

Créer des API

Avec

django

Rapport du Projet :

Gestion des Films au cinéma

Django REST Framework

Réalisé par : EZZAHRAOUI HAROUN

Encadré par : Mr. HABIB AYAD

Table des Matières

I INTRODUCTION

II PRESENTATION DU PROJET

- ⇒ Problématique et contrainte de réalisation

III ETAT DE L'ART DJANGO

- ⇒ Méthode REST
- ⇒ Django
- ⇒ Le modèle MVT

IV REALISATION

- ⇒ Langage de programmation, distributions et environnement
- ⇒ Installation des exigences
- ⇒ Créer notre projet
- ⇒ Créer notre application
- ⇒ Créer notre base de données et assurer la connexion
- ⇒ Création de nos modelés et migration
- ⇒ Administrer notre application via panneau d'administration
- ⇒ Sérialisation
- ⇒ Contrôler notre application
- ⇒ Router l'application

V TEST

- ⇒ Tester notre api avec Postman en appliquant le JWTAuthentification
- ⇒ Test sur FILMS
- ⇒ Test sur USER
- ⇒ Test sur CINEMA
- ⇒ Test sur GENRE
- ⇒ Test sur TITLE « Utilisant API externe pour recuperer les titre des films »
- ⇒ Join FILM et CINEM
- ⇒ Join FILM et GENRE

V CONCLUSION

INTRODUCTION

Framework Web (WF) ou une infrastructure d'application Web (WAF) est une infrastructure logicielle conçue pour prendre en charge le développement d'applications Web, notamment des services Web, des ressources Web et des API Web. Les Framework Web fournissent un moyen standard de créer et de déployer des applications Web sur le World Wide Web. Les Framework Web visent à automatiser les frais généraux associés aux activités courantes effectuées dans le développement Web. Par exemple, de nombreux Framework Web fournissent des bibliothèques pour l'accès aux bases de données, les Framework de création de modèles et la gestion de session, et ils encouragent souvent la réutilisation du code. Bien qu'ils visent souvent le développement de sites Web dynamiques, ils s'appliquent également aux sites Web statiques.

Il existe une myriade d'options disponibles lors du choix du Framework backend avec lequel vous souhaitez travailler. Bien que chaque Framework backend ait son propre ensemble d'avantages et d'inconvénients, il y a également quelques autres facteurs que vous voudrez prendre en compte avant de prendre une décision finale. Dans ce guide, nous le choix des Framework éprouvés et les plus récents pour vous aider à décider quel est le meilleur Framework backend pour vous.

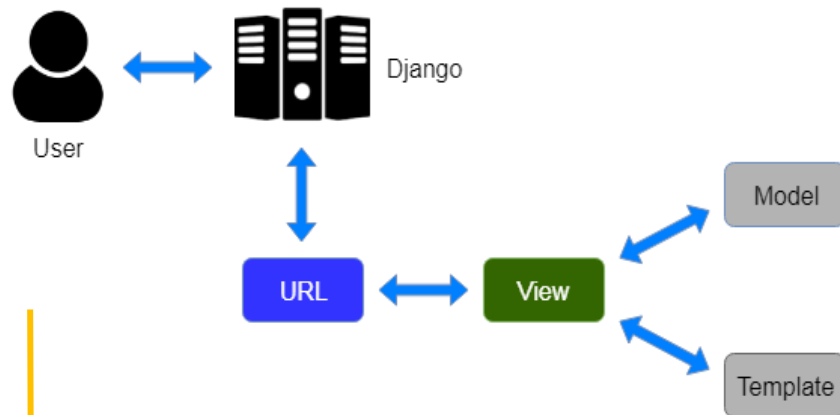
ETAT DE L'ART DJANGO

Méthode REST : Pour rendre accessible des données via un site, il existe les méthodes dites REST (représentationnel state Transfer). Il s'agit d'un ensemble de conventions et de bonnes pratiques à respecter et non d'une technologie entière. L'information de base, dans une architecture REST, est appelée ressource. Toute information qui peut être nommée est une ressource : la description d'un bâtiment, la liste des arrêts de bus ou n'importe quel concept. Dans un système hypermédia, une ressource est tout ce qui peut être référencé par un lien. L'interface entre les composants est simple et uniforme. En HTTP, cette interface est implantée par les verbes GET, PUT, POST, DELETE, . . . qui permettent aux composants de manipuler les ressources de manière simple. Par exemple quand un agent voudra récupérer la liste des arrêts de bus depuis l'application, il passera par la méthode GET qui lui retournera les ressources voulues.

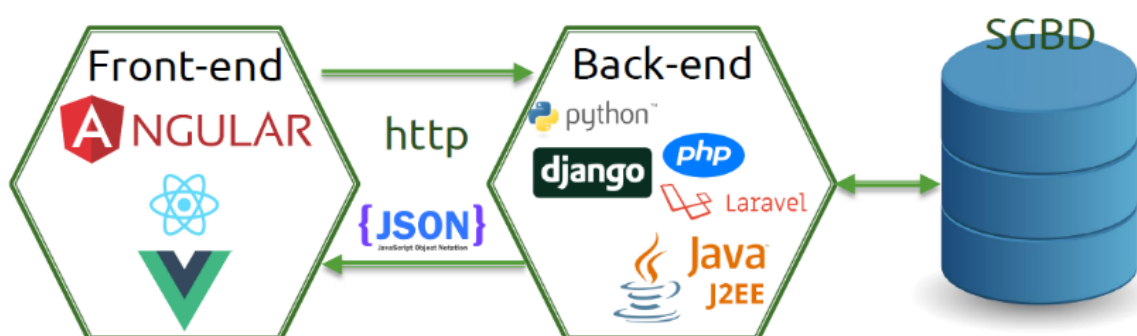
Django est un Framework web Python de haut niveau qui encourage le développement rapide et propre. Gratuit et open source, Django vous permet d'éviter de réinventer la roue grâce à toutes les librairies disponibles en Python, mais aussi de tout ce que ce Framework offre d'es son installation. Le fonctionnement d'une application Django se divise en 2 parties. La première section a pour but de préparer l'étape entre l'utilisateur et l'application en elle-même. Cette section gère la relation entre les bases de données et les applications Django, mais aussi s'occupe du routage via des règles URL. La seconde section est l'application. Organisée en modèles, vues et Template, l'application se trouve au cœur du projet Django.

Utilisé par la NASA ou le Washington Post, Django prouve sa fiabilité et sa stabilité à travers ce type d'organisme.

Le modèle MVT : Django se base sur le modèle MVT, légèrement différent du modèle MVC, le Framework gère lui-même le contrôleur et laisse place au Template.



Lors d'une requête venant de l'utilisateur, le Framework Django gère lui-même, via les règles de routage défini par le développeur, de charger la bonne vue correspondante au résultat voulu. Une Template est un fichier HTML qui sera récupéré par la vue pour être envoyé à l'utilisateur, mais entre cette étape, Django va exécuter la Template comme si c'était un fichier de code. Inclus dans les Template, le Framework propose l'utilisation des structures conditionnelles, des boucles, des variables... afin d'avoir une grande liberté de développement.



