

CEF Learning

Créer un logiciel de gestion de médiathèque

Python Django

Emmanuelle Rousseau
05/08/2024

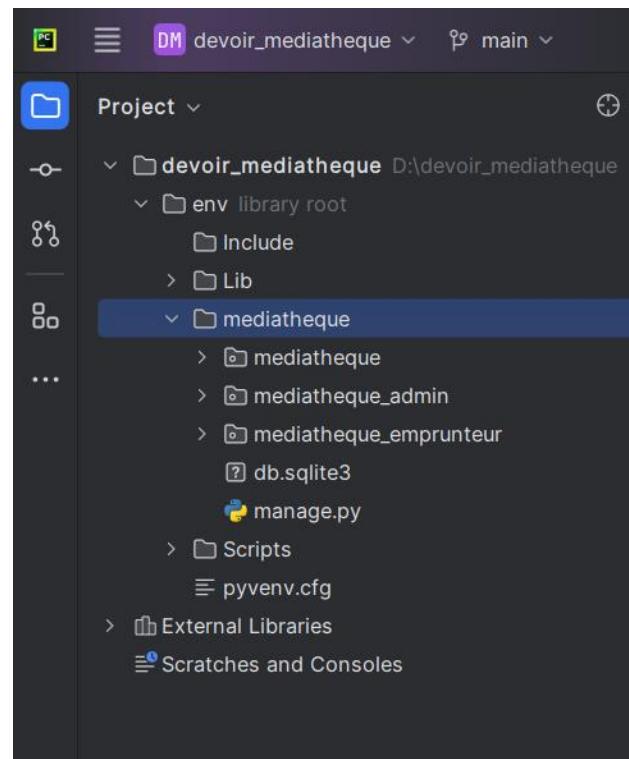
Introduction.....	2
L'application « mediatheque_admin ».....	3
Model Emprunteur	3
Model Media et enfants Livre, Cd, Dvd	4
Model Jeu_plateau	5
Views « mediatheque_admin ».....	6
Affichage des listes emprunteurs, médias et jeux de plateau.....	7
Ajouter/modifier/supprimer les emprunteurs.....	10
Ajouter/emprunter/retourner les jeux de plateau	12
Ajouter/emprunter/retourner un média	16
L'application « mediatheque_emprunteur »	21
Views « mediatheque_emprunteur »	21
Affichage des listes emprunteurs, médias et jeux de plateau.....	22
Conclusion	23

Introduction

Le présent document est destiné à exposer le processus que j'ai suivi pour créer les deux applications qui composent le logiciel de gestion de médiathèque en Python avec la librairie Django : « mediatheque ».

J'ai choisi de commencer par créer la partie destinée aux bibliothécaires que j'ai nommée : « mediatheque_admin ». Je définirai pour commencer les models, puis les views.

Nous verrons ensuite la partie destinée aux emprunteurs : « mediatheque_emprunteur ».



L'application « mediatheque_admin »

Pour débuter, nous allons voir les différents models que j'ai déterminés pour cette application :

- Emprunteur
- Media avec les enfants Livre, Cd, Dvd
- Jeu_plateau

Model Emprunteur

Le code initial prévoyait seulement les deux champs ci-dessous et ne précisait pas leur type.

```
class Emprunteur():
    name = ""
    bloque = ""
```

J'ai conçu la structure suivante :

- Champ name scindé en first_name et last_name avec des champs text charfield limités à 40 caractères
- Date de création en type date aaaa-mm-jj qui se remplit automatiquement à la date du jour
- Bloque en type booléen avec une valeur faux par défaut à la création

J'ai rajouté deux fonctions :

- Def statut_emprunteur qui définit une couleur en fonction du statut que j'ai repris dans le fichier html pour appliquer un style.
- Def __str__(self) pour afficher le nom et le prénom de l'emprunteur dans la page admin du site.

The screenshot shows the PyCharm IDE interface. On the left, the project structure is displayed with the 'devoir_mediatheque' project selected. Inside, there's a 'mediatheque' app folder containing 'migrations', 'templates', 'admin.py', 'apps.py', 'models.py', 'tests.py', 'urls.py', and 'views.py'. The 'models.py' file is currently open in the main editor window. The code in 'models.py' is as follows:

```
# Create your models here.

class Emprunteur(models.Model):
    first_name = models.CharField(max_length=40)
    last_name = models.CharField(max_length=40)
    date_creation = models.DateField(auto_now_add=True)
    bloque = models.BooleanField(default=False)

    def statut_emprunteur(self):
        if self.bloque == 1:
            return 'red'
        else:
            return 'green'

    def __str__(self):
        return self.first_name + ' ' + self.last_name
```

Créer un logiciel de gestion de médiathèque

id	first_name	last_name	date_creation	bloqué
1	Emile	Zola	2024-07-31	0
2	Elizabeth	George	2024-07-31	0
3	Paulo	Coelho	2024-07-31	0
4	JRR	Tolkien	2024-07-31	0

Model Media et enfants Livre, Cd, Dvd

Le code initial prévoyait une classe différente pour chaque type de media avec des champs qui se répétaient.

```
class livre():
    name = ""
    auteur = ""
    dateEmprunt = ""
    disponible = ""
    emprunteur = ""

class dvd():
    name = ""
    realisateur = ""
    dateEmprunt = ""
    disponible = ""
    emprunteur = ""

class cd():
    name = ""
    artiste = ""
    dateEmprunt = ""
    disponible = ""
    emprunteur = ""
```

J'ai défini la classe mère Media regroupant les champs identiques à chaque media :

1. Media_name en type texte d'une longueur maximum de 100 caractères
2. Disponible en type booléen avec une valeur faux par défaut à la création
3. Date d'emprunt en type date aaa-mm-jj qui est vide et null à la création
4. Emprunteur qui est lié à la table Emprunteur grâce à une foreign key, pour pouvoir choisir dans la liste des emprunteurs existants, et qui est vide et null à la création
5. Media en type texte qui se réfère à la liste de choix type_media étant donné qu'il n'existe que trois types pour ce logiciel exercice. Dans un contexte réel, il aurait été préférable de recourir à une table spécifique liée par une foreign key comme l'emprunteur.

Les classes enfant ne contiennent qu'un seul champ spécifique avec cependant un format identique.

J'ai rajouté une metaclass abstract. Au niveau de la base de données, cela permet de créer trois tables enfants différentes avec l'intégralité des champs.

Créer un logiciel de gestion de médiathèque

```

21 @1 class Media(models.Model):
22     type_media = models.TextChoices("type_media", "DVD CD LIVRE")
23     media_name = models.CharField(max_length=100)
24     media = models.CharField(choices=type_media, max_length=10)
25     date_emprunt = models.DateField(null=True, blank=True)
26     disponible = models.BooleanField(default=True)
27     emprunteur = models.ForeignKey(Emprunteur, null=True, blank=True, on_delete=models.CASCADE)

    # Cocotte123
    class Meta:
        abstract = True

9 usages  ▲ Cocotte123
33 class Livre(Media):
34     livre_auteur = models.CharField(max_length=100)

    # Cocotte123
37 class Cd(Media):
38     cd_artiste = models.CharField(max_length=100)

8 usages  ▲ Cocotte123
41 class Dvd(Media):
42     dvd_realisateur = models.CharField(max_length=100)

```

Nom	Type	Schéma
wango_session		CREATE TABLE "wango_se"
mediatheque_admin_cd		CREATE TABLE "mediatheq" "id" integer NOT NULL "media_name" varchar(100) "media" varchar(10) NOT N "date_emprunt" date "disponible" bool NOT NUL "cd_artiste" varchar(100) N "emprunteur_id" bigint
mediatheque_admin_dvd		CREATE TABLE "mediatheq" "id" integer NOT NULL "media_name" varchar(100) "media" varchar(10) NOT N "date_emprunt" date "disponible" bool NOT NUL "dvd_realisateur" varchar(1) "emprunteur_id" bigint
mediatheque_admin_emprunteur		CREATE TABLE "mediatheq"
mediatheque_admin_jeu_plateau		CREATE TABLE "mediatheq"
mediatheque_admin_livre		CREATE TABLE "mediatheq" "id" integer NOT NULL "media_name" varchar(100) "media" varchar(10) NOT N "date_emprunt" date "disponible" bool NOT NUL "livre_auteur" varchar(100) "emprunteur_id" bigint

Model Jeu_plateau

Le code initial prévoyait seulement les deux champs ci-dessous et ne précisait pas leur type.

```

class jeuDePlateau :
    name = ""
    createur = ""

```

J'ai repris les mêmes champs et les mêmes types que pour les tables medias :

Créer un logiciel de gestion de médiathèque

The screenshot shows two windows. The top window is PyCharm displaying the `models.py` file for the `mediatheque` application. The code defines a `Jeu_plateau` model with fields: `jeu_name`, `jeu_createur`, `date_emprunt`, `disponible`, and `emprunteur`. It also contains two methods: `statut_emprunt` and `retour_emprunt`. The bottom window is DB Browser for SQLite showing the `mediatheque_admin_jeu_plateau` table with three rows:

id	jeu_name	jeu_createur	date_emprunt	disponible	emprunteur_id
1	Les colons de catane	Klaus Teuber	2024-08-09	0	3
2	Jeu de l'cie	Inconnu	2024-08-01	0	3
3	Jeu de dame	Inconnu		NULL	1

J'ai rajouté deux fonctions qui seront utilisés dans le fichier html pour la gestion des boutons.

Nous allons maintenant aborder la construction des views de l'application « mediatheque admin ».

Views « mediatheque_admin »

Avant de décliner les différentes fonctionnalités, il est nécessaire de parler des paramétrages effectués au niveau du projet « mediatheque » :

1. On ajoute une url dans le fichier urls du projet « mediatheque » qui permet d'appeler « include » toutes les urls de l'application « mediatheque_admin »
2. On ajoute l'application dans le fichier settings

Créer un logiciel de gestion de médiathèque

The screenshot shows two code editors side-by-side. The left editor displays the file `urls.py` with the following content:

```
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     1. Add an import: from my_app import views
8     2. Add a URL to urlpatterns: path('', views.home, name='home')
9 Class-based views
10    1. Add an import: from other_app.views import Home
11    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
12 Including another URLconf
13    1. Import the include() function: from django.urls import include, path
14    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15 """
16
17 from django.contrib import admin
18 from django.urls import path, include
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path('', include('mediatheque_admin.urls')),
23     path('membre', include('mediatheque_emprunteur.urls')),
24 ]
25
```

The right editor displays the file `settings.py` with the following content:

```
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     'mediatheque_admin',
40     'mediatheque_emprunteur',
41 ]
```

J'ai également créé un fichier `home.html` dans le dossier.

Affichage des listes emprunteurs, médias et jeux de plateau

La fonction `def home` permet d'appeler les données contenues dans les tables `Emprunteur`, `Livre`, `Cd`, `Dvd` et `Jeu_plateau` en créant un context qui seront affichés dans le fichier `home.html`.

