

Documentation API-REST DJANGO

PROJET : Comparateur de prix

INSTALLATION

Avant de pouvoir créer l'API, nous avons tout d'abord commencé par installer les outils nécessaires. Pour installer Django, en ligne de commande, nous avons tapé les commandes suivantes :

```
python -m pip install Django (installation du module django)
python -m django --version (vérifier le succès de l'installation)
```

CRÉATION DU PROJET

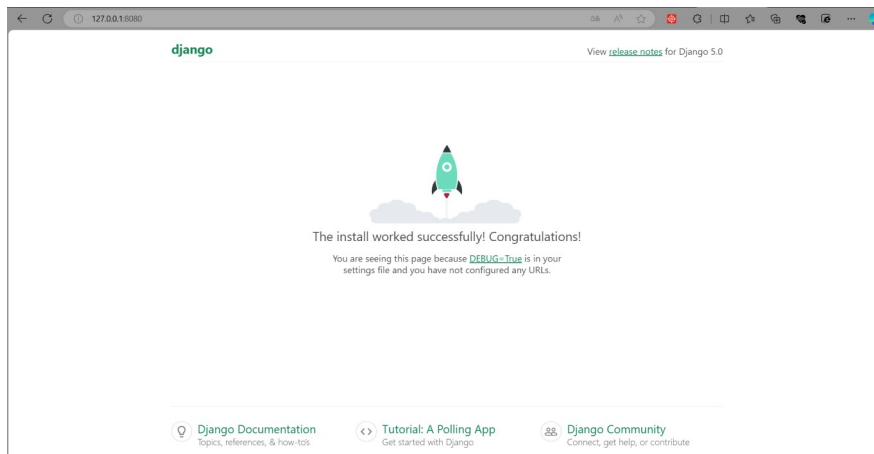
Par la suite, nous avons procédé à la création du projet, soit le répertoire.

```
python -m django startproject comparateur (crée le projet comparateur)
python manage.py runserver (permet de vérifier que le projet fonctionne correctement)
```

A la suite de cette commande, on a obtenu url vers la page d'accueil de l'application, qui permettait ainsi de vérifier le bon fonctionnement du serveur.

```
You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
June 12, 2024 - 13:57:00
Django version 5.0.4, using settings 'comparateur.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Sortie de la commande `python manage.py runserver`



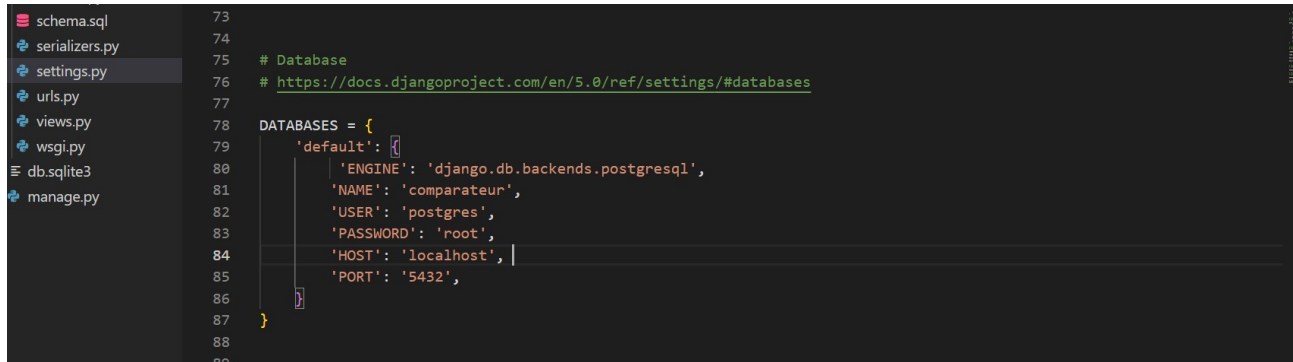
Page d'accueil du projet

Une fois le projet créé, il faudra démarrer une application.

python manage.py startapp comparateur

Ensuite, pour tester l'approche « Database first », j'ai tout d'abord créé ma base de données, crée le script SQL que j'ai ensuite exécuter dans Postgre.

Une fois la base de données correctement configurée, j'ai modifier les paramètres de Django, pour permettre la connexion à la base de données.