PRESENTATION DU PROJET

On veut créer une application qui nous aide à gérer les cinémas films et genre de tel façon que les films se récupère automatiquement d'une api externe pour qu'il nous aide pour remplir notre base de données, la table film dispose des colonnes suivant « **ID et TITLE et YEAR_PRODUCTION et DATE_PRESENATION et OWNER** », la table cinéma dispose des colonnes suivant « **ID et NAME et CITY et FILMS et OWNER** », la table genre dispose des colonnes suivant « **ID et NAME** ».

Pour assurer le bon fonctionnement la relation entre :

- **FILM et CINEMA est ManyToMany**
- **FILM et Genre est OneToMany**

Contrainte :

- ⇒ **L'utilisation des Apis Génériques « REST FRAMEWORK »**
- ⇒ **Appliquant l'authentification avec JWT.**
- ⇒ **Montrer la jointure entre les tables.**

Sans utilisation de l'interface Montrons le fonctionnement de l'api.

REALISATION

1- Langage de programmation, distributions et environnement



Python est le langage de programmation open source le plus employé par les informaticiens, à cause de sa simplicité et sa facilité d'utilisation, il est simple à coder, vous n'êtes pas obligé d'ajouter des points-virgules ou des accolades n'importe où. En python, vous pouvez écrire de petits codes pour effectuer de grandes tâches. Par conséquent, vous gagnez du temps même lors de l'écriture du code



Anaconda est une distribution libre et open source des langages de programmation Python et R appliqué au développement d'applications dédiées à la science des données et à l'apprentissage automatique (traitement de données à grande échelle, analyse prédictive, calcul scientifique), qui vise à simplifier la gestion des paquets et de déploiement.



Visual Studio Code est un éditeur de code extensible développé par Microsoft pour Windows, Linux et MacOS



Postman est une plateforme de collaboration pour le développement d'API. Les fonctionnalités de Postman simplifient chaque étape de la création d'une API et rationalisent la collaboration afin que vous puissiez créer de meilleures API, plus rapidement.



MySQL est un système de gestion de bases de données relationnelles (SGBDR). Il fait partie des logiciels de gestion de base de données les plus utilisés au monde³, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec *Oracle*, *PostgreSQL* et *Microsoft SQL Server*.

2- Installation des exigences

Django :

Dans le cadre du processus de développement de Django 3.2, Django 3.2a1 est disponible. Cette version est réservée aux utilisateurs qui souhaitent essayer la nouvelle version et aider à identifier les bogues restants avant la version 3.2.

Pour L'installation du Django tapez la commande

```
> pip install --pre django
```

Djangorestframework :

Le Framework Django REST est une boîte à outils puissante et flexible pour la création d'API Web.

Pour L'installation du djangorestframework tapez la commande

```
> pip install djangorestframework
```

*Ajoutez dans `FilmManagement\FilmManagement\settings.py`
dans `INSTALLED_APP` ajouter 'rest_framework',

MySQLclient :

Nous utiliserons MySQL comme base de données. Vous pouvez également souhaiter utiliser une autre base de données ou avoir déjà une base de données installée. Cette étape à pour d'assurer la connexion à une base de données MySQL et réaliser des migrations.

Pour L'installation du mysqlclient tapez la commande

```
> pip installer mysqlclient
```

Djangorestframework_simplejwt :

Pour bénéficier la stratégie d'authentification utilisée par les applications client/serveur, json web token nous allons utiliser la djangorestframework_simplejwt bibliothèque, recommandée par les développeurs DRF.

Pour L'installation du djangorestframework_simplejwt tapez la commande

```
> pip install djangorestframework_simplejwt
```

3- Créer notre projet

- Pour la création d'un projet dans django on utilise la commande suivante

```
D:\>
λ cd "D:\AI&GI\S3\5- Django-backend\Project"

D:\AI&GI\S3\5- Django-backend\Project
λ conda activate django1

D:\AI&GI\S3\5- Django-backend\Project
(django1) λ django-admin startproject FilmManagement

D:\AI&GI\S3\5- Django-backend\Project
(django1) λ
```

Dans notre invite de commandes on accède à notre dossier puis activé l'environnement qui contient les packages requis en suite on exécute
> **django-admin startproject FilmManagement**

4- Créer notre application

- La création d'une application au sein de notre projet FilmManagement

```
D:\AI&GI\S3\5- Django-backend\Project
(django1) λ cd FilmManagement\

D:\AI&GI\S3\5- Django-backend\Project\FilmManagement
(django1) λ python manage.py startapp filmscinema
```

Dans notre invite de commandes on accède à notre projet puis on lance la commande
> **python manage.py startapp filmscinema**

- Ajoutez dans **FilmManagement\FilmManagement\settings.py** la configuration suivante

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'filmscinema',
]
```

A la fin de INSTALLED_APP ajoutons
Le nom de notre application

5- Créer notre base de données et assurer la connexion avec elle

- La création d'une base de données en MySQL se fait de deux manières via ligne de commande ou via l'interface du navigateur
- Pour accéder à SGBDR MySQL par défaut on a :
 - ⇒ Username : root
 - ⇒ Password :



- Dans fichier `FilmManagement\FilmManagement\settings.py` ajoutant la configuration suivante

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'filmsdb',
        'USER': 'root',
        'PASSWORD': '',
        'HOST': 'localhost',
        'PORT': '3306',
        'OPTIONS': {
            'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",
        },
    },
}
```

6- Création de nos modèles et migration

- A l'intérieur de notre fichier **models.py** on va créer :

```
class Genre(models.Model):
    name = models.CharField(max_length=100)

    class Meta:
        ordering = ('name',)

    def __str__(self):
        return self.name
```

Création de notre modèle Genre dans ses colonnes on trouve le nom et on va lister les genres selon le nom

```
class Film(models.Model):
    title = models.CharField(max_length=100)
    year_prod = models.IntegerField()
    genre = models.ForeignKey(Genre, on_delete=models.CASCADE)
    presentation_date = models.DateTimeField()
    owner = models.ForeignKey(
        'auth.User',
        related_name='films',
        on_delete=models.CASCADE,
        null=True
    )

    class Meta:
        ordering = ('title',)

    def __str__(self):
        return '%s : %d' % (self.title, self.year_prod)
```

Création de notre modèle Film dans ses colonnes on trouve le titre et l'année de production et son genre comme clé étrangère, la date de sortie dans le cinéma et on va lister les Films selon le titre.

```

class Cinema(models.Model):
    name = models.CharField(max_length=200)
    city = models.CharField(max_length=100)
    films = models.ManyToManyField(Film)
    #films= models.ForeignKey(Film.title)
    owner = models.ForeignKey(
        'auth.User',
        related_name='cinemas',
        on_delete=models.CASCADE,
        null=True
    )

    class Meta:
        ordering = ('city',)

    def __str__(self):
        return '%s : %s' % (self.name,self.city)

```

Création de notre modèle Cinéma dans ses colonnes on trouve le nom et la ville, les films dans ce cinéma comme clé étrangère mais on le représente avec la relation many to many aussi les couronnées de super user et on va lister les cinémas selon leur ville.