The screenshot shows a code editor displaying the file `views.py` with the following content:

```
1 from django.http import HttpResponseRedirect
2 from django.shortcuts import render, redirect
3 from .models import *
4
5 # Create your views here.
6
7 def home(request):
8     datas_emprunteur = Emprunteur.objects.all()
9     datas_media = Livre.objects.all().union(Cd.objects.all(), Dvd.objects.all())
10    datas_jeuplateau = Jeu_plateau.objects.all()
11
12    context = {
13        'datas_emprunteur': datas_emprunteur,
14        'datas_media': datas_media,
15        'datas_jeuplateau': datas_jeuplateau,
16    }
17
18
19    return render(request, template_name='home.html', context)
```

Créer un logiciel de gestion de médiathèque

```

from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('add_emprunteur/', views.add_emprunteur, name='add_emprunteur'),
    path('delete_emprunteur/<int:id>', views.delete_emprunteur, name='delete_emprunteur'),
    path('update_emprunteur/<int:id>', views.update_emprunteur, name='update_emprunteur'),
    path('add_media/', views.add_media, name='add_media'),
    path('add_jeuplateau/', views.add_jeuplateau, name='add_jeuplateau'),
    path('emprunt_jeuplateau_page/<int:pk>', views.emprunt_jeuplateau_page, name='emprunt_jeuplateau_page'),
    path('emprunt_jeuplateau/<int:id>', views.emprunt_jeuplateau, name='emprunt_jeuplateau'),
    path('retour_jeuplateau/<int:id>', views.retour_jeuplateau, name='retour_jeuplateau'),
    path('emprunt_livre_page/', views.emprunt_livre_page, name='emprunt_livre_page'),
    path('emprunt_livre/', views.emprunt_livre, name='emprunt_livre'),
    path('emprunt_cd_page/', views.emprunt_cd_page, name='emprunt_cd_page'),
    path('emprunt_cd/', views.emprunt_cd, name='emprunt_cd'),
    path('emprunt_dvd_page/', views.emprunt_dvd_page, name='emprunt_dvd_page'),
    path('emprunt_dvd/', views.emprunt_dvd, name='emprunt_dvd'),
    path('retour_livre_page/', views.retour_livre_page, name='retour_livre_page'),
    path('retour_livre/', views.retour_livre, name='retour_livre'),
    path('retour_cd_page/', views.retour_cd_page, name='retour_cd_page'),
    path('retour_cd/', views.retour_cd, name='retour_cd'),
    path('retour_dvd_page/', views.retour_dvd_page, name='retour_dvd_page'),
    path('retour_dvd/', views.retour_dvd, name='retour_dvd'),
]

```

Pour les emprunteurs et les jeux de plateau :

- On appelle toutes les données sans filtre : `datas_emprunteur = Emprunteur.objects.all()` et `datas_jeuplateau = Jeu_plateau.objects.all()`.
- On crée une boucle dans le fichier `home.html` en utilisant le context

```

<html lang="fr">
<body>
    <h3>Liste des emprunteurs</h3>
    <div class="row">
        <table style="width: 1000px; border: solid black 2px;">
            <thead>
                <tr>
                    <td style="width: 200px">Emprunteur Id</td>
                    <td style="width: 200px">Prénom</td>
                    <td style="width: 200px">Nom</td>
                    <td style="width: 200px">Date de création</td>
                    <td style="width: 200px">Bloqué</td>
                </tr>
            </thead>
            <tbody>
                {% for emprunteur in datas_emprunteur %}<br>
                <tr>
                    <td>{{emprunteur.id}}</td>
                    <td>{{emprunteur.first_name}}</td>
                    <td>{{emprunteur.last_name}}</td>
                    <td>{{emprunteur.date_creation}}</td>
                    <td style="background-color: {{emprunteur.statut_emprunteur}}>{{emprunteur.bloque}}

```

Médiathèque

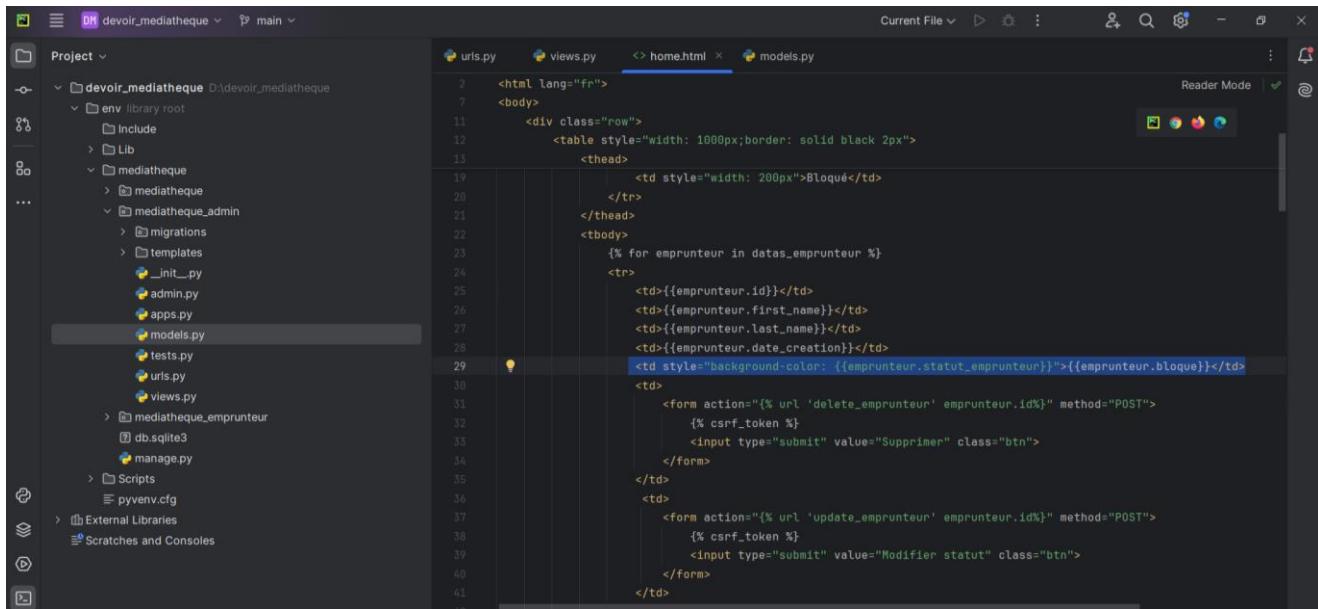
Administrateur

Liste des emprunteurs

Emprunteur Id	Prénom	Nom	Date de création	Bloqué		
1	Emile	Zola	July 31, 2024	False	Supprimer	Modifier statut
2	Elizabeth	George	July 31, 2024	False	Supprimer	Modifier statut
3	Paulo	Coelho	July 31, 2024	False	Supprimer	Modifier statut
4	JRR	Tolkien	July 31, 2024	False	Supprimer	Modifier statut

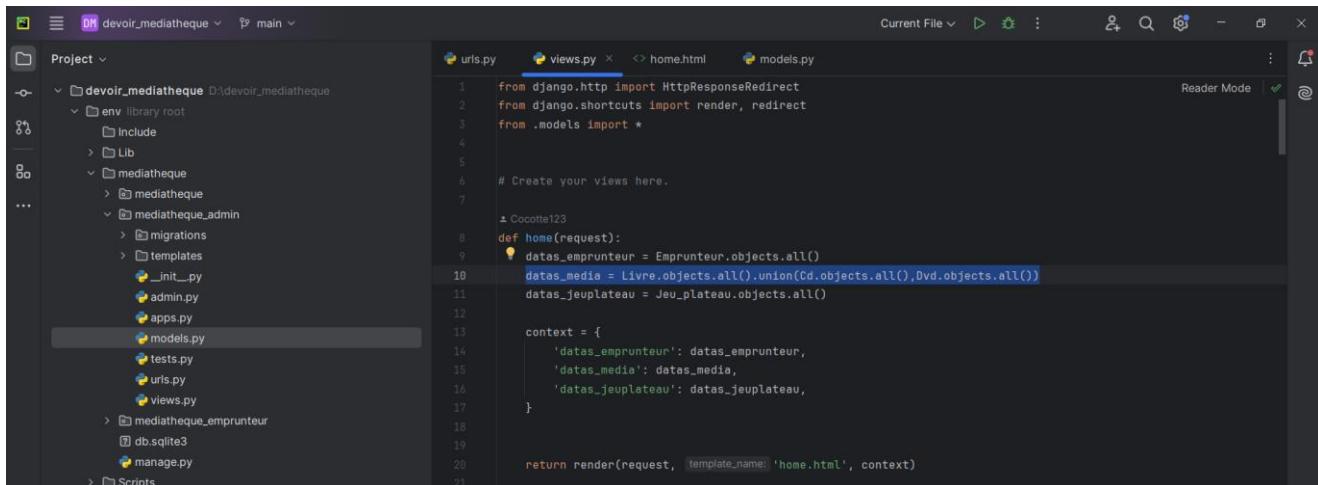
Au niveau du champ « Bloqué », on retrouve la fonction `def statut_emprunteur` au niveau du style :

Créer un logiciel de gestion de médiathèque



```
<html lang="fr">
    <body>
        <div class="row">
            <table style="width: 1000px; border: solid black 2px;">
                <thead>
                    <tr>
                        <td style="width: 200px">Bloqué</td>
                    </tr>
                </thead>
                <tbody>
                    {% for emprunteur in datas_emprunteur %}
                    <tr>
                        <td>{{emprunteur.id}}</td>
                        <td>{{emprunteur.first_name}}</td>
                        <td>{{emprunteur.last_name}}</td>
                        <td>{{emprunteur.date_creation}}</td>
                        <td style="background-color: {{emprunteur.statut_emprunteur}}">{{emprunteur.bloque}}
```

Pour les médias, on appelle les données de la table Livre, Cd et Dvd dans un seul contexte en utilisant la fonction union(). Cela est possible car ces tables ont la même structure.



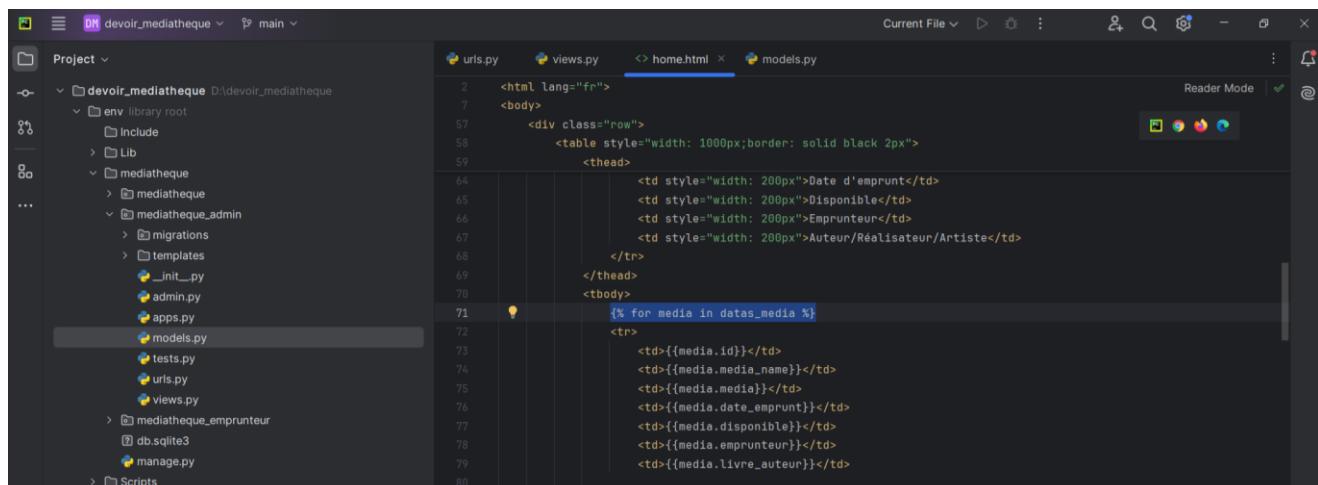
```
from django.http import HttpResponseRedirect
from django.shortcuts import render, redirect
from .models import *

# Create your views here.

def home(request):
    datas_emprunteur = Emprunteur.objects.all()
    datas_media = Livre.objects.all().union(Cd.objects.all(), Dvd.objects.all())
    datas_jeuplateau = Jeu_plateau.objects.all()

    context = {
        'datas_emprunteur': datas_emprunteur,
        'datas_media': datas_media,
        'datas_jeuplateau': datas_jeuplateau,
    }

    return render(request, template_name='home.html', context)
```



```
<html lang="fr">
    <body>
        <div class="row">
            <table style="width: 1000px; border: solid black 2px;">
                <thead>
                    <tr>
                        <td style="width: 200px">Date d'emprunt</td>
                        <td style="width: 200px">Disponible</td>
                        <td style="width: 200px">Emprunteur</td>
                        <td style="width: 200px">Auteur/Réalisateur/Artiste</td>
                    </tr>
                </thead>
                <tbody>
                    {% for media in datas_media %}
                    <tr>
                        <td>{{media.id}}</td>
                        <td>{{media.media_name}}</td>
                        <td>{{media.media_type}}</td>
                        <td>{{media.date_emprunt}}</td>
                        <td>{{media.disponible}}</td>
                        <td>{{media.emprunteur}}</td>
                        <td>{{media.livre_auteur}}</td>
                    
```

