

Comparateur de prix

Groupe Backend



Sommaire

- Cahier des charges
- Conception
- Réalisations
- Intégration
- Organisation
- Livrables
- Bilan

Cahier des charges

Comparateur de Prix

Le projet du comparateur de prix est un projet mené par **Monsieur Régis GEROMEGNACE** en collaboration avec l'association **CLCV**. Il s'agit d'une application de web permettant de comparer le 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 spécifiques en fonction de l'enseigne et de leur localisation.

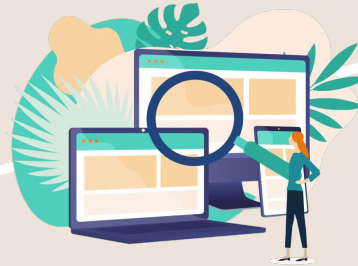
Cahier des charges

Groupe Backend

Le groupe **Backend** est en charge du développement de l'API permettant la mise en oeuvre des interactions entre le frontend et la base de données. Pour accomplir notre mission, nous avons réalisé plusieurs tâches :

- 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 oeuvre l'API
- Réaliser les tests

Conception



Recherche du framework

Lors de nos recherches, trois frameworks sont sortis du lot :

- Flask
- FastAPI
- Django

Finalement notre choix s'est porté sur django pour sa **simplicité d'utilisation**, les **modules d'authentification** et les **outils relatifs à la documentation**

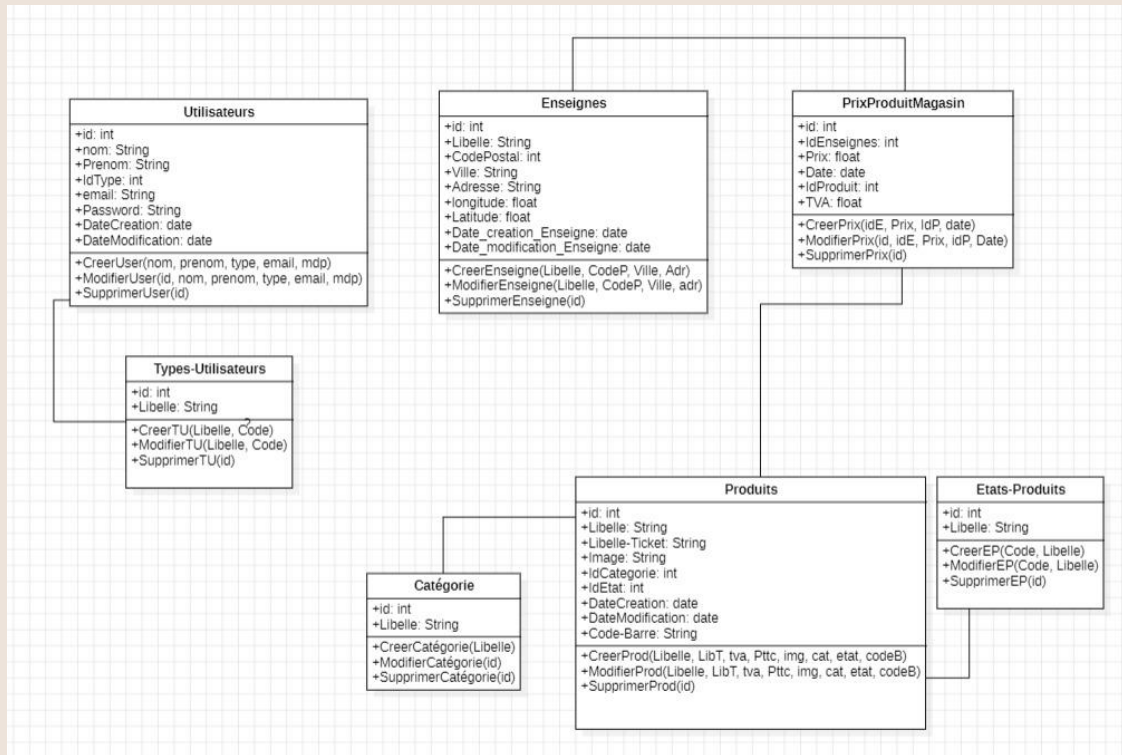


Flask



Réalisations du diagramme de classes

Pour réfléchir à la conception de l'API, nous avons réaliser un diagrammes de classes représentant les classes et leurs attributs, leurs méthodes et les liens entres les différentes classes.