- Pour que nos modèles soient physiques on doit migrer vers la base de données

```

(django1) PS D:\AI&GI\S3\5- Django-backend\Project\FilmManagement> python .\manage.py makemigrations
Migrations for 'filmscinema':
  filmscinema\migrations\0001_initial.py
    - Create model Genre
    - Create model Film
    - Create model Cinema
(django1) PS D:\AI&GI\S3\5- Django-backend\Project\FilmManagement> python .\manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, filmscinema, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying filmscinema.0001_initial... OK
  Applying sessions.0001_initial... OK
(django1) PS D:\AI&GI\S3\5- Django-backend\Project\FilmManagement>

```

Table	Action
<input type="checkbox"/> auth_group	★ Parcourir
<input type="checkbox"/> auth_group_permissions	★ Parcourir
<input type="checkbox"/> auth_permission	★ Parcourir
<input type="checkbox"/> auth_user	★ Parcourir
<input type="checkbox"/> auth_user_groups	★ Parcourir
<input type="checkbox"/> auth_user_user_permissions	★ Parcourir
<input type="checkbox"/> django_admin_log	★ Parcourir
<input type="checkbox"/> django_content_type	★ Parcourir
<input type="checkbox"/> django_migrations	★ Parcourir
<input type="checkbox"/> django_session	★ Parcourir
<input type="checkbox"/> filmscinema_cinema	★ Parcourir
<input type="checkbox"/> filmscinema_cinema_films	★ Parcourir
<input type="checkbox"/> filmscinema_film	★ Parcourir
<input type="checkbox"/> filmscinema_genre	★ Parcourir
14 tables	Somme

La migration crée directement les table d'authentification, les groups et user

La table filmscinema_cinema_films est créé automatiquement à cause de notre relation « ManyToMany » dans le modelé

7- Administrer notre application via panneau d'administration

- Maintenant on va créer un Super User pour que nous puisse accéder au panneau

```

Applying 0002_initial... OK
(django1) PS D:\AI&GI\S3\5- Django-backend\Project\FilmManagement> python .\manage.py createsuperuser
Username (leave blank to use 'harou'): admin
Email address: admin@dm.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
(django1) PS D:\AI&GI\S3\5- Django-backend\Project\FilmManagement>

```

- Dans le fichier **admin.py** on ajoute

```
from django.contrib import admin
from .models import Film, Cinema, Genre
# Register your models here.
admin.site.register(Film)
admin.site.register(Cinema)
admin.site.register(Genre)
```

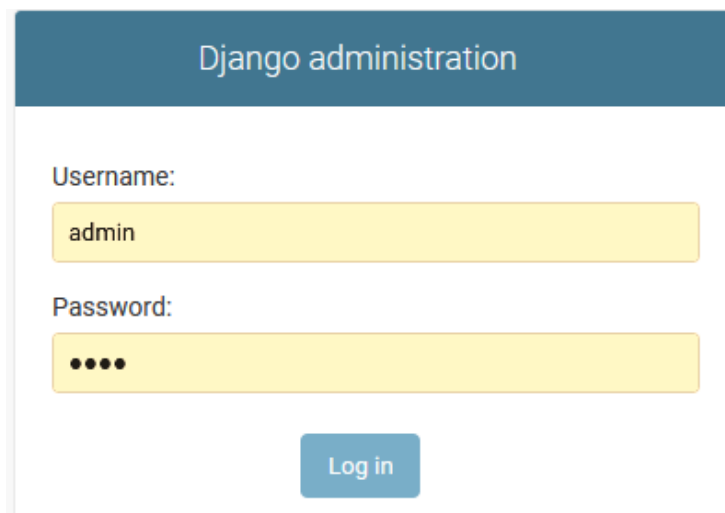
Pour que les tables se visualise dans l' AdminPanel on doit registrer les modelés qu' on a créé

- Pour vérifier on doit lancer le serveur avec la commande suivante

```
Superuser created successfully.
(django1) PS D:\AI&GI\S3\5- Django-backend\Project\FilmManagement> python .\manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 19, 2021 - 01:19:35
Django version 2.2, using settings 'FilmManagement.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

- Si on accède au lien « <http://127.0.0.1:8000/admin/> »



- On peut directement administrer notre application via adminPanel ou Django administration

Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups

+ Add

Change

Users

+ Add

Change

FILMSCINEMA

Cinemas

+ Add

Change

Films

+ Add

Change

Genres

+ Add

Change

Add genre

Ajouter un genre via DjangoAdmin

Name:

Comedy

SAVE

Save and add another

Save and continue editing

Add film

Ajouter un Film via DjangoAdmin

Title:

Friends

Year prod:

1994

Genre:

Comedy

+ Add

Presentation date:

Date: 2021-02-16

Today

Time: 00:29:48

Now

Note: You are 1 hour ahead of server time.

Owner:

admin

+ Add

Save and add another

Save and continue editing

SAVE

Add cinema

Ajouter un Cinéma via DjangoAdmin

Name:

ATLAS

City:

CASABLANCA

Films:

Friends : 1994

+ Add

Hold down "Control", or "Command" on a Mac, to select more than one.

Owner:

admin

+ Add

Save and add another

Save and continue editing

SAVE



8- Sérialisation

- La sérialisation permet aux données, complexes telles que les modèles, d'être convertis en type de données natives pour Python. Ils peuvent ensuite être facilement rendus dans les formats JSON ou XML. Dans le cas présent, la sérialisation permet aussi de sélectionner les champs à afficher pour limiter la vue des données.

On va utiliser le modèle prè défini dans Django d' user, puis on définit la relation entre utilisateur et les deux modelé films et cinémas

```
from rest_framework import serializers
from .models import Film, Cinema, Genre
from django.contrib.auth.models import User

class UserSerializer(serializers.ModelSerializer):
    films = serializers.PrimaryKeyRelatedField(many=True, queryset=Film.objects.all())
    cinemas = serializers.PrimaryKeyRelatedField(many=True, queryset=Cinema.objects.all())

    class Meta:
        model = User
        fields = ('id', 'username', 'films', 'cinemas')
```

- On va créer deux sérialiser l' un pour la modification et l' autre pour la visualisation seulement

Sérialisation des genres
En récupérant ID et NAME

```
class GenreSerializer(serializers.ModelSerializer):
    class Meta:
        model = Genre
        fields = ('id', 'name')
        depth = 1

class GenreWriteSerializer(serializers.ModelSerializer):
    class Meta:
        model = Genre
        fields = ('id', 'name')
```

Sérialisation des genres
En récupérant ID et TITLE et
YEAR_PRODUCTION et
DATE_PRESENATION et OWNER

```
class FilmSerializer(serializers.ModelSerializer):
    owner = serializers.ReadOnlyField(source='owner.username')

    class Meta:
        model = Film
        fields = ('id', 'title', 'year_prod', 'genre', 'presentation_date', 'owner')
        depth = 1

class FilmWriteSerializer(serializers.ModelSerializer):
    genre = serializers.PrimaryKeyRelatedField(queryset=Genre.objects.all(), allow_null=True)

    class Meta:
        model = Film
        fields = ('id', 'title', 'year_prod', 'genre', 'presentation_date')
```

Sérialisation des genres
On récupérant ID et NAME et CITY et FILMS et OWNER

```
class CinemaSerializer(serializers.ModelSerializer):

    owner = serializers.ReadOnlyField(source='owner.username')

    class Meta:
        model = Cinema
        fields = ('id', 'name', 'city', 'films', 'owner')
        depth = 1

class CinemaWriteSerializer(serializers.ModelSerializer):

    class Meta:
        model = Cinema
        fields = ('id', 'name', 'city', 'films', 'owner')
        #fields = ('id', 'name', 'city')
```


9- Contrôler notre application

- Pour contrôler notre application on doit créer des views dans le fichier **views.py**

Ensemble des package qu' on va utiliser