Créer un logiciel de gestion de médiathèque

Media id	Titre	Type	Date d'emprunt	Disponible	Emprunteur	Auteur/Réalisateur/Artiste
1	Harry Potter à l'école des sorciers	LIVRE	Aug. 14, 2024	False	Emile Zola	JK Rollings
1	La cinquième vague	CD	None	True	None	Adèle
1	Le cinquième élément	DVD	None	True	None	Luc Besson
2	Harry Potter et la chambre des secrets	LIVRE	None	True	None	JK Rollings
2	Un autre monde	CD	None	True	None	Téléphone

Ajouter/modifier/supprimer les emprunteurs

Ajouter

Pour ajouter un emprunteur, j'ai créé, dans le fichier `home.html`, un formulaire avec une méthode POST qui appelle l'url « `add_emprunteur` ».



```





```

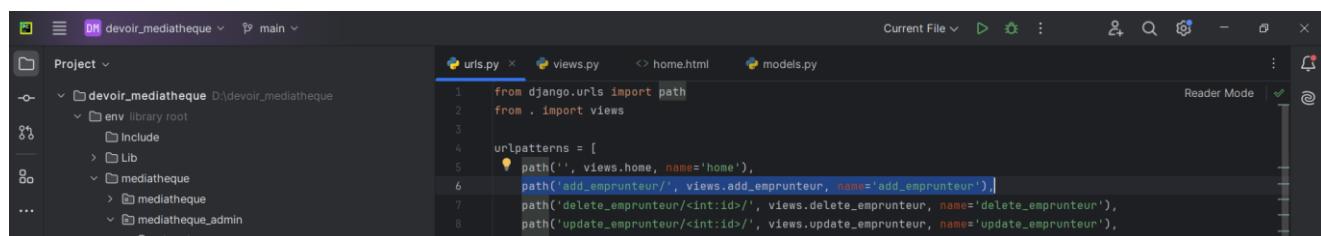
The screenshot shows a browser window with a title bar "Ajouter un emprunteur". Below it is a simple HTML form with two text input fields and a submit button. The code for this form is visible in the browser's developer tools:

```

<h3>Ajouter un emprunteur</h3>
<form action="{% url 'add_emprunteur'%}" method="POST">
    {% csrf_token %}
    <input type="text" placeholder="Prénom" name="first_name" required maxlength="40">
    <input type="text" placeholder="Nom" name="last_name" required maxlength="40">
    <button type="submit">Ajouter</button>
</form>

```

Cette url renvoie à la fonction `add_emprunteur` dans le fichier `views.py`.

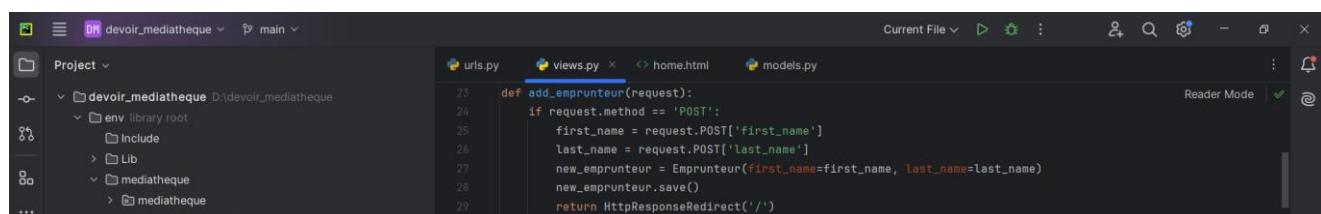


```

path('add_emprunteur/', views.add_emprunteur, name='add_emprunteur'),

```

La fonction `add_emprunteur` récupère les données des champs « `first_name` » et « `last_name` » depuis le formulaire via leur name, enregistre les données dans la table `Emprunteur` via la fonction `save()` et renvoie vers la page `home`.



```

def add_emprunteur(request):
    if request.method == 'POST':
        first_name = request.POST['first_name']
        last_name = request.POST['last_name']
        new_emprunteur = Emprunteur(first_name=first_name, last_name=last_name)
        new_emprunteur.save()
        return HttpResponseRedirect('/')

```

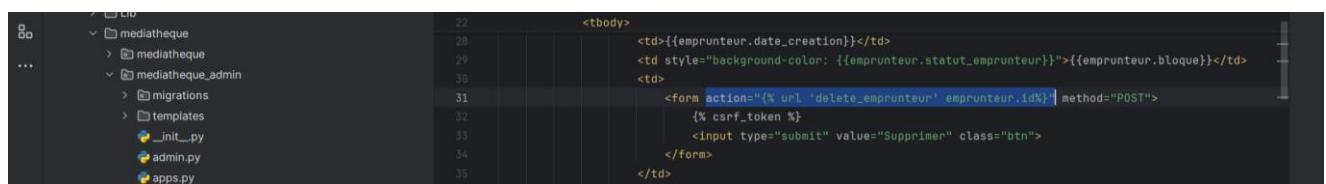
Supprimer

Créer un logiciel de gestion de médiathèque

Pour supprimer un emprunteur, j'ai créé, dans le fichier home.html, un formulaire avec une méthode POST qui appelle l'url « delete_emprunteur » en ajoutant l'id de l'emprunteur sélectionné.

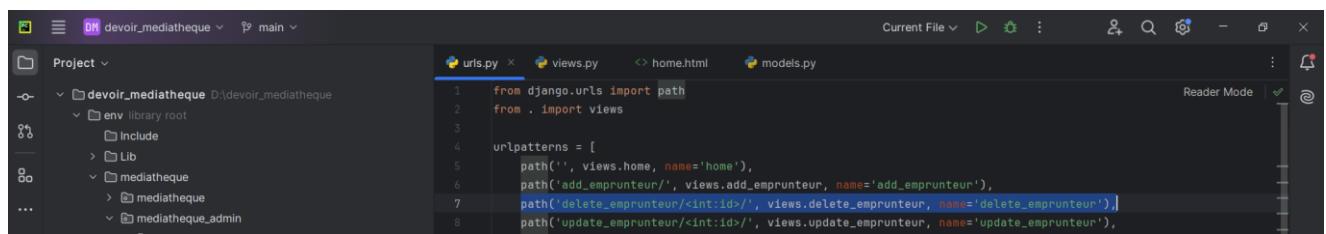
Liste des emprunteurs

Emprunteur Id	Prénom	Nom	Date de création	Bloqué	Supprimer	Modifier statut
1	Emile	Zola	July 31, 2024	False	Supprimer	Modifier statut
2	Elizabeth	George	July 31, 2024	False	Supprimer	Modifier statut
3	Paulo	Coelho	July 31, 2024	False	Supprimer	Modifier statut
4	JRR	Tolkien	July 31, 2024	False	Supprimer	Modifier statut



```
<tbody>
  <tr>
    <td>{{emprunteur.id}}</td>
    <td>{{emprunteur.prenom}}</td>
    <td>{{emprunteur.nom}}</td>
    <td>{{emprunteur.date_creation}}</td>
    <td>{{emprunteur.bloque}}</td>
    <td><a href="#" class="button-supprimer">Supprimer</a></td>
    <td><a href="#" class="button-modifier">Modifier statut</a></td>
  </tr>
</tbody>
```

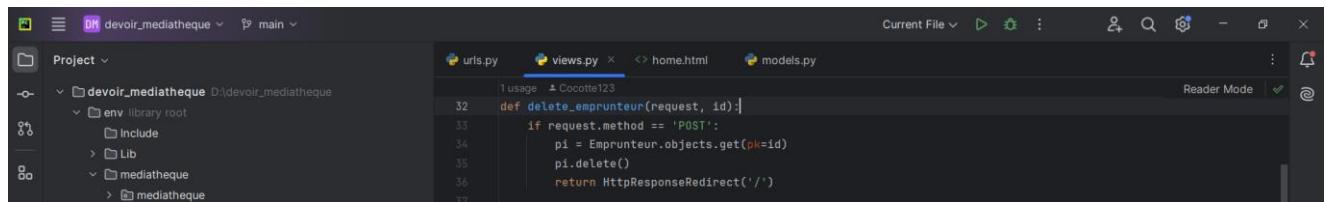
Cette url renvoie à la fonction delete_emprunteur dans le fichier views, ainsi que l'id de l'emprunteur.



```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('add_emprunteur/', views.add_emprunteur, name='add_emprunteur'),
    path('delete_emprunteur/<int:id>', views.delete_emprunteur, name='delete_emprunteur'),
    path('update_emprunteur/<int:id>', views.update_emprunteur, name='update_emprunteur'),
```

La fonction delete_emprunteur récupère les données du champ « id » depuis le formulaire, supprime l'enregistrement dans la table Emprunteur via la fonction delete() et renvoie vers la page home.



```
def delete_emprunteur(request, id):
    if request.method == 'POST':
        pi = Emprunteur.objects.get(pk=id)
        pi.delete()
    return HttpResponseRedirect('/')
```

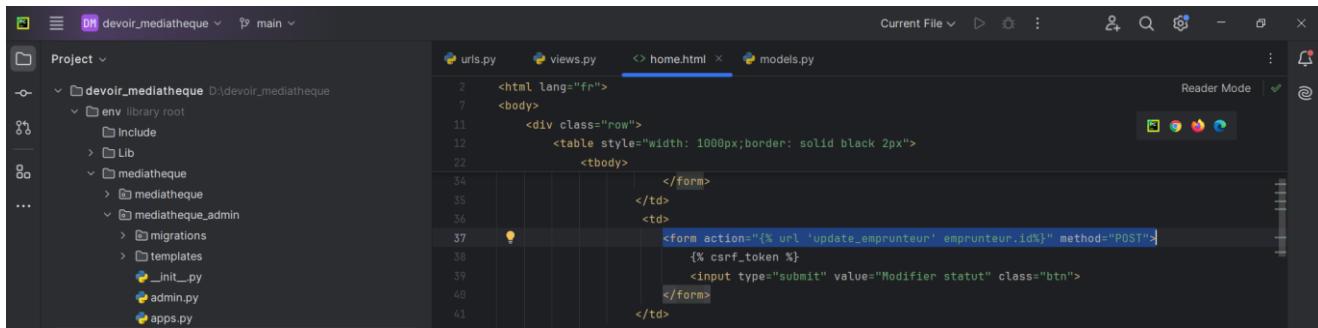
Modifier

La modification de l'emprunteur consiste simplement à passer son statut de non bloqué à non bloqué. Pour se faire, j'ai créé, dans le fichier home.html, un formulaire avec une méthode POST qui appelle l'url « update_emprunteur » en ajoutant l'id de l'emprunteur sélectionné, comme pour la suppression vue précédemment.

