NABAJOTH Jérémy

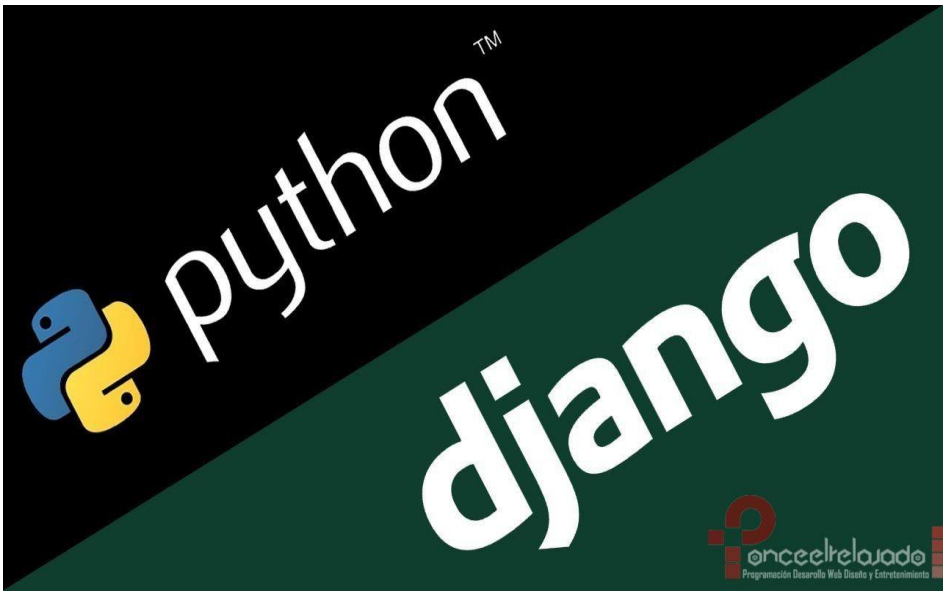
ROTSEN Zoé

CALIXTE Rama

BLANGER Harry

PROJET

COMP ARATEUR DE PRIX



React

PostgreSQL

SOMMAIRE

CAHIER DES CHARGES

Lors des premières sessions avec notre Product Owner qu'est Mr.Régis Géromégnace, nous avons pu définir clairement l'objectif de l'application (participative) web responsive ainsi que les différentes fonctionnalités retenue pour nos Sprint avenir.

En effet, il s'agit d'une application de comparateur de prix des produits du BQP en Guadeloupe en fonction des différentes enseignes.

Ce comparateur permettra de mettre en lumière la différence des prix pour un produit spécifique ou des produits spécifique en fonction de l'enseigne et de leur localisation.

Constituant le Groupe 2 : **Backend/API/Architecture logicielle**, voici donc les missions qui nous ont été confier.

- Proposer une architecture de l'application ainsi que les langages utilisés
- Réfléchir aux différentes fonctions nécessaires à l'application
- Explorer des Framework de création d'API
- Proposez un diagramme de classe des entités
- Mettre en œuvre l'API
- Réaliser les tests.

CONCEPTION

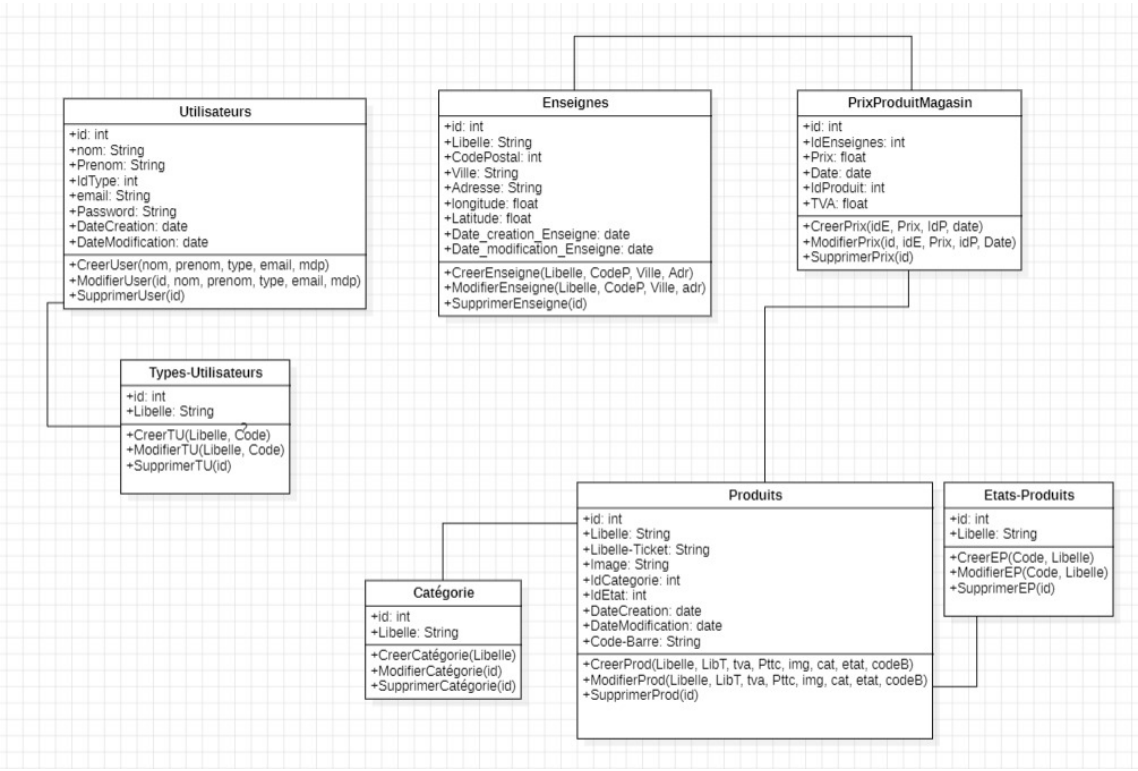
Une fois nos missions clairement établies, nous avons pu nous pencher sur ces problématiques. Pour y répondre, nous avons :