```
from django.shortcuts import render
from .models import *
from .serializers import *
from rest_framework.response import Response
from rest_framework import generics, permissions
from .permissions import IsOwnerOrReadOnly
from rest_framework.decorators import api_view
import pdb
import requests
from rest_framework import generics
from rest_framework import mixins
from rest_framework.permissions import IsAuthenticated
from datetime import datetime
```

Pour récupérer les films d' une autre api externe qui s' appelle omdbapi on parcourt la requête puis on stocke dans un dictionnaire pour assure le bon fonctionnement de api

```
@api_view(['GET'])
def film_title(request, title= 'Love'):

    # Get a List of films that have the word 'hunger' in the title
    if request.method == 'GET':
        films = requests.get('http://www.omdbapi.com/?i=tt3896198&apikey=19f3d35d&type=movie&s={}'.format(title))
        json = films.json()
        a = []
        for key in json['Search']:

            films_dict = {}
            films_dict['title'] = key['Title']
            films_dict['year_prod'] = key['Year']
            films_dict['presentation_date'] = datetime.now()

            a.append(films_dict)
        serializedFilm = FilmSerializer(a, many=True)
        return Response(serializedFilm.data)
```

Récupération et Modification des utilisateur par List ou par id

```
class UserList(generics.ListCreateAPIView):
    permission_classes = (IsAuthenticated,)
    queryset = User.objects.all()
    serializer_class = UserSerializer

class UserDetails(generics.RetrieveUpdateDestroyAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
```

L' utilisation des api Generics

Récupération et Modification des Films par List ou par id

Parcourant la requête et récupère les données dans un dictionnaire

```
class FilmList(generics.ListCreateAPIView):
    permission_classes = (permissions.IsAuthenticatedOrReadOnly, IsOwnerOrReadOnly)
    queryset = Film.objects.all()

    def get(self, request, *args, **kwargs):
        k = request.GET.keys()
        filter_dict = {}
        if(k):
            for key, value in request.GET.items():
                filter_dict[key] = value
            films = Film.objects.filter(**filter_dict)
            serialized_films = FilmSerializer(films, many=True)
            return Response(serialized_films.data)
        else:
            return Response(FilmSerializer(Film.objects.all(), many=True).data)

    def perform_create(self, serializer):
        serializer.save(owner=self.request.user)

    def get_serializer_class(self):
        if(self.request.method == 'GET'):
            return FilmSerializer
        return FilmWriteSerializer
```

L' utilisation des api Generics

```
class FilmDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Film.objects.all()
    permission_classes = (permissions.IsAuthenticatedOrReadOnly, IsOwnerOrReadOnly)

    def get_serializer_class(self):
        if(self.request.method == 'GET'):
            return FilmSerializer
        return FilmWriteSerializer
```

Récupération et Modification des Cinémas par List ou par id

Parcourant la requête et récupère les données dans un dictionnaire

```
class CinemaList(generics.ListCreateAPIView):
    queryset = Cinema.objects.all()
    serializer_class = CinemaSerializer()
    permission_classes = (permissions.IsAuthenticatedOrReadOnly,
                          IsOwnerOrReadOnly)
    def get(self, request, *args, **kwargs):
        k = request.GET.keys()
        filter_dict = {}
        if(k):
            for key, value in request.GET.items():
                filter_dict[key] = value
            cinemas = Cinema.objects.filter(**filter_dict)
            serialized_cinemas = CinemaSerializer(cinemas, many=True)
            return Response(serialized_cinemas.data)
        else:
            return Response(CinemaSerializer(Cinema.objects.all(), many=True).data)

    def get_serializer_class(self):
        if(self.request.method == 'GET'):
            return CinemaSerializer
        return CinemaWriteSerializer

    def perform_create(self, serializer):
        serializer.save(owner=self.request.user)
```

```

class CinemaDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Cinema.objects.all()
    serializer_class = CinemaSerializer
    permission_classes = (permissions.IsAuthenticatedOrReadOnly,
                          IsOwnerOrReadOnly)

    def get_serializer_class(self):
        if(self.request.method == 'GET'):
            return CinemaSerializer
        return CinemaWriteSerializer

```

Récupérer les détails sur un cinema précis

Récupération et Modification des Genres par List ou par id

Parcourant la requête et récupère les données dans un dictionnaire

```

class GenreList(generics.ListCreateAPIView):
    queryset = Genre.objects.all()
    serializer_class = GenreSerializer

    def get_serializer_class(self):
        if(self.request.method == 'GET'):
            return GenreSerializer
        return GenreWriteSerializer

class GenreDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Genre.objects.all()
    serializer_class = GenreSerializer

    def get_serializer_class(self):
        if(self.request.method == 'GET'):
            return GenreSerializer
        return GenreWriteSerializer

```

L' utilisation des api Generics

10- Router l'application

- Dans `filmscinema\urls.py` on ajoute des path et des url pour faire interagir notre api

```
from django.urls import path
from django.contrib import admin
from rest_framework.urlpatterns import format_suffix_patterns
from . import views
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView

urlpatterns = [
    path('admin/', admin.site.urls),

    path('users/', views.UserList.as_view()),
    path('users/<int:id>', views.UserDetail.as_view()),

    path('genre/', views.GenreList.as_view(), name='genre'),
    path('genre/<int:id>', views.GenreDetail.as_view(), name='genre'),

    path('films/', views.FilmList.as_view(), name='films'),
    path('films/<int:id>', views.FilmDetail.as_view(), name='films'),

    path('cinema/', views.CinemaList.as_view()),
    path('cinema/<int:id>', views.CinemaDetail.as_view(), name='Cinema'),

    path('title/<title>', views.film_title, name='filmtitles'),
    path('title/', views.film_title),

    path('film/<int:id>', views.filmcinema),
    path('film/<int:id>', views.filmgenre),

    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
```

A ce niveau, l'application propose un URI : **`http://api.yourcompany.com/films/`**

Lorsqu'un client accédera à cette adresse, Django ira lire le fichier **`url.py`** qui rédigera le client vers la vue : **`FilmList`**

- Dans **FilmManagement\urls.py** on ajoute des path et des url pour faire interagir notre api

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('filmسينما.urls')),
]
```

Le fichier **url.py** a pour but de gérer le routage de l'application. Pour faire simple, lorsqu'un client appelle une ressource via un URI, ce fichier permet de faire la liaison entre l'extension de l'adresse et la vue adéquate

TEST

1- Tester notre api avec Postman en appliquant le JWTAuthentication

➔ L'utilisation de JWT nécessite un package

```
➤ pip install djangoRESTframework_simplejwt
```

➤ Pour appliquer le JWTAuthentication on doit ajouter à la fin de notre settings.py la configuration :

```
#JWT
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': ('rest_framework.permissions.IsAuthenticated',),
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
}
```

Cette méthode allow post seulement pour récupérer le Token

POST `http://127.0.0.1:8000/api/api/token/`

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookie

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE	DESCRIPTION
✓ username	admin	
✓ password	pass	
Key	Value	

On passe comme argument le nom d'utilisateur et le mot de passe de notre super User