Collaboration avec les équipes

Lors de la phase de conception, nous avons interagi avec l'équipe base de données, pour obtenir un diagramme de classes cohérent avec la base de données . Il s'agissait d'un échange d'idées et d'une mise en commun de nos travaux.



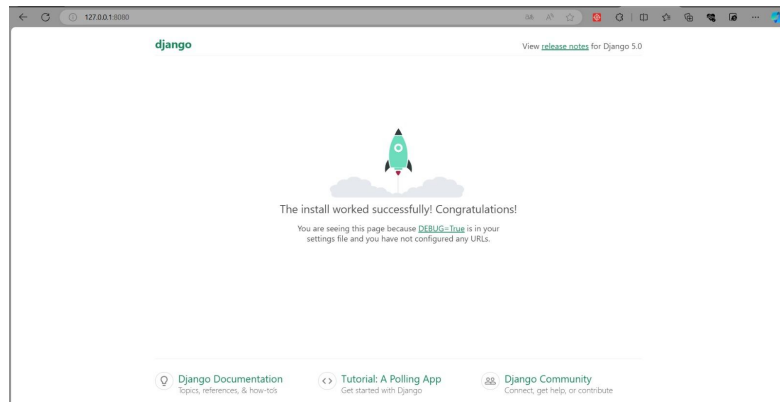
Réalisations

Lors de la phase de réalisation, nous avons procédé à :

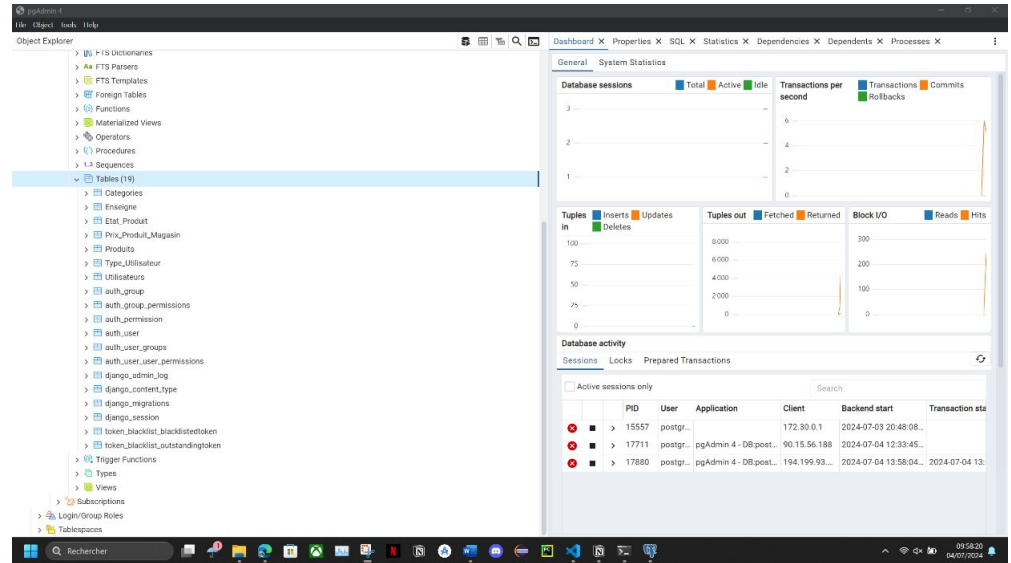
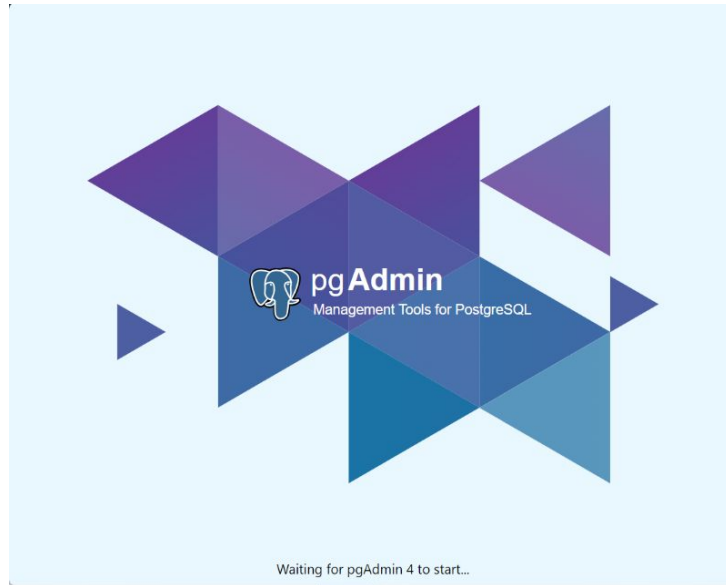
- Installation de Django et des modules annexes
- Installation PostgreSQL et importation du script de base de données
- Développement des méthodes de l'API
- Installation de Postman
- Mise en oeuvre des tests