- Dans un premier temps, effectuer un travail de documentation et de recherche afin de chercher le Framework qui nous permettra de développer l'application tout en respectant le cahier des charges ainsi que les applications utilisées dans les autres groupes tels que « UI » et « Base de données ».
- À la fin de ces recherches, plusieurs solutions s'offraient à nous. Nous avons finalement fait le choix du modèle MVC (Modèle, Vue, Contrôleur) pour ce qui concerne l'architecture de l'application.
- Pour ce qui concerne le Framework, nous avons retenu « Django » qui permettra le développement, la prise en charge des applications des autres groupes ainsi que la création d'API notamment avec le module « Django REST Framework »
- Nous avons (en étroite collaboration avec le groupe « Base de données ») effectué un diagramme de classe permettant de visualiser et d'uniformiser les entités que nous devrons établir et avec lesquelles nous serons emmenés à développer les fonctionnalités.

D'autre part, nous avons également opté pour l'approche « Database First » ce qui veut dire que la base de données devait être définie en amont afin que de notre côté, Django nous génère nos modèles de classe et le code.

Tout cela étant fait et toujours à l'aide du groupe **« Base de données »,** nous avons procédé à la création de notre diagramme de classe avec le logiciel StarUML, qui regroupe les différentes entités sur lesquelles nous allons travailler.

Diagramme de classe



Une fois ce diagramme conceptualisé, nous nous sommes réparti les tâches.

Chaque personne du groupe allait s'occuper de développer les méthodes et l'API pour chaque entité.

REALISATION

À la fin du premier sprint, chaque groupe avait présenté leurs travaux ce qui permettait d'éclaircir des zones d'ombres s'il y en avait, de savoir concrètement le rôle et les missions du groupe, et surtout d'être tous sur un même pied d'égalité.

À la fin, nous avions tous et pouvions tous enfin commencer et entrer dans le vif du sujet et pour notre groupe cela se résume au code.

Pour réaliser le projet, nous avons :

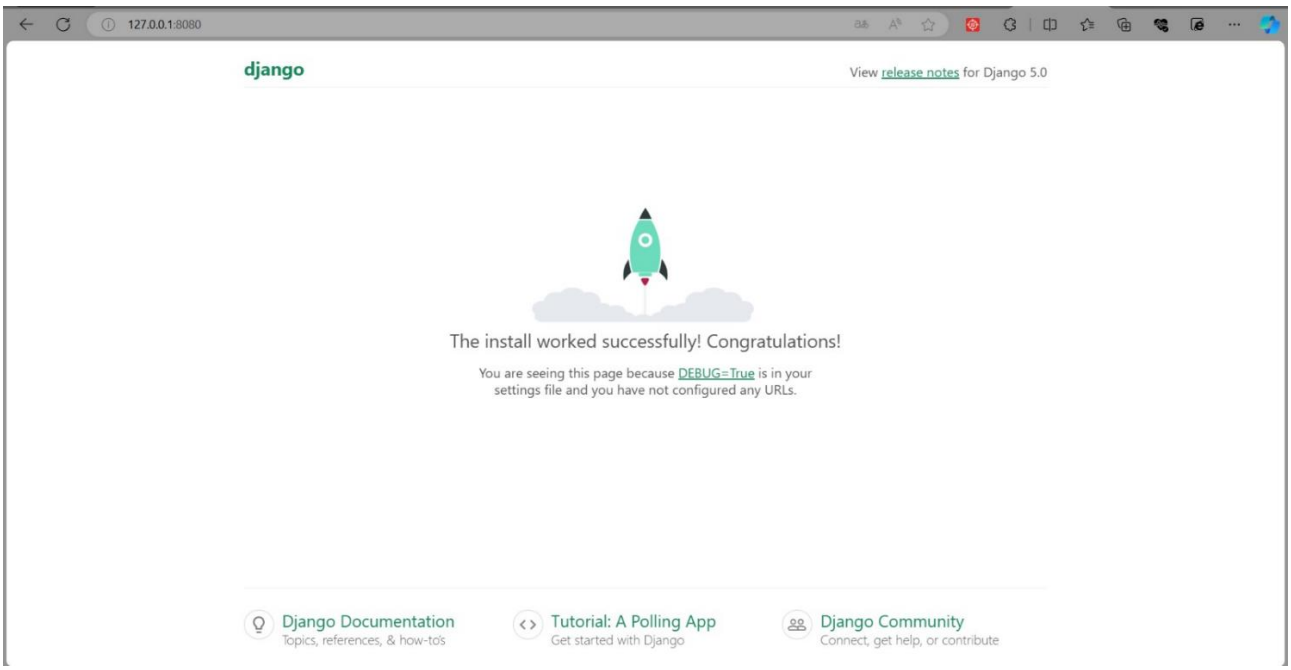
- Installer le Framework Django et les modules annexes
- Installer PostgreSQL
- Développer les méthodes et l'API
- Installer Postman
- Résultats et Tests

Ces installations étaient à faire sur nos machines surtout en ce qui concerne la base de donnéess car aucun serveur nous était mis à disposition.