Status: 200 OK Time: 112 ms Size: 650 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "refresh": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.  
    eyJ0b2t1bWVudCI6ImVmcmVzaCIsImV4cCI6MTYxMzc0NDkxMCwianRpIjoiaGVhZmVudCJlcnRpbGoiOjQyX2kiOjox  
    fQ.FCv-8sNuziGC-pTd4wtftJoEjn3zZC-ywPXycCm7NPU",  
3   "access": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.  
    eyJ0b2t1bWVudCI6ImVhbnRlcjZlcnRpbGoiOjQyX2kiOjoxfQ.Rp307DeuZMQyo3qAmz5UnSRF8iyLixNa2EAHTrEQY"  
4 }
```

Le Token Access expire après 5min.

Encoded

PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1b190eXB1Ijo1YWNjZXNzIiwiaXNjaXhwIjozNjEzNzAwODU4LCJqdGkiOiIxZmU2N2NiODQ0Y2E0YmJhOWJmNzA1MTYzNzVhbnZMSIsInVzZXJfaWQiOi0jF9.d9kqhrJHFY2MYASmqgSB6CdQb5ds05BJkhnA9wEuL0E
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

PAYLOAD: DATA

```
{
  "token_type": "access",
  "exp": 1613708858,
  "jti": "1fe67cb844ca4bba9bf78516375a6331",
  "user_id": 1
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
)
```

☒ secret base64 encoded

Signature Verified

SHARE JWT

Dans le site officiel de jwt.io

On peut tester la validité de notre Token qui est encodé sur Base64

Pour visualiser nos données ou faire des manipulations on doit

GET

http://127.0.0.1:8000/api/films/

Send

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

TYPE

Bearer Token

Inherit auth from parent

No Auth

API Key

Bearer Token

Basic Auth

Digest Auth

OAuth 1.0

OAuth 2.0

Hawk Authentication

AWS Signature

NTLM Authentication [Beta]

Akamai EdgeGrid

Token

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1b190eXB1Ijo1YWNjZXNzIiwiaXNjaXhwIjozNjEzNzAwODU4LCJqdGkiOiIxZmU2N2NiODQ0Y2E0YmJhOWJmNzA1MTYzNzVhbnZMSIsInVzZXJfaWQiOi0jF9.d9kqhrJHFY2MYASmqgSB6CdQb5ds05BJkhnA9wEuL0E

Status: 200 OK

Time: 14 ms

Size: 360 B

Save Response

Puis on le colle dans ce champ

Dans Autorisation on choisit « Bearer Token »

➤ Test sur FILMS

Testons la méthode POST

POST http://127.0.0.1:8000/api/films/ Send

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	title	KIMINO	
<input checked="" type="checkbox"/>	year_prod	2010	
<input checked="" type="checkbox"/>	genre	3	
<input checked="" type="checkbox"/>	presentation_date	2021-02-19T02:13:38	
<input checked="" type="checkbox"/>	owner	admin	

On peut poster les données par Json ou x-www-form

Body Cookies Headers (7) Test Results Status: 201 Created Time: 12 ms Size: 322 B Save Response

Pretty Raw Preview Visualize JSON ⌵ ⌵

```
1 {
2   "id": 5,
3   "title": "KIMINO",
4   "year_prod": 2010,
5   "genre": 3,
6   "presentation_date": "2021-02-19T02:13:38Z"
7 }
```

GET http://127.0.0.1:8000/api/films/ Send

Params **Authorization** Headers (7) Body Pre-request Script Tests Settings

TYPE

Bearer Token ⌵

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Token eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1bi90eXBlljoiYWNjZXNzIiwiaWF0Ijoi

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Body Cookies Headers (7) Test Results Status: 200 OK Time: 19 ms Size: 783 B Save Response

Pretty Raw Preview Visualize JSON ⌵ ⌵

```
1 [
2   {
3     "id": 2,
4     "title": "Catch Me If You Can",
5     "year_prod": 2012,
6     "genre": {
7       "id": 2,
8       "name": "Comedy"
9     },
10    "presentation_date": "2021-02-19T02:13:38Z",
11    "owner": "admin"
12  },
13  {
14    "id": 3,
15    "title": "Dragons",
16    "year_prod": 1999,
17    "genre": {
18      "id": 4,
```

Testons la méthode GET

Testons la méthode PUT

http://127.0.0.1:8000/api/films/3/ Save Send

PUT ▼ http://127.0.0.1:8000/api/films/3/ Send

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	id	5	
<input checked="" type="checkbox"/>	title	Dragons fist	
<input checked="" type="checkbox"/>	year_prod	2010	
<input checked="" type="checkbox"/>	genre	2	
<input checked="" type="checkbox"/>	presentation_date	2021-02-19T13:54:36.246634Z	

Body Cookies Headers (7) Test Results ⌐ Status: 200 OK Time: 19 ms Size: 345 B Save Response

Pretty Raw Preview Visualize JSON ▼ ↺

```

1  [
2    {
3      "id": 3,
4      "title": "Dragons fist",
5      "year_prod": 2010,
6      "genre": 2,
7      "presentation_date": "2021-02-19T13:54:36.246634Z"
8    }
9  ]

```

PATCH ▼ http://127.0.0.1:8000/api/films/2/ Send

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

	KEY	VALUE	DESCRIPTION
<input type="checkbox"/>	id	5	
<input type="checkbox"/>	title	Dragons fist	
<input checked="" type="checkbox"/>	year_prod	2020	
<input type="checkbox"/>	genre	2	
<input type="checkbox"/>	presentation_date	2021-02-19T13:54:36.246634Z	

Body Cookies Headers (7) Test Results ⌐ Status: 200 OK Time: 13 ms Size: 345 B Save Response

Pretty Raw Preview Visualize JSON ▼ ↺

```

1  [
2    {
3      "id": 2,
4      "title": "KIMINO Yo i2",
5      "year_prod": 2020,
6      "genre": 2,
7      "presentation_date": "2021-02-19T13:54:36.246634Z"
8    }
9  ]

```

Testons la méthode PATCH

⇒ Permet de modifier un champ précis

DELETE ▼ http://127.0.0.1:8000/api/films/3/ Send

Params Authorization ● Headers (9) ● **Body** ● Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL

	KEY	VALUE	DESCRIPTION	...	Bul
<input checked="" type="checkbox"/>	id	5			
<input checked="" type="checkbox"/>	title	Dragons fist			
<input checked="" type="checkbox"/>	year_prod	2010			
<input checked="" type="checkbox"/>	genre	2			
<input checked="" type="checkbox"/>	presentation_date	2021-02-19T13:54:36.246634Z			

Body Cookies Headers (6) Test Results 🌐 Status: 204 No Content Time: 16 ms Size: 211 B Save Respo

Pretty Raw Preview Visualize Text ▼ ≡

1

http://127.0.0.1:8000/api/films/2/ Save ▼ ✎

OPTIONS ▼ http://127.0.0.1:8000/api/films/2/ Send

Params Authorization ● Headers (9) ● **Body** ● Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL

	KEY	VALUE	DESCRIPTION	...	Bu
<input type="checkbox"/>	id	5			

Body Cookies Headers (7) Test Results 🌐 Status: 200 OK Time: 15 ms Size: 900 B Save Respo

Pretty Raw Preview Visualize JSON ▼ ≡