Installation de Django et des modules annexes

```
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 de PostgreSQL



Importation du script SQL

```
comparateur.sql
29
30 CREATE TABLE "Categories" (
31     "ID_categorie" SERIAL PRIMARY KEY,
32     "Nom_categorie" VARCHAR(255)
33 );
34
35 CREATE TABLE "Etat_Produit" (
36     "ID_etat" SERIAL PRIMARY KEY,
37     "Libelle_etat" VARCHAR(255)
38 );
39
40 CREATE TABLE "Produits" (
41     "ID_produit" SERIAL PRIMARY KEY,
42     "Libelle" VARCHAR(255),
43     "Libelle_Ticket" VARCHAR(255),
44     "PrixTTC" DECIMAL,
45     "Code_barre" VARCHAR(255),
46     "ID_categorie" INTEGER,
47     "TVA" DECIMAL,
48     "Image" VARCHAR(255),
49     "Id_Etat" INTEGER,
50     "Date_creation_Produit" DATE,
51     "Date_modif_Produit" DATE,
52     FOREIGN KEY ("ID_categorie") REFERENCES "Categories" ("ID_categorie"),
53     FOREIGN KEY ("Id_Etat") REFERENCES "Etat_Produit" ("ID_etat")
54 );
55
56
57 CREATE TABLE "Prix_Produit_Magasin" (
58     "ID_prix_Produit_Magasin" SERIAL PRIMARY KEY,
```

Développement des méthodes de l'API

```
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'
```

Fichier models.py généré grâce à l'approche DatabaseFirst. Nous avons modifié les classes y ajouter des méthodes.

Développement des méthodes de l'API

```
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},
        }
```

Fichier serializers.py, qui permet de convertir les objets Python en JSON

Développement des méthodes de l'API

```
#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' # Remplacer 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()
```

Fichier **views.py** qui permet de réaliser les méthodes de réponses aux requêtes HTTP. On y retrouve également les méthodes d'authentification.













Développement des méthodes de l'API