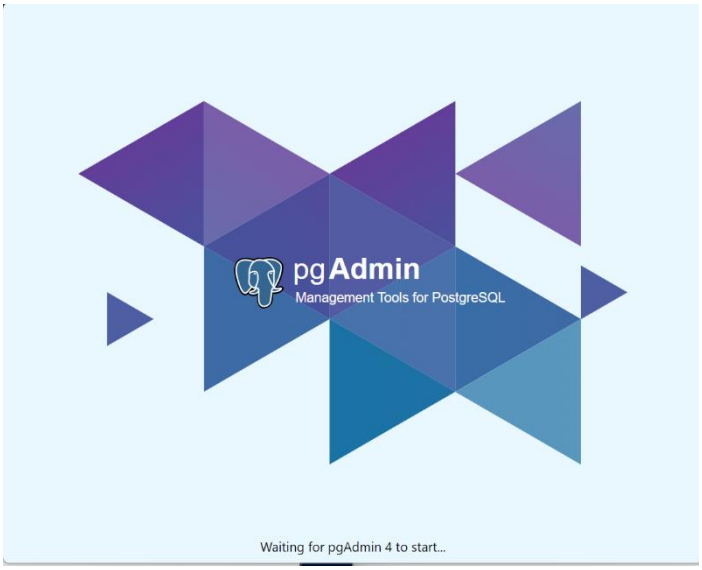
Cependant, grâce à l'utilisation de GitHub, nous avons pu faire du développement collaboratif et s'échanger les nouvelles versions du code à chaque fois qu'un membre du groupe avait terminer leur partie.

Installation Django

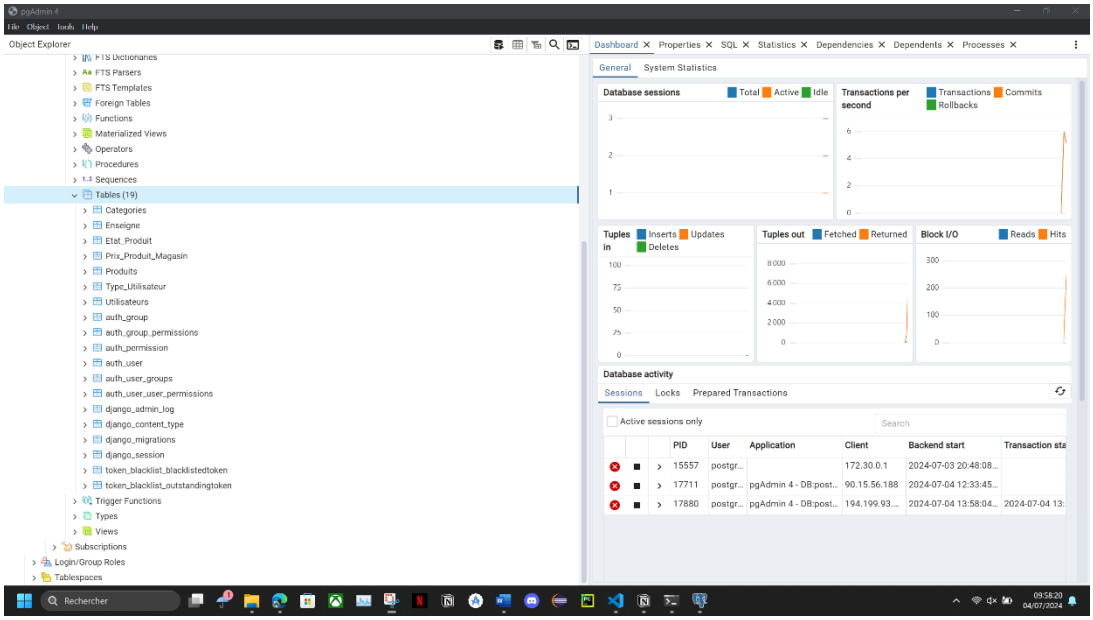
```
PS C:\Users\jeje9\Documents\MIAGE\Cours_NSI\Python\comparateurPrix> pip install django
Requirement already satisfied: django in c:\python312\lib\site-packages (5.0.6)
Requirement already satisfied: asgiref<4,>=3.7.0 in c:\python312\lib\site-packages (from django) (3.8.1)
Requirement already satisfied: sqlparse>=0.3.1 in c:\python312\lib\site-packages (from django) (0.5.0)
Requirement already satisfied: tzdata in c:\python312\lib\site-packages (from django) (2024.1)
PS C:\Users\jeje9\Documents\MIAGE\Cours_NSI\Python\comparateurPrix>
```