Liste des emprunteurs

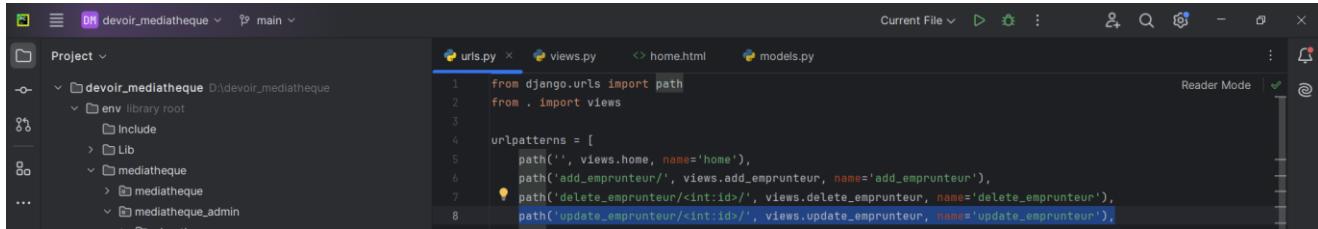
Emprunteur Id	Prénom	Nom	Date de création	Bloqué	Supprimer	Modifier statut
1	Emile	Zola	July 31, 2024	False	Supprimer	Modifier statut
2	Elizabeth	George	July 31, 2024	False	Supprimer	Modifier statut
3	Paulo	Coelho	July 31, 2024	False	Supprimer	Modifier statut
4	JRR	Tolkien	July 31, 2024	False	Supprimer	Modifier statut

Créer un logiciel de gestion de médiathèque



```
<html lang="fr">
    <body>
        <div class="row">
            <table style="width: 1000px; border: solid black 2px;">
                <tr>
                    <td>
                        <form action="{% url 'update_emprunteur' emprunteur.id %}" method="POST">
                            {% csrf_token %}
                            <input type="submit" value="Modifier statut" class="btn">
                        </form>
                    </td>
                </tr>
            </table>
        </div>
    </body>
</html>
```

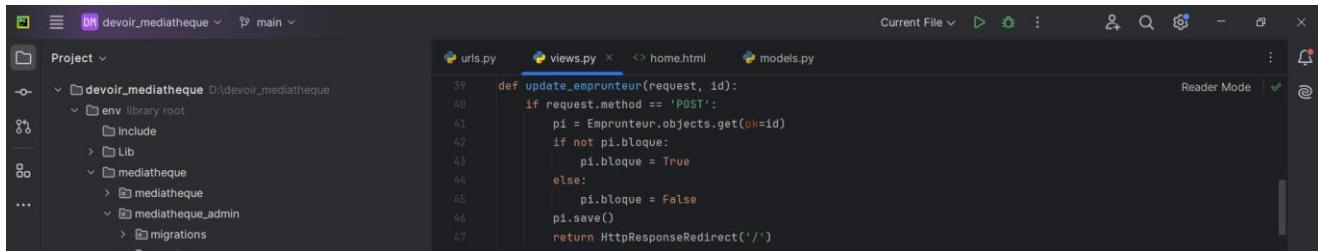
Cette url renvoie à la fonction update_emprunteur dans le fichier views, ainsi que l'id de l'emprunteur.



```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('add_emprunteur/', views.add_emprunteur, name='add_emprunteur'),
    path('delete_emprunteur/<int:id>', views.delete_emprunteur, name='delete_emprunteur'),
    path('update_emprunteur/<int:id>', views.update_emprunteur, name='update_emprunteur'),
```

La fonction update_emprunteur récupère les données du champ « id » depuis le formulaire. Elle teste la valeur du champ « bloque » et le modifie. Si le champ est à true, il passe à false et inversement. On sauvegarde ce changement dans la table Emprunteur via la fonction save() et on renvoie vers la page home.



```
def update_emprunteur(request, id):
    if request.method == 'POST':
        pi = Emprunteur.objects.get(pk=id)
        if not pi.bloque:
            pi.bloque = True
        else:
            pi.bloque = False
        pi.save()
    return HttpResponseRedirect('/')
```

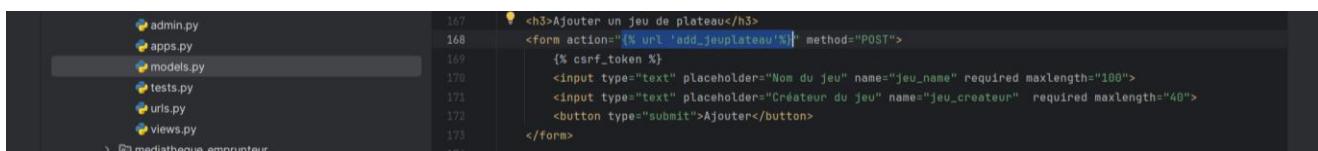
Ajouter/emprunter/retourner les jeux de plateau

Ajouter

Pour ajouter un jeu de plateau, j'ai reproduit le même processus que pour l'ajout d'un emprunteur : ajout d'un formulaire qui appelle l'url add_jeuplateau, qui appelle elle-même la fonction add_jeuplateau en transmettant les champs « jeu_name » et « jeu_createur ».

Ajouter un jeu de plateau

Nom du jeu Créateur du jeu Ajouter



```
<h3>Ajouter un jeu de plateau</h3>
<form action="{% url 'add_jeuplateau' %}" method="POST">
    {% csrf_token %}
    <input type="text" placeholder="Nom du jeu" name="jeu_name" required maxlength="100">
    <input type="text" placeholder="Créateur du jeu" name="jeu_createur" required maxlength="40">
    <button type="submit">Ajouter</button>
</form>
```

Créer un logiciel de gestion de médiathèque

The image shows two side-by-side code editors within a development environment. Both editors have tabs for 'urls.py', 'views.py', and 'home.html'. The top editor shows the 'urls.py' file with URL patterns for various actions like home, adding or deleting borrowers, updating loans, adding media, and adding or updating game plates. The bottom editor shows the 'views.py' file with a function 'add_jeuplateau' that handles POST requests to add a new game plate with a specific name and creator.

```
urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('add_emprunteur/', views.add_emprunteur, name='add_emprunteur'),
    path('delete_emprunteur/<int:id>', views.delete_emprunteur, name='delete_emprunteur'),
    path('update_emprunteur/<int:id>', views.update_emprunteur, name='update_emprunteur'),
    path('add_media/', views.add_media, name='add_media'),
    path("add_jeuplateau/", views.add_jeuplateau, name='add_jeuplateau'),
    path('emprunt_jeuplateau_page/<int:pk>', views.emprunt_jeuplateau_page, name='emprunt_jeuplateau_page'),
]
```

```
views.py
def add_jeuplateau(request):
    if request.method == 'POST':
        jeu_name = request.POST['jeu_name']
        jeu_createur = request.POST['jeu_createur']
        new_jeuplateau = Jeu_plateau(jeu_name=jeu_name, jeu_createur=jeu_createur)
        new_jeuplateau.save()
        return HttpResponseRedirect('/')
```

Emprunter

Pour emprunter un jeu de plateau, il faut sélectionner un emprunteur, mettre une date d'emprunt et passer son statut à non-disponible.

Pour ce faire, j'ai ajouté, dans le fichier home.html, un lien qui appelle l'url « emprunt_jeuplateau_page » en ajoutant l'id du jeu sélectionné.

The image shows a code editor displaying the 'home.html' template. It contains an HTML table that lists games from a database. For each game, there is a link labeled 'Emprunter' which, when clicked, would trigger a POST request to the 'emprunt_jeuplateau_page' view with the game's ID as a parameter. The 'models.py' file is also visible in the project tree on the left.

```
<html lang="fr">
<body>
    <div class="row">
        <table style="width: 1000px; border: solid black 2px;">
            <% for jeu in datas_jeuplateau %>
                <tr>
                    <td>{{jeu.id}}</td>
                    <td>{{jeu.jeu_name}}</td>
                    <td>{{jeu.date_emprunt}}</td>
                    <td>{{jeu.disponible}}</td>
                    <td>{{jeu.emprunteur}}</td>
                    <td>{{jeu.jeu_createur}}</td>
                    <td>
                        <a href="{% url 'emprunt_jeuplateau_page' jeu.id%}">
                            <button style="visibility: {{jeu.statut_emprunt2}}">Emprunter</button>
                        </a>
                    </td>
                </tr>
            <% endfor %>
        </table>
    </div>
</body>

```

A noter, il faut que cette ligne soit disponible. Par conséquent, le bouton « Emprunter » s'affiche uniquement si le champ Disponible est à true. Pour cela, on applique la fonction statut_emprunt2 définie dans le model Jeu_plateau :

Créer un logiciel de gestion de médiathèque

```

12 usages à Cocotte123
class Jeu_plateau(models.Model):
    jeu_name = models.CharField(max_length=100)
    jeu_createur = models.CharField(max_length=100)
    date_emprunt = models.DateField(null=True, blank=True)
    disponible = models.BooleanField(default=True)
    emprunteur = models.ForeignKey(Emprunteur, null=True, blank=True, on_delete=models.CASCADE)

    def statut_emprunt(self):
        if self.disponible == 0:
            return 'hidden'
        else:
            return 'visible'

```

Liste des jeux de plateau

Jeu id	Nom du jeu	Date d'emprunt	Disponible	Emprunteur	Créateur du jeu	
1	Les colons de catane	Aug. 9, 2024	False	Paulo Coelho	Klaus Teuber	Retour
2	Jeu de l'oie	Aug. 1, 2024	False	Paulo Coelho	Inconnu	Retour
3	Jeu de dame	None	True	None	Inconnu	Emprunter

L'url affectée au lien renvoie à la fonction `emprunt_jeuplateau_page` dans le fichier `views.py`, ainsi que l'id du jeu.

```

from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('add_emprunteur/', views.add_emprunteur, name='add_emprunteur'),
    path('delete_emprunteur/<int:id>', views.delete_emprunteur, name='delete_emprunteur'),
    path('update_emprunteur/<int:id>', views.update_emprunteur, name='update_emprunteur'),
    path('add_media/', views.add_media, name='add_media'),
    path('add_jeuplateau/', views.add_jeuplateau, name='add_jeuplateau'),
    path('emprunt_jeuplateau_page/<int:pk>', views.emprunt_jeuplateau_page, name='emprunt_jeuplateau_page'),
]

```

Contrairement aux précédentes fonctions, celle-ci permet de créer un nouveau contexte pour afficher :

- Uniquement le jeu sélectionné en appelant son id
- Uniquement les emprunteurs non bloqués via la fonction filter() sur le champ bloqué à false

```

def emprunt_jeuplateau_page(request, pk):
    datas_emprunteur = Emprunteur.objects.all().filter(bloque = "False")
    jeuplateau_emprunte = Jeu_plateau.objects.get(id=pk)

    context = {
        'datas_emprunteur': datas_emprunteur,
        'jeuplateau_emprunte': jeuplateau_emprunte
    }

    return render(request, template_name='emprunt_jeuplateau.html', context)

```

Ensuite, elle permet d'ouvrir une nouvelle page qui appelle le fichier `emprunt_jeuplateau.html`.