```
73
74
75 # Database
76 # https://docs.djangoproject.com/en/5.0/ref/settings/#databases
77
78 DATABASES = {
79     'default': {
80         'ENGINE': 'django.db.backends.postgresql',
81         'NAME': 'comparateur',
82         'USER': 'postgres',
83         'PASSWORD': 'root',
84         'HOST': 'localhost',
85         'PORT': '5432',
86     }
87 }
88
89
```

Configuration de la connexion à la base de données

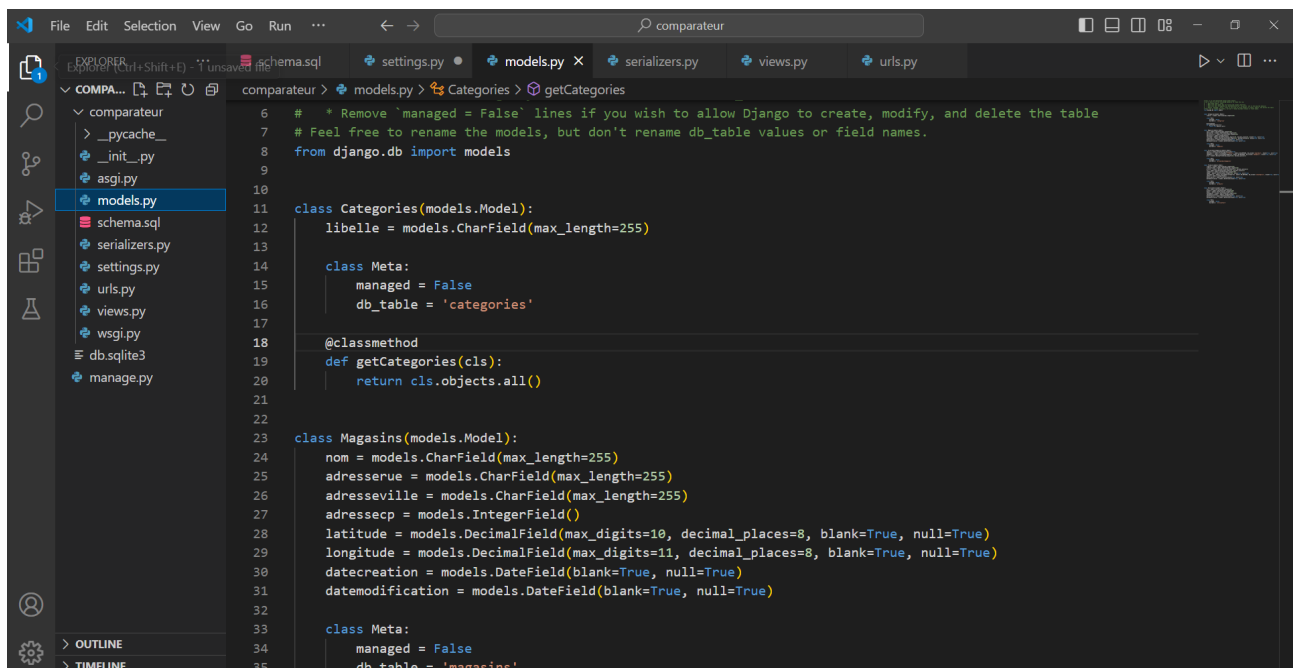
Pour mettre à jour le fichier modèle de django, il faudra lancer les commandes :

python manage.py inspectdb >comparateur/models.py

python manage.py makemigrations

python manage.py migrate

Ces commandes auront permis de créer le fichier models.py et d'y ajouter toutes les classes de la base de données.



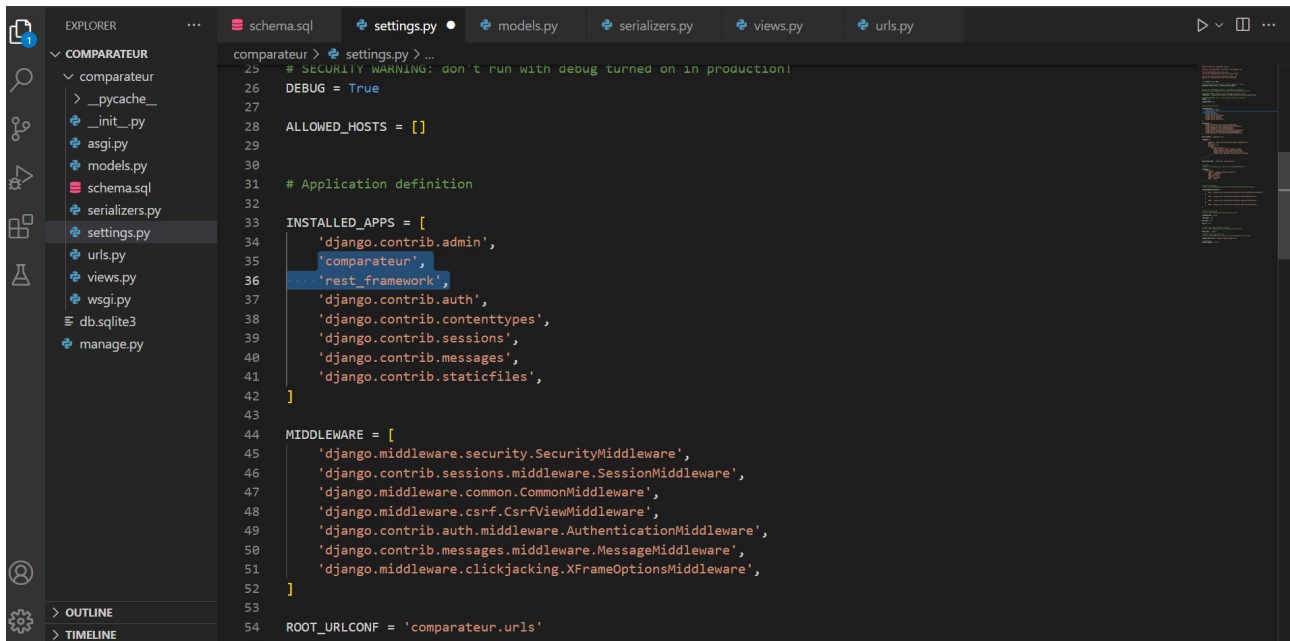
```
6 # * Remove 'managed = False' lines if you wish to allow Django to create, modify, and delete the table
7 # Feel free to rename the models, but don't rename db_table values or field names.
8 from django.db import models
9
10
11 class Categories(models.Model):
12     libelle = models.CharField(max_length=255)
13
14     class Meta:
15         managed = False
16         db_table = 'categories'
17
18     @classmethod
19     def getCategories(cls):
20         return cls.objects.all()
21
22
23 class Magasins(models.Model):
24     nom = models.CharField(max_length=255)
25     adresserue = models.CharField(max_length=255)
26     adresseville = models.CharField(max_length=255)
27     adressecp = models.IntegerField()
28     latitude = models.DecimalField(max_digits=10, decimal_places=8, blank=True, null=True)
29     longitude = models.DecimalField(max_digits=11, decimal_places=8, blank=True, null=True)
30     datecreation = models.DateField(blank=True, null=True)
31     datemodification = models.DateField(blank=True, null=True)
32
33     class Meta:
34         managed = False
35         db_table = 'magasins'
```

Fichier models.py

Pour permettre la création d'API avec django, il faut installer le module, via la commande :

pip install djangoestframework


Il faudra également mettre à jour le fichier settings comme suit :