Installation PostgreSQL



La base de données



Développement des méthodes et de l'API

Les URL

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('categories/get/', views.CategoriesListAPIView.as_view(), name='categories-list'),
    path('categories/get/<int:id_categorie>/', views.CategoriesByIdListAPIView.as_view(), name='categories-list'),
    path('categories/create/', views.CreateCategoriesListAPIView.as_view(), name='categories-create'),
    path('categories/update/<int:id_categorie>/', views.UpdateCategoriesDetailAPIView.as_view(), name='categories-detail'),
    path('categories/delete/<int:id_categorie>/', views.DeleteCategoriesDetailAPIView.as_view(), name='categories-detail'),
    path('enseignes/get/', views.EnseigneListAPIView.as_view(), name='enseignes-list'),
    path('enseignes/get/<int:id_enseigne>/', views.EnseigneByIdListAPIView.as_view(), name='enseignes-list'),
    path('enseignes/create/', views.CreateEnseigneAPIView.as_view(), name='enseignes-list'),
    path('enseignes/update/<int:id_enseigne>/', views.UpdateEnseigneDetailAPIView.as_view(), name='enseignes-detail'),
    path('enseignes/delete/<int:id_enseigne>/', views.DeleteEnseigneDetailAPIView.as_view(), name='enseignes-detail'),
    path('prixProduitMagasin/get/', views.PrixMagasinProduitListAPIView.as_view(), name='prix_produit_magasin_list'),
    path('prixProduitMagasin/get/<int:idP>/<int:idM>/', views.PrixProduitMagasinByIdsListAPIView.as_view(), name='prix_produit_magasin_list'),
    path('prixProduitMagasin/delete/<int:idP>/<int:idM>/', views.DeletePrixProduitMagasinByIdAPIView.as_view(), name='delete_prix_produit_magasin'),
    path('prixProduitMagasin/create/', views.CreatePrixProduitMagasinAPIView.as_view(), name='create_prix_produit_magasin'),
    path('prixProduitMagasin/update/<int:idP>/<int:idM>/', views.UpdatePrixProduitMagasinAPIView.as_view(), name='update_prix_produit_magasin'),
    path('etatProduit/get/', views.EtatProduitListAPIView.as_view(), name='etatProduit-list'),
    path('etatProduit/get/<int:id_etat>/', views.EtatProduitByIdListAPIView.as_view(), name='etatProduit-list'),
    path('etatProduit/delete/<int:id_etat>/', views.DeleteEtatProduitByIdListAPIView.as_view(), name='etatProduit-delete'),
    path('etatProduit/create/', views.CreateEtatProduitAPIView.as_view(), name='etatProduit-create'),
    path('etatProduit/update/<int:id_etat>/', views.UpdateEtatProduitAPIView.as_view(), name='etatProduit-update'),
    path('produits/get/', views.ProduitsListAPIView.as_view(), name='produits-list'),
    path('produits/get/<int:id_produit>/', views.ProduitsByIdListAPIView.as_view(), name='produit-list'),
    path('produits/get/nom/<str:libelle>/', views.ProduitsByNameListAPIView.as_view(), name='produit-list'),
    path('produits/get/categorie/<int:id_categorie>/', views.ProduitsByIdCategorieListAPIView.as_view(), name='produits-list'),
    path('produits/delete/<int:id_produit>/', views.DeleteProduitsByIdListAPIView.as_view(), name='produits-delete'),
    path('produits/create/', views.CreateProduitsAPIView.as_view(), name='produits-create'),
    path('produits/update/<int:id_produit>/', views.UpdateProduitsAPIView.as_view(), name='produits-update'),
    path('utilisateurs/create/', views.CreateUtilisateursAPIView.as_view(), name='utilisateurs-create'),
    path('utilisateurs/create/lambda_connecte/', views.CreateLambdaConnecteAPIView.as_view(), name='utilisateurs-create'),
    path('utilisateurs/create/certifie/', views.CreateCertifieAPIView.as_view(), name='utilisateurs-create'),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('swagger/', schema_view.with_ui('swagger', cache_timeout=0), name='schema-swagger-ui'),
    path('redoc/', schema_view.with_ui('redoc', cache_timeout=0), name='schema-redoc'),

]
```

Structure d’échange des données

```python
from rest_framework import serializers
from .models import Categories
from .models import EtatProduit
from .models import Produits
from .models import Enseigne, PrixProduitMagasin
from .models import PrixProduitMagasin
from django.contrib.auth.models import User
from rest_framework_simplejwt.serializers import TokenObtainPairSerializer
from rest_framework_simplejwt.views import TokenObtainPairView

class CategoriesSerializer(serializers.ModelSerializer):
    class Meta:
        model = Categories
        fields = ['id_categorie', 'nom_categorie']

        extra_kwargs = {
            'nom_categorie': {'required': True},
        }


class EnseigneSerializer(serializers.ModelSerializer):
    class Meta:
        model = Enseigne
        fields = [
            'id_enseigne', 'libelle', 'adresse', 'code_postal', 'ville',
            'longitude', 'latitude', 'date_creation_enseigne', 'date_modif_enseigne'
        ]
        extra_kwargs = {
            'libelle': {'required': True},
            'adresse': {'required': True},
            'code_postal': {'required': True},
            'ville': {'required': True},
        }

class PrixProduitMagasinSerializer(serializers.ModelSerializer):
    class Meta:
        model = PrixProduitMagasin
        fields = [
            'id_prix_produit_magasin',
            'id_produit',
            'id_magasin',
            'prix',
            'date_creation_prix_produit_magasin',
            'date_modif_prix_produit_magasin'
        ]

        extra_kwargs = {
            'id_produit': {'required': True},
            'id_magasin': {'required': True},
            'prix': {'required': True},
        }

class EtatProduitSerializer(serializers.ModelSerializer):
    class Meta:
        model = EtatProduit
        fields = ['id_etat', 'libelle_etat']

        extra_kwargs = {
            'libelle_etat': {'required': True},
        }
```

Modèle de classe