Emprunter un jeu de plateau

3-Jeu de dame

Nom de l'emprunteur: jj/mm/aaaa

Dans ce nouveau fichier, j'ai créé un nouveau formulaire qui appelle l'url `emprunt_jeuplateau` et transmet l'id du jeu emprunté. Il permet à l'utilisateur de sélectionner une date d'emprunt (`date_emprunt`) et un emprunteur non bloqué (`emprunteur`). Ces paramètres serviront dans la fonction `emprunt_jeuplateau` qui enregistre les modifications via la fonction `save()`.

Créer un logiciel de gestion de médiathèque

The screenshot shows a code editor with three tabs: `urls.py`, `views.py`, and `emprunt_jeuplateau.html`. The `urls.py` tab contains the URL configuration for the application, including paths for home, adding/updating/deleting emprunteurs, adding media, adding games, viewing game loans, and returning games. The `views.py` tab contains the Python logic for these views, such as `add_emprunteur` and `emprunt_jeuplateau`. The `emprunt_jeuplateau.html` tab shows the HTML template for borrowing a game, which includes a form with fields for the borrower and the game.

L'utilisateur est ensuite redirigé vers la page `home.html`.

[Retourner](#)

Pour le retour d'un jeu de plateau, il faut remettre les champs `emprunteur` et `date_d'emprunt` à null, et passer son statut à disponible.

Le cheminement est le même que pour la modification du statut emprunteur.

Un bouton, accessible uniquement si le jeu sélectionné est non-disponible grâce à la fonction `retour_emprunt` du model `Jeu_plateau`, permet d'appeler la fonction `retour_jeuplateau` via l'url `retour_jeuplateau` et sin id. Celle-ci effectue et enregistre les modifications citées plus haut.

Créer un logiciel de gestion de médiathèque

```
<html lang="fr">
<body>
    <div class="row">
        <table style="width: 100%;border: solid black 2px">
            <tr>
                <td>{{jeu.id}}</td>
                <td>{{jeu.jeu_name}}</td>
                <td>{{jeu.date_emprunt}}</td>
                <td>{{jeu.disponible}}</td>
                <td>{{jeu.emprunteur}}</td>
                <td>{{jeu.jeu_createur}}</td>
            </td>
            <a href="{% url 'emprunt_jeuplateau_page' jeu.id%}">
                <button style="visibility: {{jeu.statut_emprunt2}}">Emprunter</button>
            </a>
        </td>
        <td>
            <form action="{% url 'retour_jeuplateau' jeu.id%}" method="POST" style="visibility: {{jeu.retour_emprunt2}}">
                {% csrf_token %}
                <input type="submit" value="Retour" class="btn">
            </form>
        </td>
    </tr>
</table>

```

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('add_emprunteur/', views.add_emprunteur, name='add_emprunteur'),
    path('delete_emprunteur/<int:id>', views.delete_emprunteur, name='delete_emprunteur'),
    path('update_emprunteur/<int:id>', views.update_emprunteur, name='update_emprunteur'),
    path('add_media/', views.add_media, name='add_media'),
    path('add_jeuplateau/', views.add_jeuplateau, name='add_jeuplateau'),
    path('emprunt_jeuplateau_page/<int:pk>/', views.emprunt_jeuplateau_page, name='emprunt_jeuplateau_page'),
    path('emprunt_jeuplateau/<int:id>', views.emprunt_jeuplateau, name='emprunt_jeuplateau'),
    path('retour_jeuplateau/<int:id>', views.retour_jeuplateau, name='retour_jeuplateau'),
    path('emprunt_livre_page/', views.emprunt_livre_page, name='emprunt_livre_page'),
    path('retour_livre_page/<int:id>', views.retour_livre_page, name='retour_livre_page')
]
```

```
def retour_jeuplateau(request,id):
    if request.method == 'POST':
        pi = jeu_plateau.objects.get(pk=id)
        pi.disponible = True
        pi.date_emprunt = None
        pi.emprunteur = None
        pi.save()
    return HttpResponseRedirect('/')
```

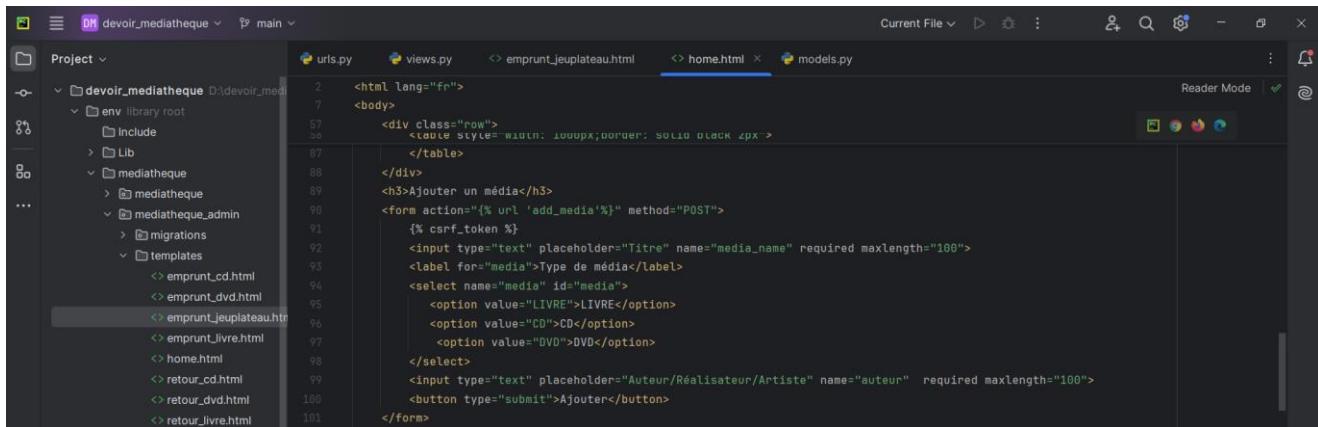
Ajouter/emprunter/retourner un média

Ajouter

Concernant la création d'un média, j'ai créé un formulaire dans le fichier home.html qui permet :

- De saisir le nom du média (media_name)
- De sélectionner le type de média via une liste de choix identique à celle du champ type_media du model Media
- De saisir l'auteur, réalisateur ou artiste du média

Créer un logiciel de gestion de médiathèque

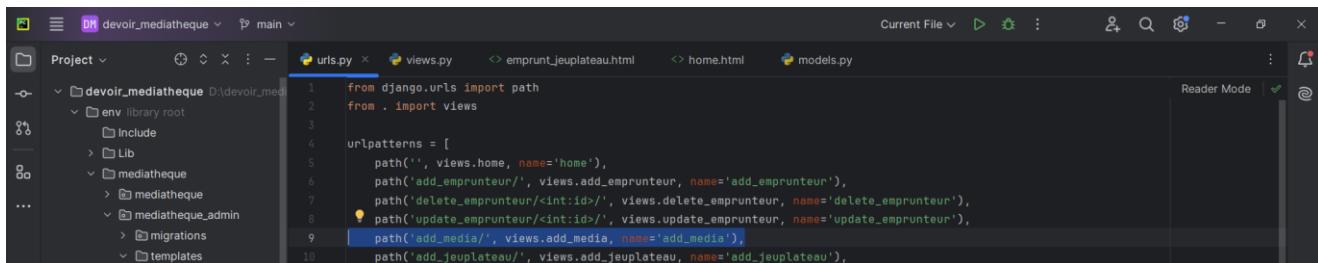


```
<html lang="fr">
    <body>
        <div class="row">
            <table style="width: 100px; border: solid black 1px;">
                </table>
        </div>
        <h3>Ajouter un média</h3>
        <form action="{% url 'add_media'%}" method="POST">
            {% csrf_token %}
            <input type="text" placeholder="Titre" name="media_name" required maxlength="100">
            <label for="media">Type de média</label>
            <select name="media" id="media">
                <option value="LIVRE">LIVRE</option>
                <option value="CD">CD</option>
                <option value="DVD">DVD</option>
            </select>
            <input type="text" placeholder="Auteur/Réalisateur/Artiste" name="auteur" required maxlength="100">
            <button type="submit">Ajouter</button>
        </form>
```

Ajouter un média

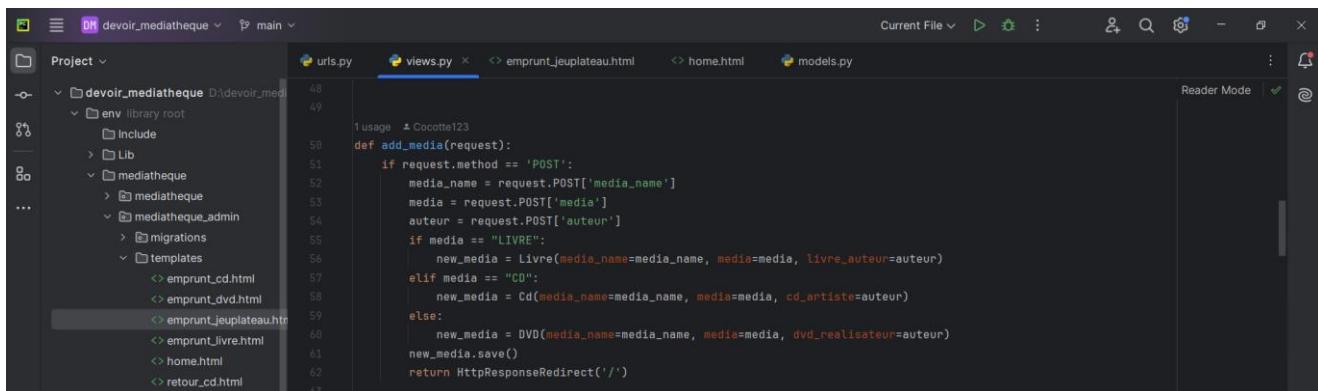
Titre Type de média LIVRE Auteur/Réalisateur/Artiste Ajouter

Ce formulaire appelle l'url add_media et, de là, la fonction add_media.