```
25 # SECURITY WARNING: don't run with debug turned on in production!
26 DEBUG = True
27
28 ALLOWED_HOSTS = []
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     'comparateur',
40     'rest_framework',
41 ]
42
43 MIDDLEWARE = [
44     'django.middleware.security.SecurityMiddleware',
45     'django.contrib.sessions.middleware.SessionMiddleware',
46     'django.middleware.common.CommonMiddleware',
47     'django.middleware.csrf.CsrfViewMiddleware',
48     'django.contrib.auth.middleware.AuthenticationMiddleware',
49     'django.contrib.messages.middleware.MessageMiddleware',
50     'django.middleware.clickjacking.XFrameOptionsMiddleware',
51 ]
52
53 ROOT_URLCONF = 'comparateur.urls'
```

Mise à jour des propriétés `INSTALLED_APPS`

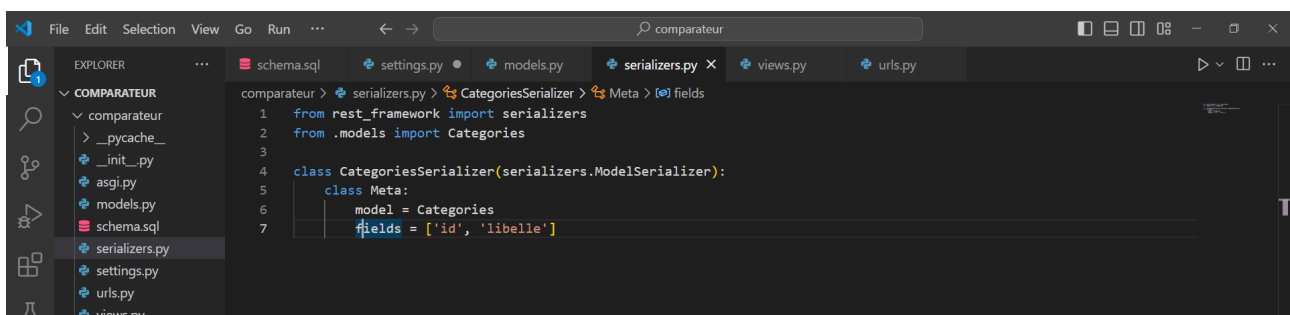
Pour tester le bon fonctionnement de l'API, nous avons alimenté la base de données avec une catégorie. On a ensuite créé une méthode dans le fichier `models.py` pour récupérer toutes les catégories.