```

1  [
2    {
3      "name": "Film Detail",
4      "description": "",
5      "renders": [
6        {
7          "application/json",
8          "text/html"
9        },
10     ],
11     "parses": [
12       {
13         "application/json",
14         "application/x-www-form-urlencoded",
15         "multipart/form-data"
16       },
17     ],
18     "actions": {
19       "PUT": {
20         "id": {
21           "type": "integer",
22           "required": false,
23           "read_only": true,
24           "label": "ID"
25         },
26         "title": {
27           "type": "string",
28           "required": true,
29           "read_only": false,
30           "label": "Title",
31           "max_length": 100
32         },
33         "year_prod": {

```

Testons la méthode OPTIONS

Pour savoir quel méthodes un serveur support

➤ Test sur User

Testons la méthode GET by ID

GET

Params Authorization Headers (9) Body Pre-request Script Tests Settings

TYPE

Bearer Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Token

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1b90eXBlljoYWNjZXRzIj0.eyJ0b2t1b90eXBlljoYWNjZXRzIj0.

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 21 ms Size: 290 B Save Re

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 2,
3   "username": "admin1",
4   "films": [
5     4
6   ],
7   "cinemas": [
8     4
9   ]
10 }
```

GET

Params Authorization Headers (7) Body Pre-request Script Tests Settings

TYPE

Bearer Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Token

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1b90eXBlljoYWNjZXRzIj0.eyJ0b2t1b90eXBlljoYWNjZXRzIj0.

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 14 ms Size: 339 B Save Re

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 1,
4     "username": "admin",
5     "films": [
6       2,
7       3,
8       1
9     ],
10    "cinemas": [
11      2,
12      3
13    ]
14  },
15  {
16    "id": 2,
17    "username": "admin1",
18    "films": [
```

GET tous les utilisateurs

PUT : modifier user dont id = 2

PUT http://127.0.0.1:8000/api/users/2/ Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

<input checked="" type="checkbox"/>	username	admin1-1	
<input checked="" type="checkbox"/>	films	2	
<input checked="" type="checkbox"/>	cinemas	4	
<input type="checkbox"/>			
<input type="checkbox"/>			
<input type="checkbox"/>			
	Key	Value	Description

Body Cookies Headers (7) Test Results Status: 200 OK Time: 44 ms Size: 292 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 2,
3   "username": "admin1-1",
4   "films": [
5     2
6   ],
7   "cinemas": [
8     4
9   ]
10 }
```

Modifier le nom d'utilisateur
par la méthode PATCH

PATCH http://127.0.0.1:8000/api/users/2/ Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

<input checked="" type="checkbox"/>	username	admin1-2	
<input type="checkbox"/>	films	2	
<input type="checkbox"/>	cinemas	4	
	Value		Description

Body Cookies Headers (7) Test Results Status: 200 OK Time: 19 ms Size: 292 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 2,
3   "username": "admin1-2",
4   "films": [
5     2
6   ],
7   "cinemas": [
8     4
9   ]
10 }
```

DELETE http://127.0.0.1:8000/api/users/2/ Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Headers 7 hidden

KEY	VALUE	DESCRIPTION		Bulk Edit
Key	Value	Description		

Body Cookies Headers (6) Test Results Status: 204 No Content Time: 71 ms Size: 211 B Save Response

Pretty Raw Preview Visualize Text

```
1
```

La méthode DELETEE
supprimer user avec id :2

➤ Test sur Cinema

La méthode POST

Ajouter un cinéma avec le nom yourHome et ville : Casablanca et les films avec id : 4, 5 et utilisateur 1

POST `http://127.0.0.1:8000/api/cinema/`

Params Authorization Headers (11) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   .... "name": "YourHome",
3   .... "city": "CASABLANCA",
4   .... "films": [4,5],
5   .... "owner": 1
6 }
7
8
9
10
```

Body Cookies Headers (7) Test Results

Status: 201 Created Time: 49 ms Size: 297 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   .... "id": 5,
3   .... "name": "YourHome",
4   .... "city": "CASABLANCA",
5   .... "films": [
6     .... 4,
7     .... 5
8   ],
9   .... "owner": 1
10 }
```

GET `http://127.0.0.1:8000/api/cinema/`

Params **Authorization** Headers (7) Body Pre-request Script Tests Settings

TYPE

Bearer Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1biI9OiJ0eXBiljoIYWNjZXNz

! Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 16 ms Size: 980 B Save Response

Pretty Raw Preview Visualize JSON

```
18 {
19   .... "id": 3,
20   .... "name": "STARTS",
21   .... "city": "MEKNAS",
22   .... "films": [
23     .... {
24       .... "id": 2,
25       .... "title": "Catch Me If You Can",
26       .... "year_prod": 2012,
27       .... "presentation_date": "2021-02-19T02:13:38Z",
28       .... "genre": 2,
29       .... "owner": 1
30     },
31     .... {
32       .... "id": 3,
33       .... "title": "Dragons",
34       .... "year_prod": 1999,
35       .... "presentation_date": "2021-02-19T02:14:50Z",
```

Lister tous les cinémas

Modifier la ville à Fès

PATCH http://127.0.0.1:8000/api/cinema/2/ Send

Params Authorization Headers (11) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	city	Fes	
	Key	Value	Description

Body Cookies Headers (7) Test Results Status: 200 OK Time: 21 ms Size: 296 B Save Res

Pretty Raw Preview Visualize JSON

```
1 [
2   ... "id": 2,
3   ... "name": "ATLAS",
4   ... "city": "Fes",
5   ... "films": [
6     ... 4,
7     ... 5
8   ],
9   ... "owner": 1
10 ]
```

PUT http://127.0.0.1:8000/api/cinema/3/ Send

Params **Authorization** Headers (11) Body Pre-request Script Tests Settings

TYPE

Bearer Token

The authorization header will be automatically generated when you send

Token eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1biI9Oi90eXBlljoiYWNjZXNzIiwiaWF0Ij0i

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Body Cookies Headers (7) Test Results Status: 200 OK Time: 37 ms Size: 306 B Save Res

Pretty Raw Preview Visualize JSON

```
1 [
2   ... "id": 3,
3   ... "name": "AtlasVIP",
4   ... "city": "CASABLANCA",
5   ... "films": [
6     ... 4,
7     ... 5
8   ],
9   ... "owner": 1
10 ]
```

PUT On passant tous les champs

DELETE http://127.0.0.1:8000/api/cinema/2/ Send

Params **Authorization** Headers (11) Body Pre-request Script Tests Settings

TYPE

Bearer Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Token eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1biI9Oi90eXBlljoiYWNjZXNzIiwiaWF0Ij0i

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Body Cookies Headers (6) Test Results Status: 204 No Content Time: 18 ms Size: 211 B Save Res

Pretty Raw Preview Visualize Text

```
1
```

DELETE le cinéma id : 2

31

➤ Test sur Genre

Passer un genre avec
Name : art

POST ⌵ http://127.0.0.1:8000/api/genre/ Send

Params Authorization ● Headers (11) Body ● Pre-request Script Tests Settings

	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Pre
<input checked="" type="checkbox"/>	id	6				
<input checked="" type="checkbox"/>	name	art				
	Key	Value	Description			

Body Cookies Headers (7) Test Results 🌐 Status: 201 Created Time: 16 ms Size: 248 B Save Res

Pretty Raw Preview Visualize JSON ⌵ ≡