```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('add_emprunteur/', views.add_emprunteur, name='add_emprunteur'),
    path('delete_emprunteur/<int:id>', views.delete_emprunteur, name='delete_emprunteur'),
    path('update_emprunteur/<int:id>', views.update_emprunteur, name='update_emprunteur'),
    path('add_media/', views.add_media, name='add_media'),
    path('add_jeuplateau/', views.add_jeuplateau, name='add_jeuplateau'),
```



```
def add_media(request):
    if request.method == 'POST':
        media_name = request.POST['media_name']
        media = request.POST['media']
        auteur = request.POST['auteur']
        if media == "LIVRE":
            new_media = Livre(media_name=media_name, media=media, livre_auteur=auteur)
        elif media == "CD":
            new_media = Cd(media_name=media_name, media=media, cd_artiste=auteur)
        else:
            new_media = DVD(media_name=media_name, media=media, dvd_realisateur=auteur)
        new_media.save()
    return HttpResponseRedirect('/')
```

Etant donné qu'il existe trois tables différentes selon le type de média (Livre, Cd ou Dvd), la fonction teste le champ media. Le contenu de celui-ci détermine où sera enregistré la saisie.

Modifier

Le process d'emprunt d'un média est le même pour chaque type. Il existe trois boutons car les tables appelées sont différentes.

Emprunter un média

Emprunter un livre Emprunter un CD Emprunter un DVD

Je présente ici uniquement le schéma pour l'emprunt d'un livre.

Le bouton « Emprunter un livre » appelle une l'url emprunt_livre_page qui permet d'accéder à la fonction emprunt_livre_page.

Créer un logiciel de gestion de médiathèque

```
<html lang="fr">
    <body>
        <h3>Emprunter un média</h3>
        <div class="row">
            <a href="{% url 'emprunt_livre_page'%}">
                <button>Emprunter un livre</button>
            </a>
        </div>
    </body>
</html>
```

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('add_emprunteur/', views.add_emprunteur, name='add_emprunteur'),
    path('delete_emprunteur<int:id>', views.delete_emprunteur, name='delete_emprunteur'),
    path('update_emprunteur<int:id>', views.update_emprunteur, name='update_emprunteur'),
    path('add_media/', views.add_media, name='add_media'),
    path('add_jeuplateau/', views.add_jeuplateau, name='add_jeuplateau'),
    path('emprunt_jeuplateau_page<int:pk>', views.emprunt_jeuplateau_page, name='emprunt_jeuplateau_page'),
    path('emprunt_jeuplateau/<int:id>', views.emprunt_jeuplateau, name='emprunt_jeuplateau'),
    path('retour_jeuplateau/<int:id>', views.retour_jeuplateau, name='retour_jeuplateau'),
    path('emprunt_livre_page/', views.emprunt_livre_page, name='emprunt_livre_page'),
    path('emprunt_livre/', views.emprunt_livre, name='emprunt_livre'),
]
```

Celle-ci crée un contexte qui permet de filtrer les emprunteurs non bloqués et les livres disponibles et ouvre un nouveau fichier : emprunt_livre.html.

```
def emprunt_livre_page(request):
    datas_emprunteur = Emprunteur.objects.all().filter(bloque = "False")
    datas_livre = Livre.objects.all().filter(disponible = "True")

    context = {
        'datas_emprunteur': datas_emprunteur,
        'datas_livre': datas_livre
    }

    return render(request, template_name='emprunt_livre.html', context)
```

Emprunter un livre

Nom de l'emprunteur [1 - Emile Zola] Titre du livre [2 - Harry Potter et la chambre des secrets] jj/mm/aaaa Valider

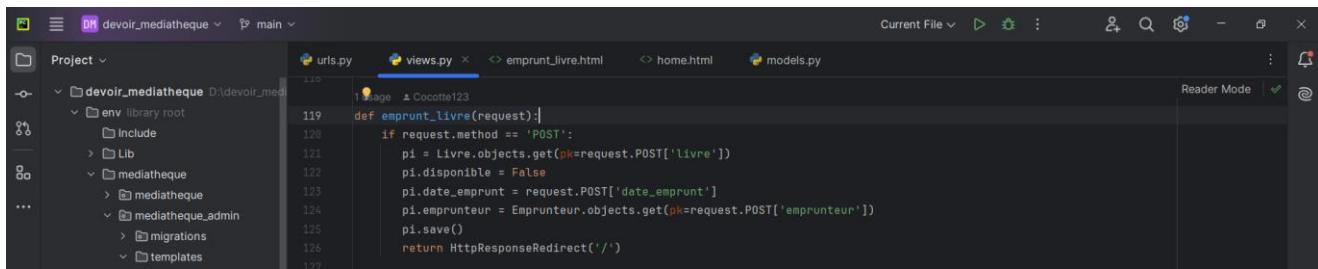
Dans cette nouvelle page, s'affiche un formulaire qui permet à l'utilisateur de sélectionner un emprunteur issu du context datas_emprunteur, un livre issu du context datas_livre et une date d'emprunt.

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="UTF-8">
        <title>Projet Python Médiathèque</title>
    </head>
    <body>
        <h3>Emprunter un livre</h3>

        <form action="{% url 'emprunt_livre'%}" method="POST">
            {% csrf_token %}
            <label for="emprunteur">Nom de l'emprunteur</label>
            <select name="emprunteur" id="emprunteur" required>
                {% for emprunteur in datas_emprunteur %}
                    <option value="{{emprunteur.id}}>{{emprunteur.id}} - {{emprunteur.first_name}} {{emprunteur.last_name}}</option>
                {% endfor %}
            </select>
            <label for="livre">Titre du livre</label>
            <select name="livre" id="livre" required>
                {% for livre in datas_livre %}
                    <option value="{{livre.id}}>{{livre.id}} - {{livre.media_name}}</option>
                {% endfor %}
            </select>
            <input type="date" name="date_emprunt">
            <input type="submit" value="Valider" class="btn">
        </form>
    </body>
</html>
```

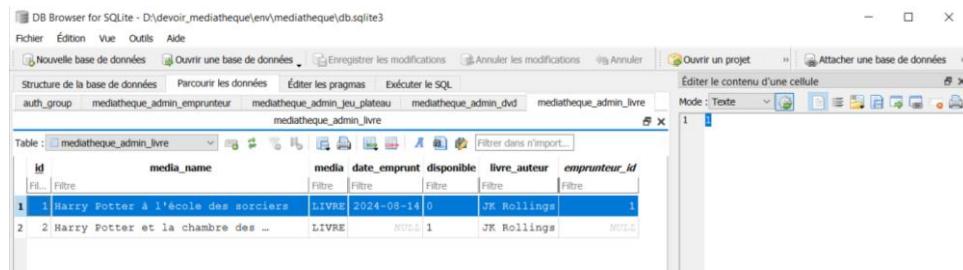
Créer un logiciel de gestion de médiathèque

Le formulaire appelle l'url emprunt_livre qui appelle la fonction emprunt_livre.



```
def emprunt_livre(request):
    if request.method == 'POST':
        pi = Livre.objects.get(pk=request.POST['livre'])
        pi.disponible = False
        pi.date_emprunt = request.POST['date_emprunt']
        pi.emprunteur = Emprunteur.objects.get(pk=request.POST['emprunteur'])
        pi.save()
    return HttpResponseRedirect('/')
```

Ici, on sélectionne le livre à modifier via la fonction get() et à l'id récupéré du formulaire. Le champ disponible passe à false. Le champ date_emprunt est complété la date indiquée. Le champ emprunteur est rempli par l'id de l'emprunteur choisi.



id	media_name	media	date_emprunt	disponible	livre_auteur	emprunteur_id
1	Harry Potter à l'école des sorciers	LIVRE	2024-08-14	0	JK Rollings	1
2	Harry Potter et la chambre des	LIVRE	NULL	1	JK Rollings	NULL

A la fin de la fonction, l'utilisateur est redirigé vers la page home.html.

[Retourner](#)

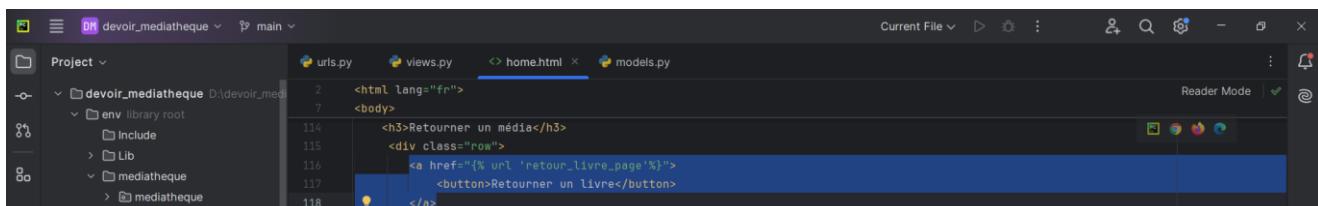
Le processus de retour d'un média est le même pour chaque type. Il existe trois boutons car les tables appelées sont différentes.

[Retourner un média](#)

[Retourner un livre](#) [Retourner un CD](#) [Retourner un DVD](#)

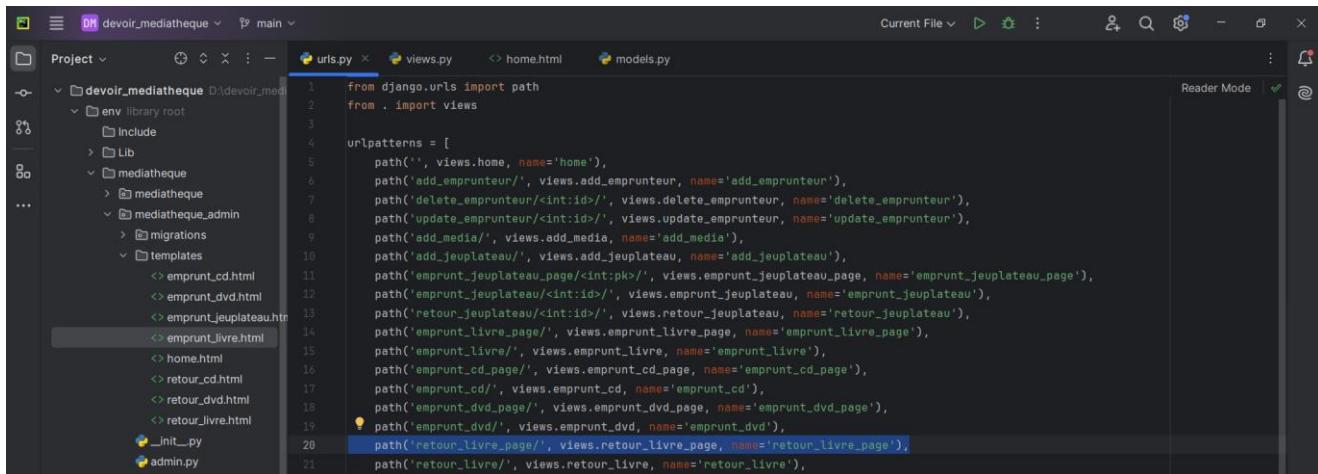
Je présente ici uniquement le schéma pour le retour d'un livre.

Le bouton « Retourner un livre » appelle une l'url retour_livre_page qui permet d'accéder à la fonction retour_livre_page.



```
<html lang="fr">
<body>
    <h3>Retourner un média</h3>
    <div class="row">
        <a href="{% url 'retour_livre_page'%}">
            <button>Retourner un livre</button>
        </a>
```

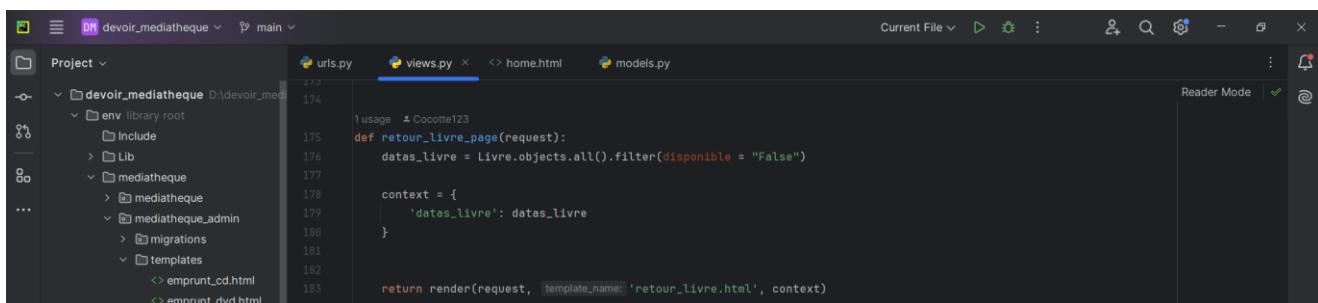
Créer un logiciel de gestion de médiathèque



```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('add_emprunteur/', views.add_emprunteur, name='add_emprunteur'),
    path('delete_emprunteur/<int:id>', views.delete_emprunteur, name='delete_emprunteur'),
    path('update_emprunteur/<int:id>', views.update_emprunteur, name='update_emprunteur'),
    path('add_media/', views.add_media, name='add_media'),
    path('add_jeuplateau/', views.add_jeuplateau, name='add_jeuplateau'),
    path('emprunt_jeuplateau_page/<int:pk>', views.emprunt_jeuplateau_page, name='emprunt_jeuplateau_page'),
    path('emprunt_jeuplateau/<int:id>', views.emprunt_jeuplateau, name='emprunt_jeuplateau'),
    path('retour_jeuplateau/<int:id>', views.retour_jeuplateau, name='retour_jeuplateau'),
    path('emprunt_livre_page/', views.emprunt_livre_page, name='emprunt_livre_page'),
    path('emprunt_livre/', views.emprunt_livre, name='emprunt_livre'),
    path('emprunt_cd_page/', views.emprunt_cd_page, name='emprunt_cd_page'),
    path('emprunt_cd/', views.emprunt_cd, name='emprunt_cd'),
    path('emprunt_dvd_page/', views.emprunt_dvd_page, name='emprunt_dvd_page'),
    path('emprunt_dvd/', views.emprunt_dvd, name='emprunt_dvd'),
    path('retour_livre_page/', views.retour_livre_page, name='retour_livre_page'),
    path('retour_livre/', views.retour_livre, name='retour_livre'),
]
```

Celle-ci crée un contexte qui permet de filtrer les livres non disponibles et ouvre un nouveau fichier : retour_livre_page.html.