```python
class Categories(models.Model):
    id_categorie = models.AutoField(db_column='ID_categorie', primary_key=True)  # Field name made lowercase.
    nom_categorie = models.CharField(db_column='Nom_categorie', max_length=255, blank=True, null=True)  # Field name made lowercase.

    class Meta:
        managed = False
        db_table = 'Categories'


class Enseigne(models.Model):
    id_enseigne = models.AutoField(db_column='ID_enseigne', primary_key=True)  # Field name made lowercase.
    libelle = models.CharField(db_column='Libelle', max_length=127, blank=True, null=True)  # Field name made lowercase.
    adresse = models.CharField(db_column='Adresse', max_length=255, blank=True, null=True)  # Field name made lowercase.
    code_postal = models.IntegerField(db_column='Code_postal', blank=True, null=True)  # Field name made lowercase.
    ville = models.CharField(db_column='Ville', max_length=50, blank=True, null=True)  # Field name made lowercase.
    longitude = models.DecimalField(db_column='Longitude', max_digits=65535, decimal_places=65535, blank=True, null=True)  # Field name made lowercase.
    latitude = models.DecimalField(db_column='Latitude', max_digits=65535, decimal_places=65535, blank=True, null=True)  # Field name made lowercase.
    date_creation_enseigne = models.DateField(db_column='Date_creation_Enseigne', blank=True, null=True)  # Field name made lowercase.
    date_modif_enseigne = models.DateField(db_column='Date_modif_Enseigne', blank=True, null=True)  # Field name made lowercase.

    class Meta:
        managed = False
        db_table = 'Enseigne'


class EtatProduit(models.Model):
    id_etat = models.AutoField(db_column='ID_etat', primary_key=True)  # Field name made lowercase.
    libelle_etat = models.CharField(db_column='Libelle_etat', max_length=255, blank=True, null=True)  # Field name made lowercase.

    class Meta:
        managed = False
        db_table = 'Etat_Produit'


class PrixProduitMagasin(models.Model):
    id_prix_produit_magasin = models.AutoField(db_column='ID_prix_Produit_Magasin', primary_key=True)  # Field name made lowercase.
    id_produit = models.ForeignKey('Produits', models.DO_NOTHING, db_column='ID_produit', blank=True, null=True)  # Field name made lowercase.
    id_magasin = models.ForeignKey(Enseigne, models.DO_NOTHING, db_column='ID_magasin', blank=True, null=True)  # Field name made lowercase.
    prix = models.DecimalField(db_column='Prix', max_digits=65535, decimal_places=65535, blank=True, null=True)  # Field name made lowercase.
    date_creation_prix_produit_magasin = models.DateField(db_column='Date_creation_prix_Produit_Magasin', blank=True, null=True)  # Field name made lowercase.
    date_modif_prix_produit_magasin = models.DateField(db_column='Date_modif_prix_Produit_Magasin', blank=True, null=True)  # Field name made lowercase.
    tva = models.DecimalField(db_column='TVA', max_digits=65535, decimal_places=65535, blank=True, null=True)  # Field name made lowercase.

    class Meta:
        managed = False
        db_table = 'Prix_Produit_Magasin'


class Produits(models.Model):
    id_produit = models.AutoField(db_column='ID_produit', primary_key=True)  # Field name made lowercase.
    libelle = models.CharField(db_column='Libelle', max_length=255, blank=True, null=True)  # Field name made lowercase.
    libelle_ticket = models.CharField(db_column='Libelle_Ticket', max_length=255, blank=True, null=True)  # Field name made lowercase.
    code_barre = models.CharField(db_column='Code_barre', max_length=255, blank=True, null=True)  # Field name made lowercase.
    id_categorie = models.ForeignKey(Categories, models.DO_NOTHING, db_column='ID_categorie', blank=True, null=True)  # Field name made lowercase.
    image = models.CharField(db_column='Image', max_length=255, blank=True, null=True)  # Field name made lowercase.
    id_etat = models.ForeignKey(EtatProduit, models.DO_NOTHING, db_column='Id_Etat', blank=True, null=True)  # Field name made lowercase.
    date_creation_produit = models.DateField(db_column='Date_creation_Produit', blank=True, null=True)  # Field name made lowercase.
    date_modif_produit = models.DateField(db_column='Date_modif_Produit', blank=True, null=True)  # Field name made lowercase.