```
1 [
2   ... "id": 6,
3   ... "name": "Art"
4 ]
```

GET ⌵ http://127.0.0.1:8000/api/genre/1/ Send

Params Authorization ● Headers (7) Body Pre-request Script Tests Settings

TYPE

Bearer Token ⌵

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#) ➤

⚠ Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#) ➤

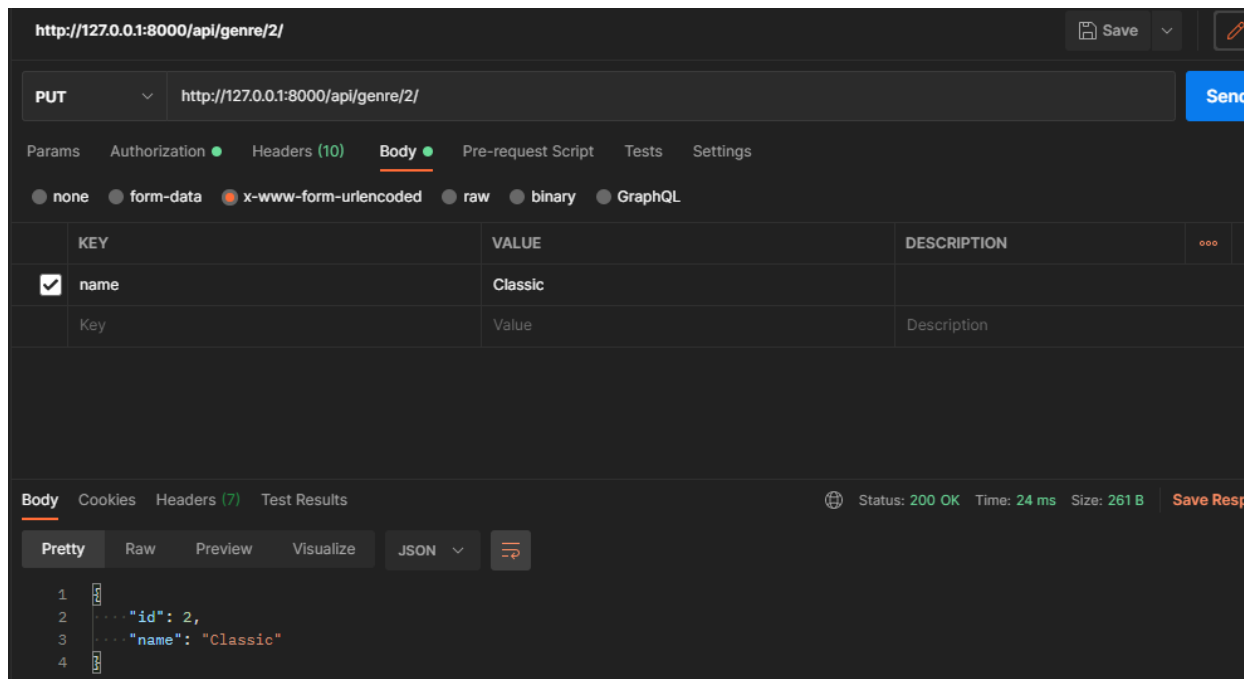
Token eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1biI9OiBjbGoiYWNjZXM

Body Cookies Headers (7) Test Results 🌐 Status: 200 OK Time: 9 ms Size: 323 B Save

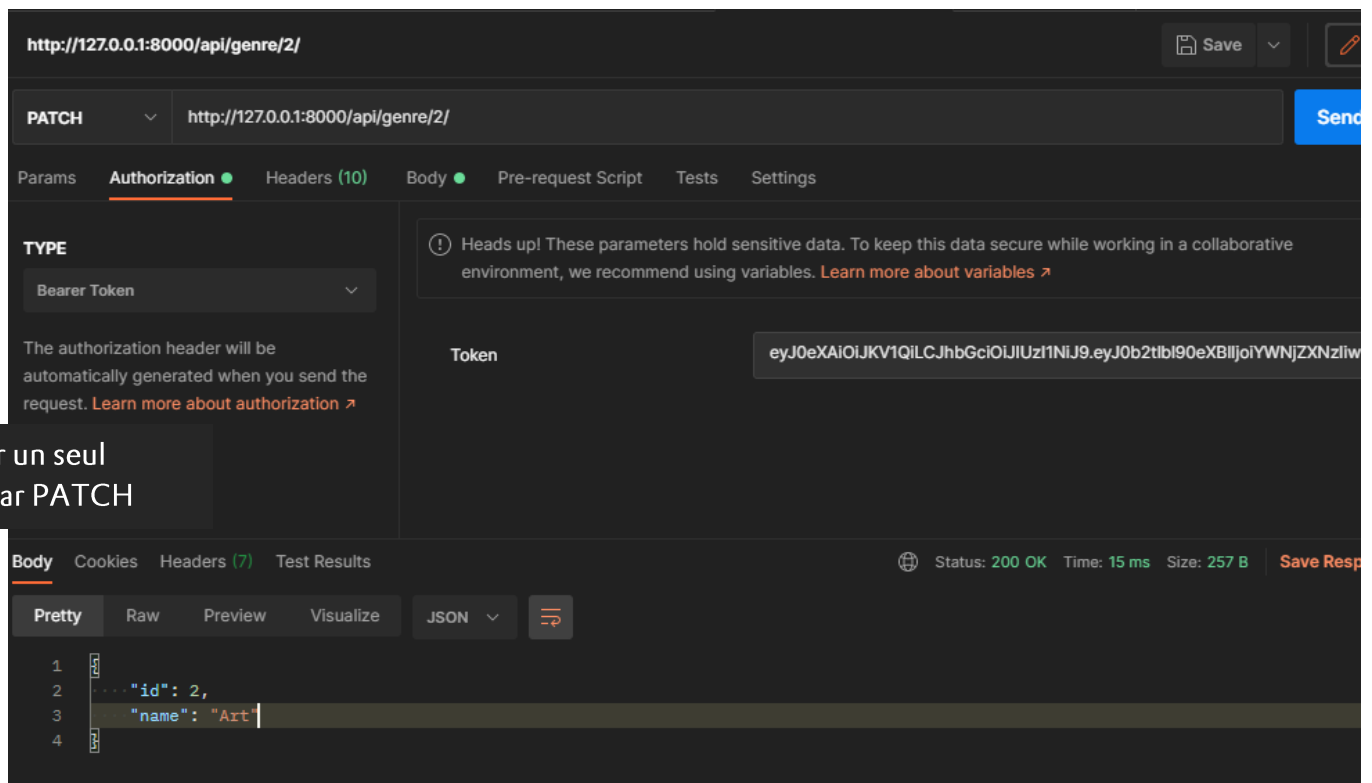
Pretty Raw Preview Visualize JSON ⌵ ≡