```
def retour_livre_page(request):
    datas_livre = Livre.objects.all().filter(disponible = "False")

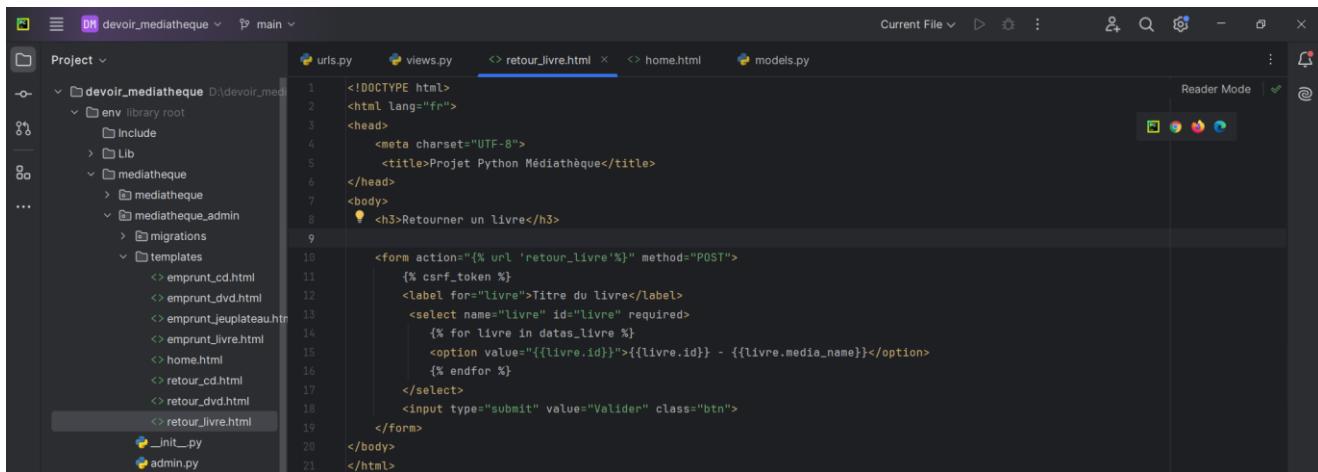
    context = {
        'datas_livre': datas_livre
    }

    return render(request, template_name= 'retour_livre.html', context)
```

Retourner un livre

Titre du livre Valider
1 - Harry Potter à l'école des sorciers

Dans cette nouvelle page, s'affiche un formulaire qui permet à l'utilisateur de sélectionner le livre à retourner issu du context datas_livre.

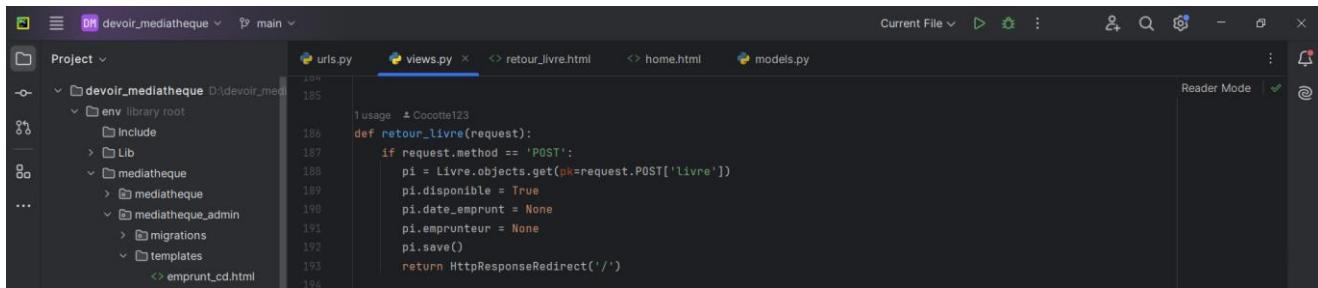


```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Projet Python Médiathèque</title>
</head>
<body>
    <h3>Retourner un livre</h3>

    <form action="{% url 'retour_livre'%}" method="POST">
        {% csrf_token %}
        <label for="livre">Titre du livre</label>
        <select name="livre" id="livre" required>
            {% for livre in datas_livre %}
                <option value="{{livre.id}}>{{livre.id}} - {{livre.media_name}}</option>
            {% endfor %}
        </select>
        <input type="submit" value="Valider" class="btn">
    </form>
</body>
</html>
```

Le formulaire appelle l'url retour_livre qui appelle la fonction retour_livre.

Créer un logiciel de gestion de médiathèque



```
1 usage  Cocotte123
2
3 def retour_livre(request):
4     if request.method == 'POST':
5         pi = Livre.objects.get(pk=request.POST['livre'])
6         pi.disponible = True
7         pi.date_emprunt = None
8         pi.emprunteur = None
9         pi.save()
10        return HttpResponseRedirect('/')
```

Ici, on sélectionne le livre à retourner via la fonction `get()` et à l'id récupéré du formulaire. Le champ `disponible` passe à true. Le champ `date_emprunt` et le champ `emprunteur` sont passés à null.

A la fin de la fonction, l'utilisateur est redirigé vers la page `home.html`.

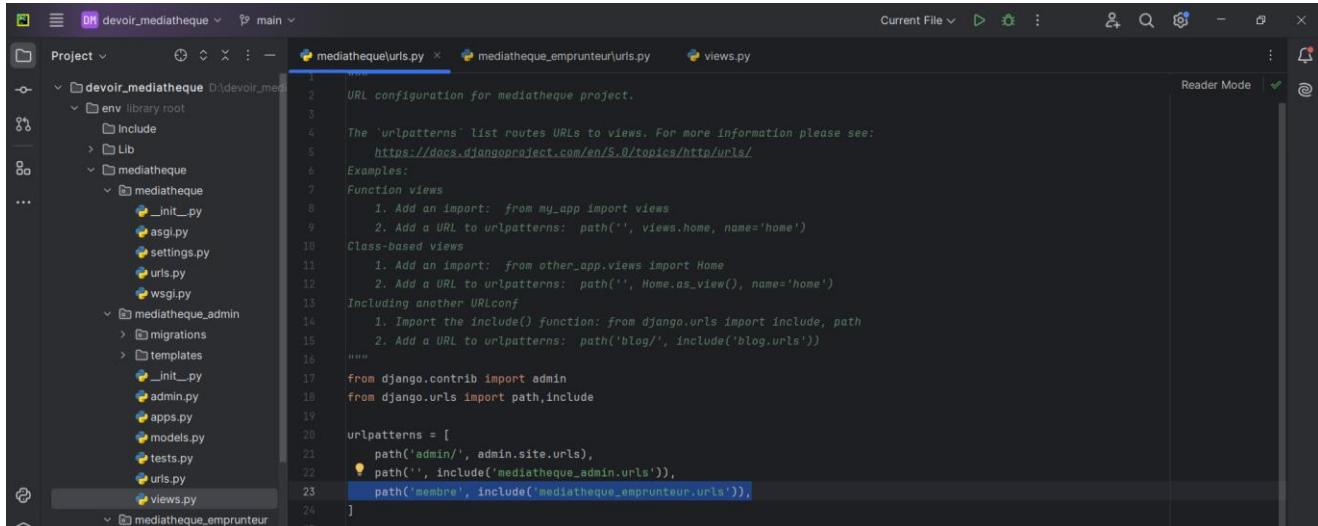
Tout au long de ces points, nous avons vu les `models` et les `views` de l'application `mediatheque_admin`. Nous allons voir maintenant voir ceux liés à l'application `mediatheque_emprunteur`.

L'application « mediatheque_emprunteur »

Views « mediatheque_emprunteur »

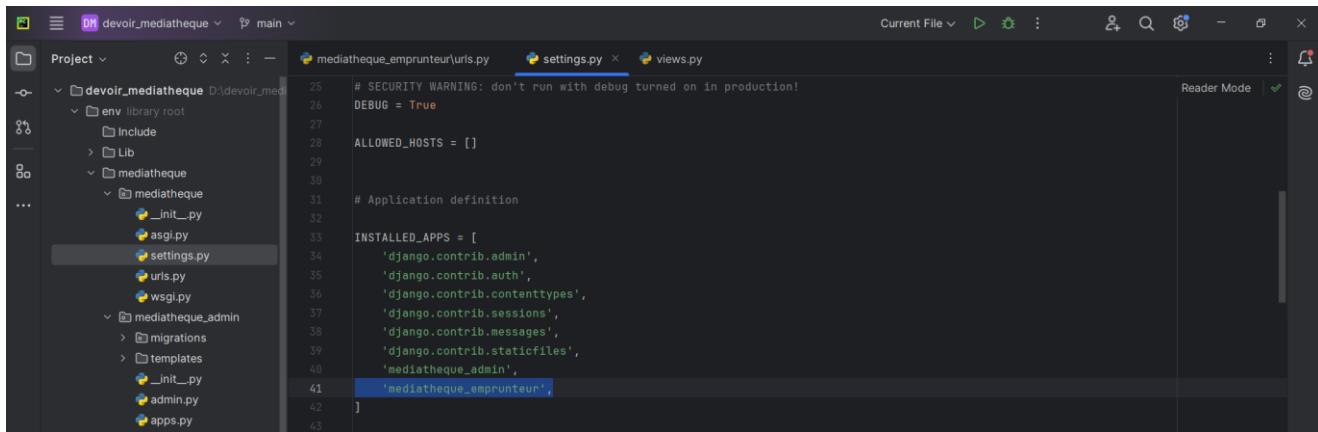
A l'image de l'application « `mediatheque_admin` », il est nécessaire de réaliser certains paramètres effectués au niveau du projet « `mediatheque` » :

1. On ajoute une url dans le fichier `urls.py` du projet « `mediatheque` » qui permet d'appeler « `include` » toutes les urls de l'application « `mediatheque_admin` »
2. On ajoute l'application dans le fichier `settings.py`



