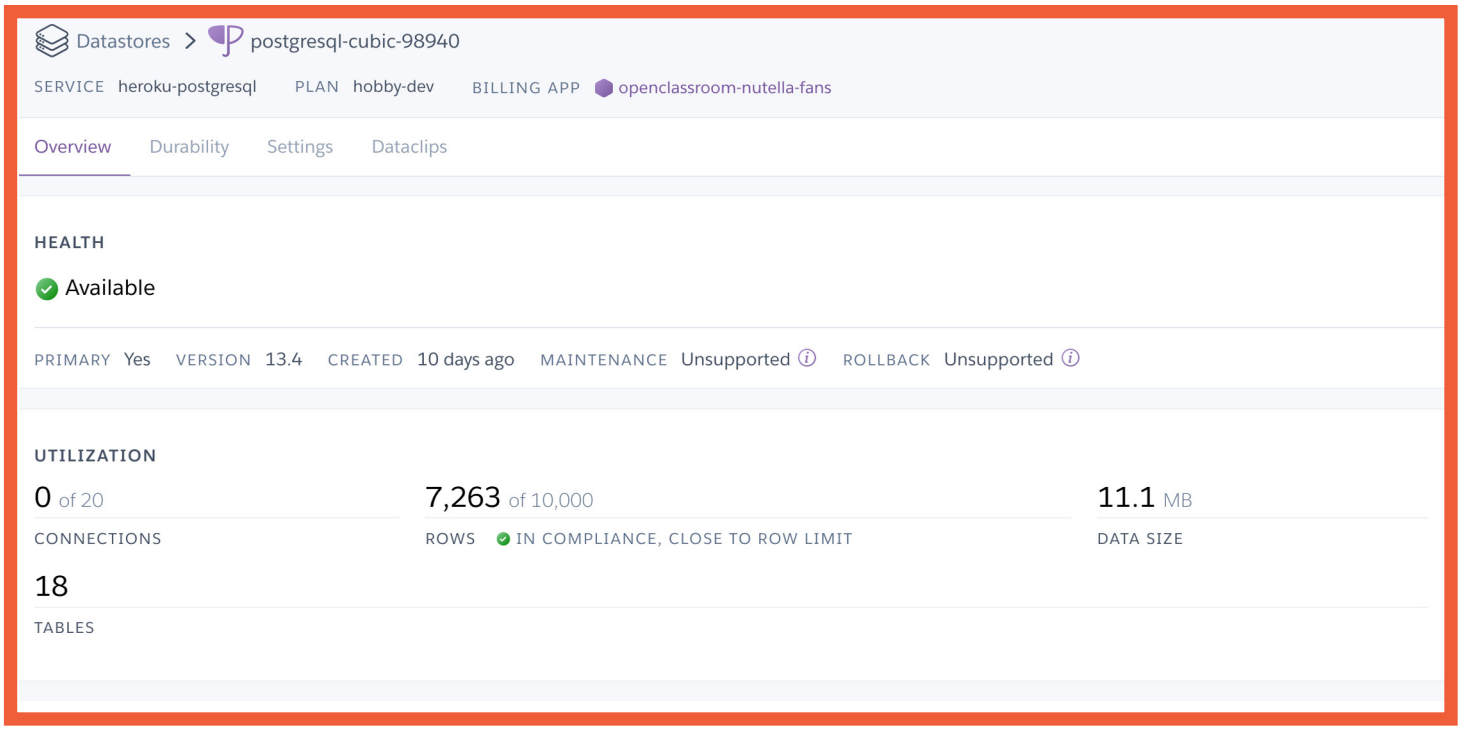
# Heroku --> settings

Several settings
--> local (production)
--> development (deployment)

heroku run bash ou heroku run dans le terminal
**git push heroku main**

**heroku run python3 manage.py migrate**
The process of generating tables within the database.
**heroku run manage.py import_off**
importing data through API calls.




Nutella Fans