    class Meta:
        managed = False
        db_table = 'Produits'
```

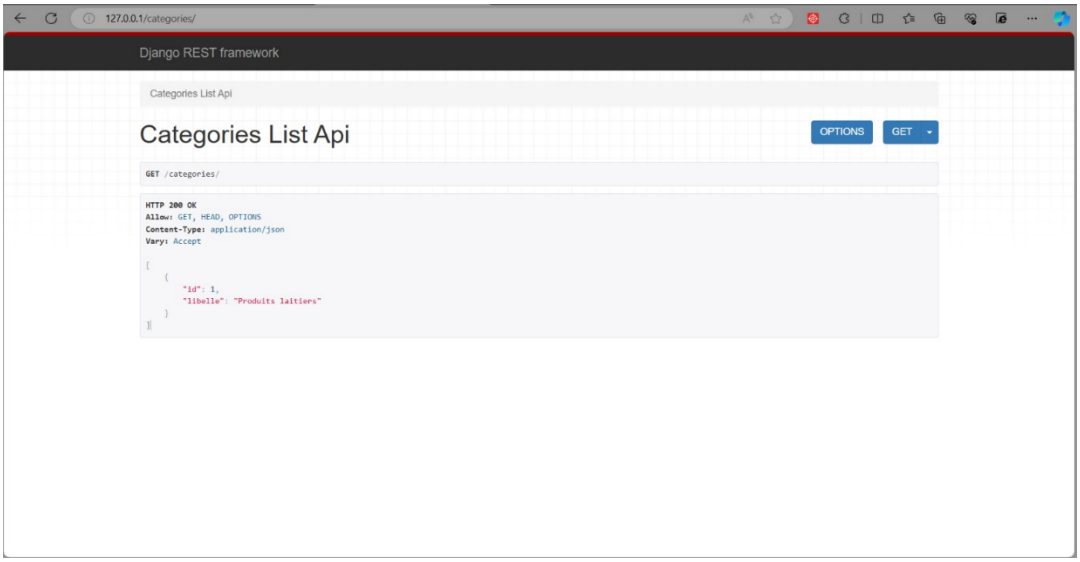
Liaison des URL aux méthodes de l'API

```python
#Gestion des permissions
class EstLambdaConnecte(BasePermission):
    #Permission personnalisée pour vérifier si l'utilisateur appartient au groupe lambda connecté
    def has_permission(self, request, view):
        # Vérifier si l'utilisateur est connecté
        if not request.user.is_authenticated:
            return False

        # Vérifier si l'utilisateur appartient au groupe spécifique
        group_name = 'lambda_connecté'
        try:
            group = Group.objects.get(name=group_name)
        except Group.DoesNotExist:
            return False

        return request.user.groups.filter(name=group_name).exists()

class EstCertifie(BasePermission):
    #Permission personnalisée pour vérifier si l'utilisateur appartient au groupe lambda connecté
    def has_permission(self, request, view):
        # Vérifier si l'utilisateur est connecté
        if not request.user.is_authenticated:
            return False

        # Vérifier si l'utilisateur appartient au groupe spécifique
        group_name = 'certifié'
        try:
            group = Group.objects.get(name=group_name)
        except Group.DoesNotExist:
            return False

        return request.user.groups.filter(name=group_name).exists()

class EstAdministrateur(BasePermission):
    #Permission personnalisée pour vérifier si l'utilisateur appartient au groupe administrateur
    def has_permission(self, request, view):
        # Vérifier si l'utilisateur est connecté
        if not request.user.is_authenticated:
            return False

        # Vérifier si l'utilisateur appartient au groupe spécifique
        group_name = 'administrateur'  # Remplacez par le nom de votre groupe
        try:
            group = Group.objects.get(name=group_name)
        except Group.DoesNotExist:
            return False