```
10 class Categories(models.Model):
11     libelle = models.CharField(max_length=255)
12
13     class Meta:
14         managed = False
15         db_table = 'categories'
16
17     @classmethod
18     def getCategories(cls):
19         return cls.objects.all()
```

Méthode pour récupérer toutes les catégories

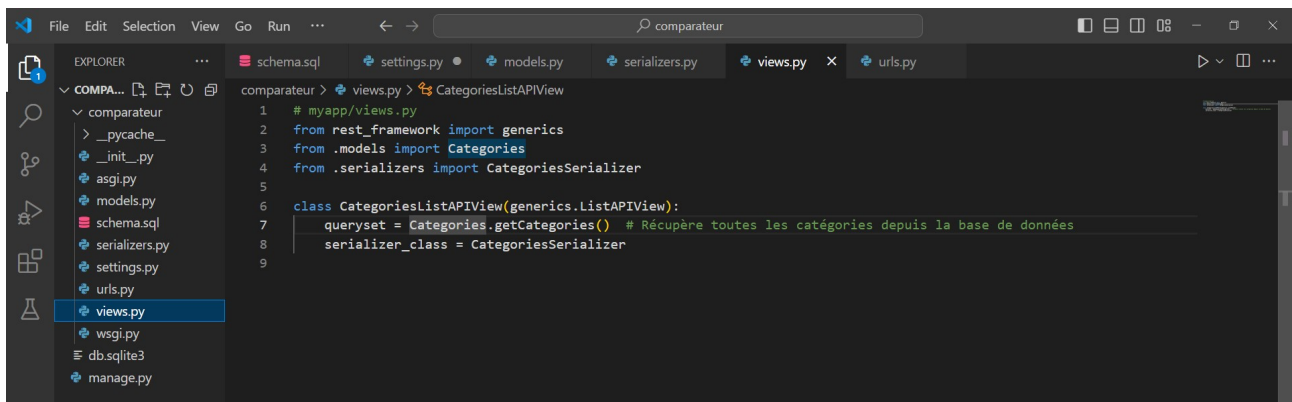
Une fois la méthode créée, on met en place le Serializer. La principale fonction d'un serializer est de convertir des objets Django (généralement des modèles) en formats de données comme JSON qui peuvent être facilement transmis via HTTP.



