REPUBLIQUE TUNISIENNE
MINISTERE DE L'ENSEIGNEMENT
SUPERIEUR, DE LA RECHERCHE SCIENTIFIQUE
ET DES TECHNOLOGIES DE L'INFORMATION ET DE LA
COMMUNICATION

DIRECTION GENERALE DES ETUDES TECHNOLOGIQUES
INSTITUT SUPÉRIEUR DES ETUDES TECHNOLOGIQUES DE SFAX
DÉPARTEMENT TECHNOLOGIE DE L'INFORMATIQUE

Travaux Pratiques n°7

API Rest Framework Django

Une API Rest peut être créée dès le début d'un projet, mais aussi être mise en place sur un projet existant pour le faire évoluer.

Nous allons **adapter notre projet Django « mysite » pour lui intégrer une API avec Django REST Framework** (DRF).

I. Installation du DRF

Django REST Framework est une librairie permettant la mise en place d'une API pour Django (vous pouvez regarder sa documentation sur le site officiel, en anglais). Basée sur le framework, elle propose la mise en place des endpoints (une **URL sur laquelle on réalise différents appels**) d'une façon similaire à la mise en place des URL, Views et Forms de Django.

Lancez l'installation du Rest framework

```
pip install djangorestframework
```

II. Configurations

Ajoutez `'rest_framework'` à votre `INSTALLED_APPS` décor.

```
INSTALLED_APPS = [
    ...
    'rest_framework',
]
```

Si vous avez l'intention d'utiliser l'API navigable, vous souhaiterez probablement également ajouter les vues de connexion et de déconnexion du framework REST. Ajoutez ce qui suit à votre `urls.py` fichier racine.

```
urlpatterns = [
    ...
    path('api-auth/', include('rest_framework.urls'))
]
```

III. Création d'un premier endpoint

En avant, mettons en place ce premier endpoint en permettant à nos visiteurs d'accéder à la liste des catégories de nos produits.

La toute première chose à faire lors de la réalisation d'un endpoint est de se demander **quelles sont les informations importantes que nous souhaitons en tirer**.

Dans notre cas, pour afficher la liste des catégories, nous allons avoir besoin de leur **nom**, mais également de leur **identifiant**. L'identifiant sera utile aux clients (application front, mobile…) pour identifier de manière claire et unique les catégories s'ils ont des actions à faire dessus, comme afficher les produits qui constituent une catégorie.

DRF met à notre disposition des **serializers** qui permettent de transformer nos models Django en un autre format qui, lui, est exploitable par une API.

Lorsque notre API sera consultée, le serializer va nous permettre de transformer notre objet en un JSON et, inversement, lorsque notre API va recevoir du JSON, elle sera capable de le transformer en un objet.

1- Créons un fichier `serializers.py` sous le répertoire de l'application magasin et **écrivons notre premier serializer**, que nous nommerons `CategorySerializer` pour rester clair et précis. 😉

Il est nécessaire de définir sa classe `Meta` et la liste des champs, exactement comme pour un formulaire. De plus, les noms des attributs sont identiques.

```
from rest_framework.serializers import ModelSerializer

from shop.models import Categorie

class CategorySerializer(ModelSerializer):

    class Meta:
        model = Categorie
        fields = ['id', 'name']
```

2- Modifier le fichier views.py

DRF nous propose une **`APIView`** qui a également un fonctionnement similaire aux **`Views`** de Django.

Nous devons réécrire la méthode `get` qui réalisera les actions suivantes :

- Récupérer toutes les catégories en utilisant l'ORM de Django ;
- Sérialiser les données à l'aide de notre serializer ;
- Renvoyer une réponse qui contient les données sérialisées.

```
from rest_framework.views import APIView
from rest_framework.response import Response

from shop.models import Categorie
```

```
from magasin.serializers import CategorySerializer

class CategoryAPIView(APIView):

   def get(self, *args, **kwargs):
      categories = Categorie.objects.all()
      serializer = CategorySerializer(categories, many=True)
      return Response(serializer.data)
```

✍ Le paramètre `many` permet de préciser au Serializer qu'il va devoir générer une liste d'éléments à partir de l'itérable (notre queryset) qui lui est transmis.

✍ Dans le cas où il y a seul élément à sérialiser, il n'est pas nécessaire de préciser le paramètre `many`, et l'objet peut directement être donné au Serializer sans le mettre dans un itérable.

Enfin, pour obtenir les données sérialisées, nous appelons la propriété `data` de notre serializer. Ce sont ces données qui sont utilisées pour construire la réponse.

3- Modifier urls.py

Il ne reste plus qu'à créer l'URL correspondante et l'associer à notre View dans le fichier urls.py de l'application magasin:

```
from .views import CategoryAPIView

urlpatterns = [
   .....
   path('api/category/', CategoryAPIView.as_view())
]
```

Notre endpoint est maintenant accessible sur l'URL http://127.0.0.1:8000/magasin/api/category/ .
L'interface proposée par DRF nous permet de voir également le status code obtenu. Dans notre cas, 200, qui indique un succès.

IV. Créer un deuxième endPoint

Je vous propose de prendre la main et de **mettre en place un nouvel endpoint** qui va permettre de lister tous les produits de notre boutique en ligne (magasin).

Utilisons l'URL suivante : http://127.0.0.1:8000/magasin/api/produits/.
Cet endpoint doit retourner les informations suivantes des produits :

- Son identifiant id.
- Son nom libellé.
- Sa description
- L'identifiant de sa catégorie

1- Ajouter dans le fichier **serializers.py** la classe **ProduitSerializer**

2- Ajouter dans le fichier **views.py** la classe **ProduitAPIView**
3- Ajouter l'url correspondant dans le fichier **urls.py**

V. Créer un troisième endPoint

1- On sélectionne un produit dont l'identifiant est donné en paramètre. La classe définie dans views.py sera :

```
from rest_framework import viewsets
class ProductViewset(viewsets.ReadOnlyModelViewSet):


   serializer_class = ProduitSerializer

   def get_queryset(self):
      queryset = Produit.objects.filter(active=True)
      category_id = self.request.GET.get('category_id')
      if category_id:
         queryset = queryset.filter(categorie_id=category_id)
      return queryset
```

2- Puis ajouter l'url correspondant dans le fichier urls.py du projet

```
from rest_framework import routers

 from magasin.views import ProductViewset, CategoryAPIView


# Ici nous créons notre routeur

router = routers.SimpleRouter()

# Puis lui déclarons une url basée sur le mot clé 'category' et notre view

# afin que l'url générée soit celle que nous souhaitons '/api/category/'

router.register('produit', ProductViewset, basename='produit')


urlpatterns=[
      ……,
        path('api/', include(router.urls)),
]…
```

3- Accès à l'endPoint dans le navigateur web

En utilisant l'URL suivante : http://127.0.0.1:8000/api/produit/1 : ce qui donne le produit dont id=1.

En utilisant l'URL suivante : http://127.0.0.1:8000/api/produit/?category_id=2 ce qui donne les produits appartenant à la catégorie dont id=2.