```
1 [
2   {
3     ... "id": 4,
4     ... "name": "action"
5   },
6   {
7     ... "id": 2,
8     ... "name": "Comedy"
9   },
10  {
11    ... "id": 3,
12    ... "name": "Love"
13  },
14  {
15    ... "id": 5,
16    ... "name": "science"
17  }
18 ]
```

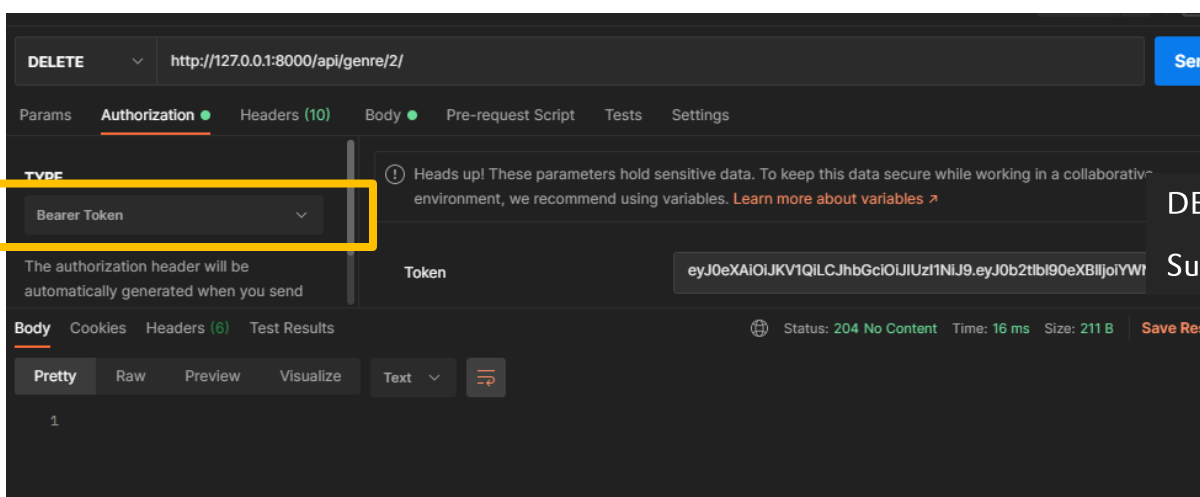
Lister tous les genres



PUT Méthode



Modifier un seul
champ par PATCH



DELETE :

Supprimer le genre a id : 2

➤ Test sur TITLE

Cherchant les films
contenant le nom
« dragon » dans le titre.

GET `http://127.0.0.1:8000/api/title/dragon` Send

Params Authorization Headers (11) Body Pre-request Script Tests Settings

TYPE

Bearer Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Token `eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1bi90eXBlljoiYWVjZXNzIiwiaWF0Ijoi`

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "title": "How to Train Your Dragon",
4     "year_prod": 2010,
5     "presentation_date": "2021-02-19T16:32:37.778053Z"
6   },
7   {
8     "title": "The Girl with the Dragon Tattoo",
9     "year_prod": 2011,
10    "presentation_date": "2021-02-19T16:32:37.778053Z"
11  },
12  {
13    "title": "How to Train Your Dragon 2",
14    "year_prod": 2014,
15    "presentation_date": "2021-02-19T16:32:37.778053Z"
16  },
17  {
18    "title": "Crouching Tiger, Hidden Dragon",
19    "year_prod": 2000
20  }
21 ]
```

Faisant appeler à une API externe qui permet de récupérer les titres et l'année de production avec le code dans la partie 9, on peut directement les lister et remplir notre base de données par la suite.

GET `http://127.0.0.1:8000/api/title/love/` Send

Params Authorization Headers (11) Body Pre-request Script Tests Settings

Type

Bearer Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Token `eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1bi90eXBlljoiYWVjZXNzIiwiaWF0Ijoi`

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "title": "Crazy, Stupid, Love.",
4     "year_prod": 2011,
5     "presentation_date": "2021-02-20T20:15:47.455584Z"
6   },
7   {
8     "title": "Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb",
9     "year_prod": 1964,
10    "presentation_date": "2021-02-20T20:15:47.455584Z"
11  },
12  {
13    "title": "Love Actually",
14    "year_prod": 2003,
15    "presentation_date": "2021-02-20T20:15:47.455584Z"
16  },
17  {
18    "title": "Shakespeare in Love",
19    "year_prod": 1998,
20    "presentation_date": "2021-02-20T20:15:47.455584Z"
21  },
22  {
23    "title": "The Love Guru",
24    "year_prod": 2008,
25    "presentation_date": "2021-02-20T20:15:47.455584Z"
26  }
27 ]
```

Cherchant les films
contenant le nom
« love » dans le titre.

➤ **Join FILM et CINEMA**

Affichant tous les cinémas qui vont présenter un film passé par url

```
@api_view(['GET'])
def filmcinema(request,id):
    cinema = Cinema.objects.filter(films__id=id)
    serializer = CinemaSerializer(cinema,many=True)
    return Response(serializer.data)
```

Pour effectuer la jointure on doit appeler la méthode `filter()` en passant id de l' autre table et `api_view` directement détecte la relation entre les deux tables

```
path('filc/<int:id>', views.filmcinema),
```

Pour retourner les résultats on doit créer

The screenshot shows the Postman interface for a REST client. At the top, a GET request is configured to `http://127.0.0.1:8000/api/fil/5`. The "Authorization" tab is active, displaying a Bearer Token. Below the token, there's a warning about sensitive data. The "Body" tab is also visible, showing the response in JSON format. The response is a detailed object for film ID 5, including its title "KIMINO", year of production (2010), presentation date, genre, and owner.

```

{
  "id": 5,
  "name": "KIMINO",
  "city": "PARIS",
  "films": [
    {
      "id": 4,
      "title": "Infenity",
      "year_prod": 2018,
      "presentation_date": "2021-02-15T02:17:22Z",
      "genre": 5,
      "owner": 1
    },
    {
      "id": 5,
      "title": "KIMINO",
      "year_prod": 2010,
      "presentation_date": "2021-02-19T02:13:38Z",
      "genre": 3,
      "owner": 1
    }
  ]
}

```

Les cinémas ayant le film id :
5 « KIMINO »

➤ JOIN FILM et GENRE

Affichant tous les films qui appartient à un genre passé par url

```
@api_view(['GET'])
def filmgenre(request,id):
    film = Film.objects.filter(genre__id=id)
    serializer = FilmSerializer(film,many=True)
    return Response(serializer.data)
```

Pour effectuer la jointure on doit appeler la méthode filter() en passant id de l' autre table et api_view directement détecte la relation entre les deux tables

FILM et GENRE

Urls.py

```
path('film/<int:id>/', views.filmgenre),
```

Pour retourner les résultats on doit créer

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://127.0.0.1:8000/api/filg/5/
- Authorization:** Bearer Token (Token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2t1bi90eXBlljoIYWNjZXNzliw)
- Status:** 200 OK, Time: 23 ms, Size: 350 B
- Response Body (JSON):**

```
[
  {
    "id": 4,
    "title": "Infenity",
    "year_prod": 2018,
    "genre": {
      "id": 5,
      "name": "science"
    },
    "presentation_date": "2021-02-15T02:17:22Z",
    "owner": "admin"
  }
]
```

Les films ayant le genre id :5 « science »

CONCLUSION

Nous avons vu comment Django REST Framework facilite la création de ressources d'une API REST, augmentant considérablement notre productivité.

Django REST Framework n'est pas limité aux seuls sujets traités dans ce rapport, dans la documentation DRF, vous avez de nombreuses autres fonctionnalités liées aux API REST telles que la limitation, la documentation, les validateurs, le contrôle de version et bien plus encore.

DRF a également des capacités pour prendre en charge le modèle d'API HATEOAS. Comme Django, Django REST Framework est extrêmement puissant, fiable et peut vous aider à résoudre vos problèmes avec le moins d'effort.