```
1 from rest_framework import serializers
2 from .models import Categories
3
4 class CategoriesSerializer(serializers.ModelSerializer):
5     class Meta:
6         model = Categories
7         fields = ['id', 'libelle']
```

Fichier `serializer.py`

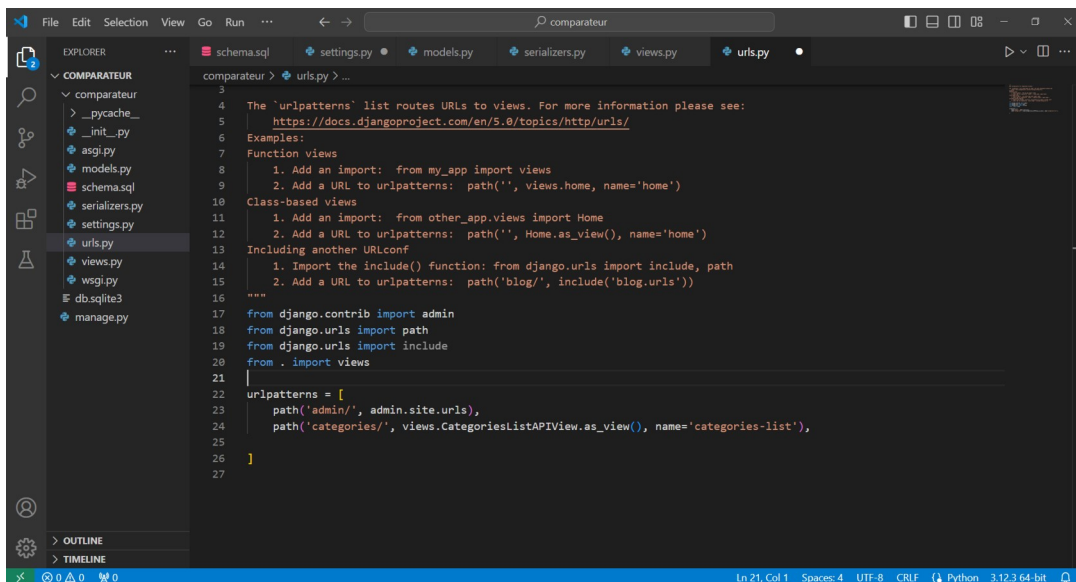
Ensuite, il faudra créer le fichier pour la vue.

A screenshot of the Visual Studio Code editor. The Explorer sidebar on the left shows a project named 'comparateur' with files like __init__.py, asgi.py, models.py, schema.sql, serializers.py, settings.py, urls.py, and views.py. The views.py file is selected and open in the editor. The code in views.py defines a Django REST framework class view:

```
1 # myapp/views.py
2 from rest_framework import generics
3 from .models import Categories
4 from .serializers import CategoriesSerializer
5
6 class CategoriesListAPIView(generics.ListAPIView):
7     queryset = Categories.objects.all() # Récupère toutes les catégories depuis la base de données
8     serializer_class = CategoriesSerializer
9
```

Fichier view.py

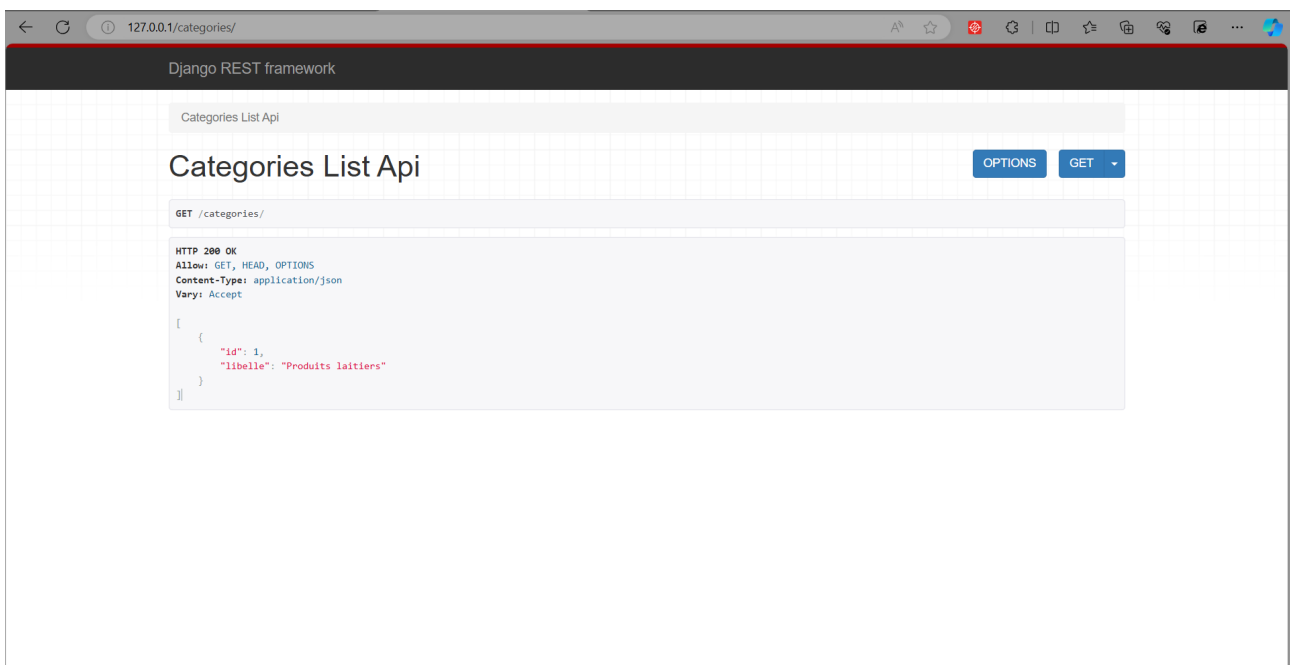
Enfin, il faudra configurer les urls de requêtes.

A screenshot of the Visual Studio Code editor showing the urls.py file. The Explorer sidebar shows the same project structure as before. The urls.py file contains the following code:

```
3
4 The 'urlpatterns' list routes URLs to views. For more information please see:
5   https://docs.djangoproject.com/en/5.0/topics/http/urls/
6 Examples:
7 Function views
8     1. Add an import: from my_app import views
9     2. Add a URL to urlpatterns: path('', views.home, name='home')
10 Class-based views
11     1. Add an import: from other_app.views import Home
12     2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
13 Including another URLconf
14     1. Import the include() function: from django.urls import include, path
15     2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
16 """
17 from django.contrib import admin
18 from django.urls import path
19 from django.urls import include
20 from . import views
21
22 urlpatterns = [
23     path('admin/', admin.site.urls),
24     path('categories/', views.CategoriesListAPIView.as_view(), name='categories-list'),
25 ]
26
27
```

Fichier urls.py

Maintenant lorsqu'on interroge l'url on obtient :



Pour les besoins de notre API, nous auront besoin de modules supplémentaires.

pip install djangorestframework-simplejwt (Module permettant l'authentification basée sur les jetons)

pip install psycopg2-binary (Module permettant la connexion à une base de données PostgreSQL)

pip install django-cors-headers (Module qui permet de gérer les entêtes CORS)

pip install drf-yasg (Module pour la génération automatique de la documentation de l'API)

pip install setuptools (Si erreur sur le module setuptools, permet de gérer les dépendances)