        return request.user.groups.filter(name=group_name).exists()
```

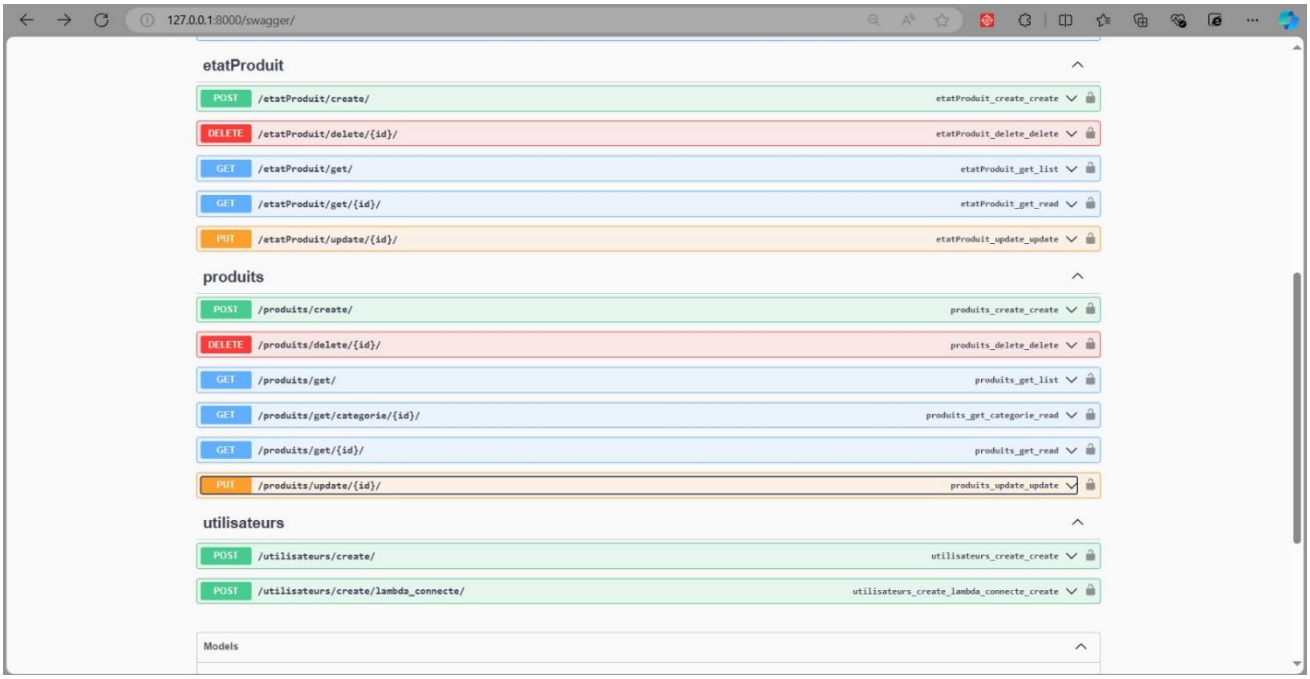
Installer Postman

Résultats et Tests

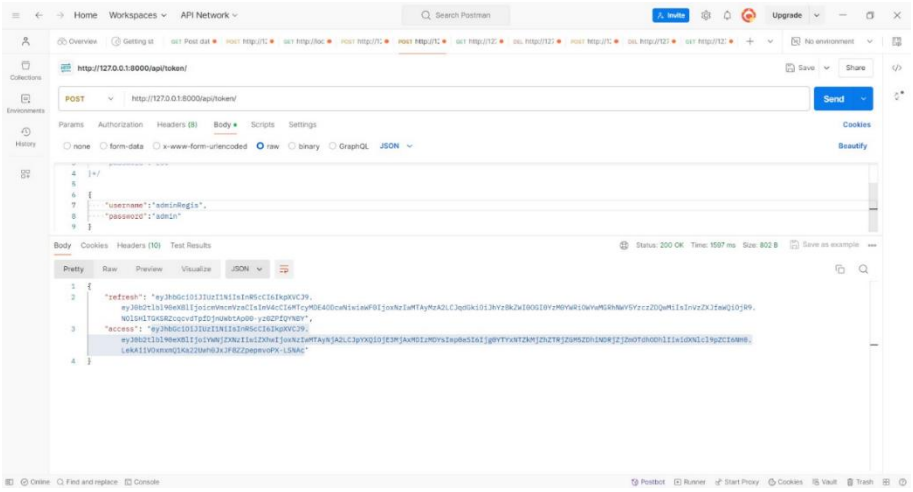
Sur Django REST Framework

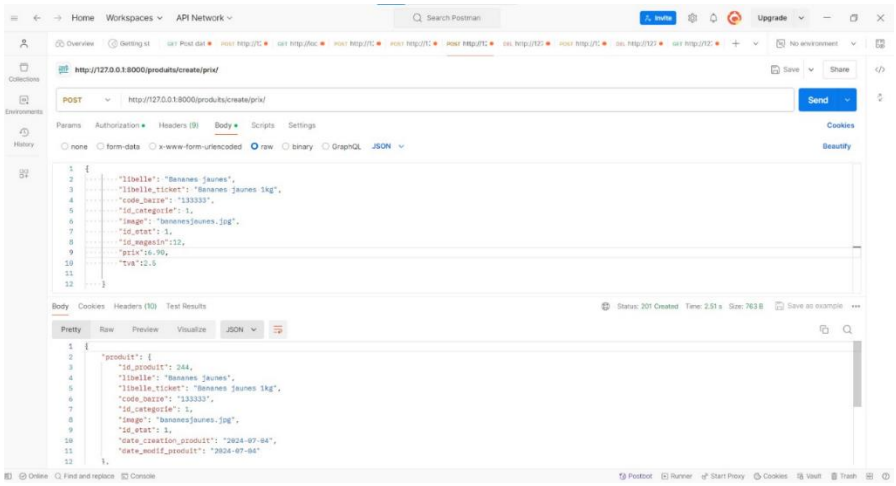


Sur l'interface de documentation d'API (Swagger)

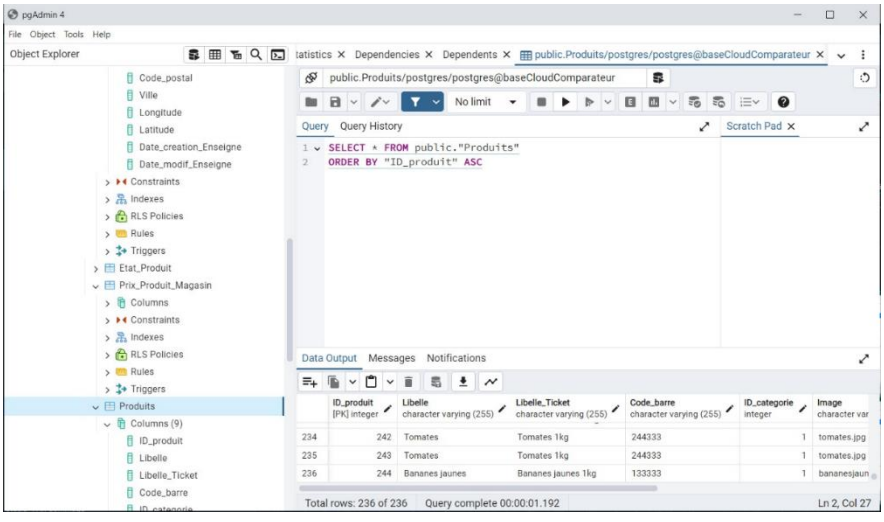


Sur Postman





Sur la base de données



INTEGRATION

Cette année, l'organisation du projet a été grandement modifiée par notre Scrum Master, Mr G.Rinaldo. Pour la conception de ce projet, nous nous sommes réparti les différents composants de l'application en groupe.

En effet, 4 groupes ont été formé pour la réalisation du projet. Dans un premier temps, nous avons le groupe 1 « UI » charger du maquettage, du design et du Front-End.

Ensuite le groupe 2 « Backend/API » charger de la conception de l'architecture de l'application ainsi que développer les fonctionnalités.

En outre, le groupe 3 « Base de données » charger de la réalisation du modèle Conceptuel des Données et de fournir cette base de données.

Enfin, nous avons le groupe 4 « Test/Intégration/Infrastructure » charger de la création d'un jeu de données afin d'alimenter la base de données, ainsi que la réalisation de tests afin d'assurer le bon fonctionnement de l'application.

Compte tenu de cette nouvelle disposition, nous avons été amenés à interagir avec différents groupes, tout au long du développement de l'application. Parmi ces groupes, nous avons :

- **Base de données** : Tout d'abord, nous étions en étroite collaboration, et ce, dès le début car nous devions uniformiser nos classes avec leur MCD afin d'avoir la même modélisation. De plus, ils l'obtention de la Base de données était primordial pour le commencement du développement Backend étant donné que nous avions opté pour l'approche « DataBase First ». Enfin, nous communiquions lors de mise à jour de la BD.

- **UI** : Ensuite, nous avons également fait appel à l'équipe UI pour donner suite à des propositions de fonctionnalités, leur facilitant la manipulation d'éléments, l'affichage des données ou encore l'utilisation de requêtes envers notre API. Par ailleurs, la base de données n'étant pas accessible en ligne immédiatement, nous leur avons permis d'effectuer des premiers jets de tests de notre API sur leur interface.

- **Test/Intégration/Infrastructure :** Enfin, sur nos temps libres, nous avons aidé l'équipe « Test/Intégration/Infrastructure » concernant la conception d'un code python permettant la conversion de données au format csv en requête SQL leur permettant d'implémenter ces données dans la base de données.

ORGANISATION DE PROJET