```
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.PrixProduitMagasinByIdListAPIView.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.ProduitListAPIView.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.ProduitsByNomListAPIView.as_view(), name='produit-list'),
    path('produits/get/categorie/<int:id_categorie>/', views.ProduitsByCategorieListAPIView.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_connecto/', views.CreateLambdaConnectoAPIView.as_view(), name='utilisateurs-create'),
    path('utilisateurs/create/certifie/', views.CreateCertifieAPIView.as_view(), name='utilisateurs-create'),
    path('api/token/', views.TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', views.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'),
]
```

Fichier `urls.py` qui permet de définir les différentes urls de l'API et d'y associer une méthode de réponse.

Authentication

Lors de la mise en place de l'authentification, nous avons eu quelques difficultés à lier Django avec notre classe utilisateurs. Cependant, Django met à disposition un ensemble de classes pour gérer ce module notamment en permettant la gestion des utilisateurs, des tokens, des groupes d'utilisateurs, des permissions etc.

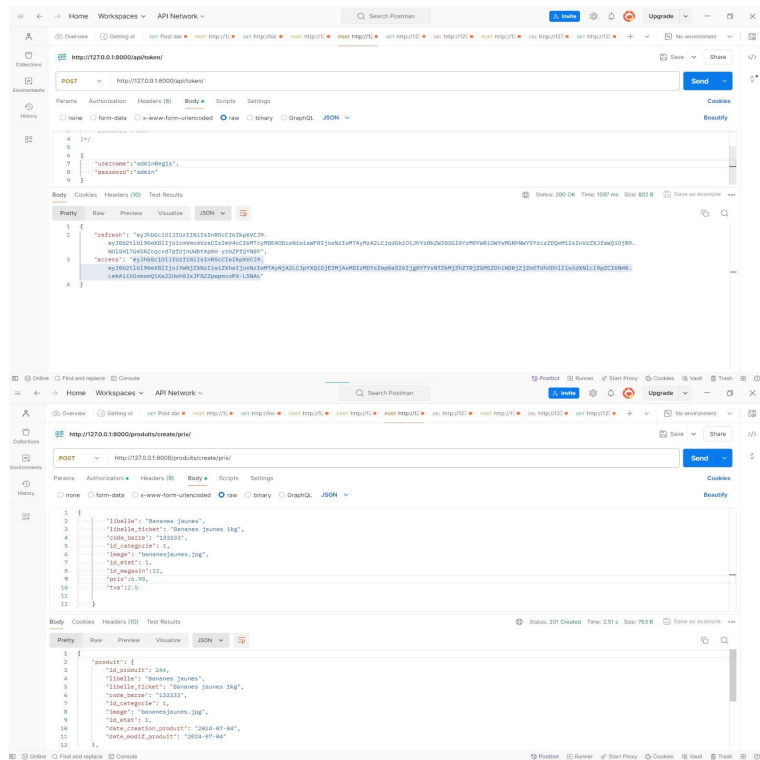
- >  auth_group
- >  auth_group_permissions
- >  auth_permission
- >  auth_user
- >  auth_user_groups
- >  auth_user_user_permissions
- >  django_admin_log
- >  django_content_type
- >  django_migrations
- >  django_session
- >  token_blacklist_blacklistedtoken
- >  token_blacklist_outstandingtoken

Installation de Postman

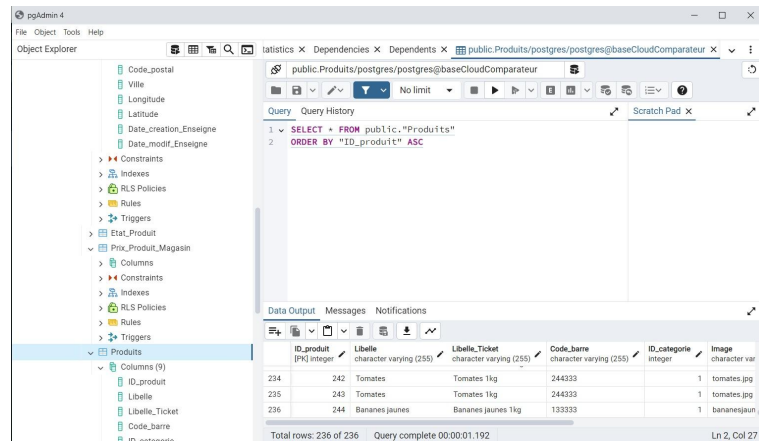


Postman est un outil qui permet de créer, tester des requêtes HTTP et d'inspecter les réponses.

Mise en oeuvre des tests



Test de création d'un produit avec le procédé d'authentification



Intégration

Comme énoncé plus tôt, notre groupe a été en étroite collaboration avec le groupe **base de données** pour la réalisation du diagrammes de classes, mais aussi lors du déploiement de la base sur le serveur. Ainsi, nous pu essayer avec leur aide la connexion de notre API avec la **base distante**. Nous avons également travaillé avec le groupe UI. En effet, nous avons pu communiquer avec ces-derniers et ainsi recueillir leurs recommandations, notamment pour l'ajout d'une requête unique permettant l'ajout d'un produit et d'un prix pour le produit dans un magasin donné. Par ailleurs, pour que l'équipe UI intègre notre API et la teste, nous avons dû installer l'API sur leurs postes.

Organisation

Concernant notre organisation, nous avons choisi d'attribuer à chaque membres du groupe les classes pour lesquelles réaliser les requêtes.

CALIXTE Rama :

- Prix Produit Magasin

BLANGER Harry :

- Catégorie
- Enseigne

NABAJOTH Jérémy :

- Utilisateur
- Type Utilisateur

ROTSSEN Zoé :

- Etat Produit
- Produit
- Modules d'authentification

De plus, pour permettre un travail collaboratif permettant de gérer le versioning nous avons mis en place un Git.

Livrables

Durant ce projet, nous avons produits différents livrables :

- API
- Fiche installation de Django et ses modules
- Documentation du projet

Tous nos livrables sont disponibles sur notre **Git**.

Lien : [ZoeRotsen/CompareurDePrix \(github.com\)](https://github.com/ZoeRotsen/CompareurDePrix)

Bilan