Afin de gérer l'organisation de nos tâches respectives, l'avancé de chacun et plus largement, l'avancée du projet, nous avons utilisé les outils :

- **WhatsApp :** WhatsApp a été utilisé pour s'échanger des documents et se tenir informé des avancées de chacun.

- **Discord :** Discord occupant le centre de notre organisation, car c'est avec cet outil que nous avons pu organiser chaque week-end des réunions concrètes avec partage d'écran pour réellement s'intéresser, et prendre connaissance des avancées de tous. C'est aussi au cours de ces réunions que nous avons pu vraiment effectuer le management de projet.

- **GitHub :** GitHub a été utilisé dans un premier pour de faire une gestion de versionning. De plus, il nous a permis d'effectuer du développement collaboratif, car nous avons été en mesure de s'échanger les mises à jour du code et des fonctionnalités développés par chacun. Aussi, il rassemblera nos différents livrables tout au long du projet.

EQUIPE

- L'équipe Backend se compose de 4 personnes :
- - BLANGER Harry → Backend/API classe Catégorie_Produit et Enseignes
- - CALIXTE Rama → Backend/API classe Prix_Produit_Magasin
- -NABAJOTH Jérémy → Backend/API classe Utilisateurs et Types_Utilisateurs
- ROTSEN Zoé → Backend/API classe Produits, Etat_Produit, Authentification API

AVIS ET APPORT

Apports Techniques

Ce projet, par sa complexité et son organisation, nous a apporté à la fois sur le plan technique que personnel. Premièrement, nous avons découvert la méthode Agile (SCRUM) notamment le concept de sprint grâce à Mr R.Géromégnace, de plus nous avons pu découvrir le Framework Django avec ces méthodes et modules pour les API et l'authentification.

Apports personnels

Grâce à l'organisation particulière du projet et sa division en plusieurs groupes, nous avons développé une attache pour le travail collaboratif et le travail d'équipe. De plus, il nous a également permis de développer notre autonomie et notre sens de l'organisation, sans oublier le soutiens et l'entraide.

Avis

Ce projet est très prometteur, étant donné le contexte économique actuel. Un comparateur de prix en Guadeloupe est un projet totalement pertinent. Malheureusement, nous ne pourrons pas poursuivre son développement, mais en garderons tout de même une expérience très enrichissante pour chacun de nous.

LIVRABLES

L'entièreté de nos livrables ce trouve à l'adresse : GitHub - ZoeRotsen/ComparateurDePrix

Dans ce répertoire, vous trouverez :

- Les dossiers « comparateur » et « comparateurPrix » représentant le cœur de notre projet avec l'ensemble des fichiers python.
- Le dossier « Annexes » contenant les comptes-rendus des séances, la documentation et fiche technique du projet, ainsi que le PowerPoint de présentation du projet.