```
1 # URL configuration for mediatheque project.
2
3 The 'urlpatterns' list routes URLs to views. For more information please see:
4     https://docs.djangoproject.com/en/3.0/topics/http/urls/
5 Examples:
6 Function views
7     1. Add an import: from my_app import views
8     2. Add a URL to urlpatterns: path('', views.home, name='home')
9 Class-based views
10    1. Add an import: from other_app.views import Home
11    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
12 Including another URLconf
13    1. Import the include() function: from django.urls import include, path
14    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15
16 from django.contrib import admin
17 from django.urls import path, include
18
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path('', include('mediatheque_admin.urls')),
23     path('membre', include('mediatheque_emprunteur.urls')),
24 ]
```

Créer un logiciel de gestion de médiathèque



```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

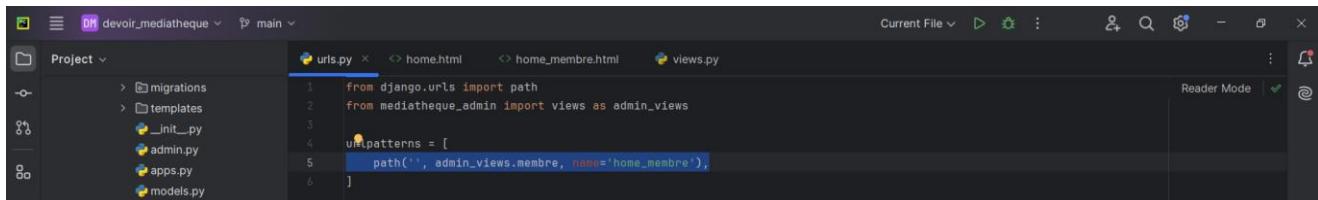
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'mediatheque_admin',
    'mediatheque_emprunteur',
]
```

J'ai également créé un fichier home_membre.html dans le dossier.

Affichage des listes emprunteurs, médias et jeux de plateau

Même si la base de données se trouve au niveau du projet et pas au niveau des applications, la création des tables se fait via les models. Or les différents models utilisés dans cette application sont les mêmes que ceux de l'application mediatheque_admin.

Par conséquent, la fonction qui permet d'afficher les médias et les jeux de plateau disponibles pour l'emprunteur est créé dans le fichier views de l'application mediatheque_admin. Il est alors nécessaire d'importer ce fichier au niveau du fichier urls de l'application mediatheque_emprunteur.

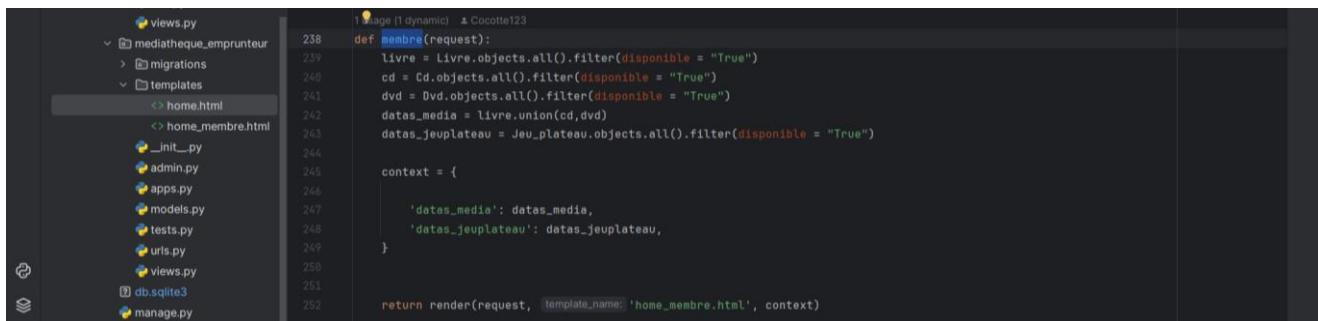


```
from django.urls import path
from mediatheque_admin import views as admin_views

urlpatterns = [
    path('', admin_views.membre, name='home_membre'),
]
```

Etant donné qu'il s'agit uniquement de l'affichage de données, la fonction membre permet de créer un context qui sélectionne :

- Les jeux de plateau disponibles avec filter() et le champ disponible à true
- Les trois types de média disponibles avec filter() et le champ disponible à true unis une nouvelle fois par la fonction union().



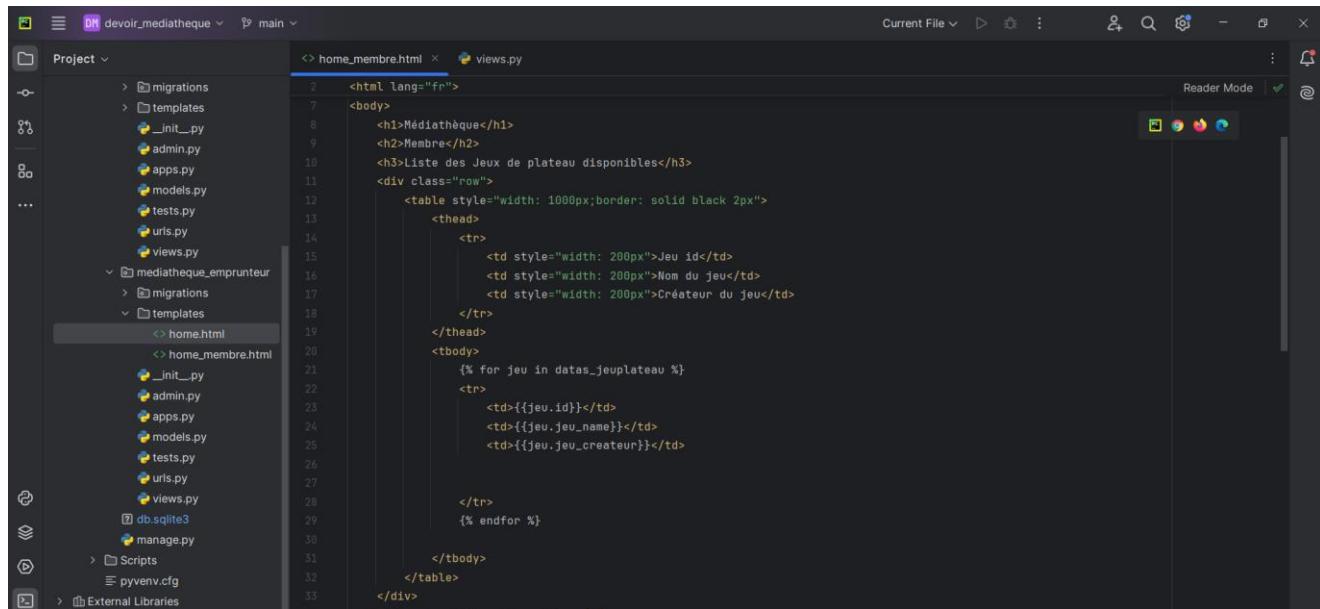
```
def membre(request):
    livre = Livre.objects.all().filter(disponible = "True")
    cd = Cd.objects.all().filter(disponible = "True")
    dvd = Dvd.objects.all().filter(disponible = "True")
    datas_media = livre.union(cd,dvd)
    datas_jeuplateau = Jeu_plateau.objects.all().filter(disponible = "True")

    context = {
        'datas_media': datas_media,
        'datas_jeuplateau': datas_jeuplateau,
    }

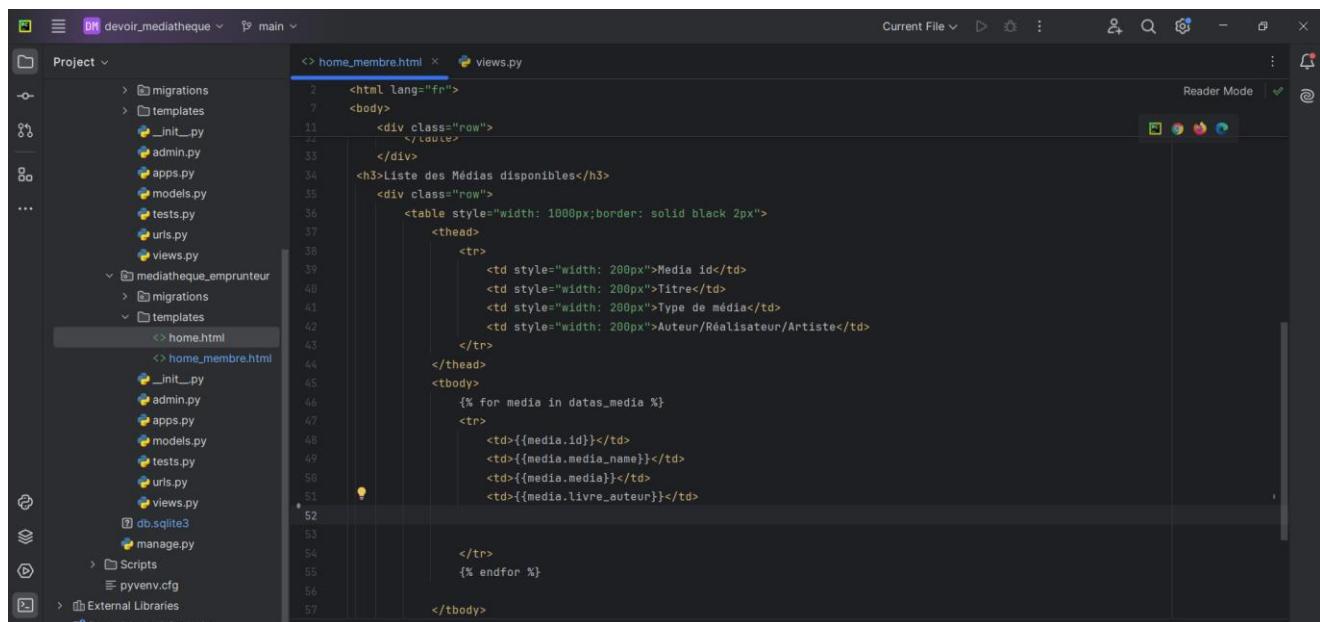
    return render(request, template_name= 'home_membre.html', context)
```

Les données sont appelées dans le fichier home_membre.html via des boucles d'affichage.

Créer un logiciel de gestion de médiathèque



```
<html lang="fr">
    <body>
        <h1>Médiathèque</h1>
        <h2>Membre</h2>
        <h3>Liste des Jeux de plateau disponibles</h3>
        <div class="row">
            <table style="width: 1000px; border: solid black 2px">
                <thead>
                    <tr>
                        <td style="width: 200px">Jeu id</td>
                        <td style="width: 200px">Nom du jeu</td>
                        <td style="width: 200px">Créateur du jeu</td>
                    </tr>
                </thead>
                <tbody>
                    {% for jeu in datas_jeuplateau %}
                    <tr>
                        <td>{{jeu.id}}</td>
                        <td>{{jeu.jeu_name}}</td>
                        <td>{{jeu.jeu_createur}}</td>
                    </tr>
                    {% endfor %}
                </tbody>
            </table>
        </div>
```



```
<html lang="fr">
    <body>
        <div class="row">
            <div>
                <h3>Liste des Médias disponibles</h3>
                <div class="row">
                    <table style="width: 1000px; border: solid black 2px">
                        <thead>
                            <tr>
                                <td style="width: 200px">Media id</td>
                                <td style="width: 200px">Titre</td>
                                <td style="width: 200px">Type de média</td>
                                <td style="width: 200px">Auteur/Réalisateur/Artiste</td>
                            </tr>
                        </thead>
                        <tbody>
                            {% for media in datas_media %}
                            <tr>
                                <td>{{media.id}}</td>
                                <td>{{media.media_name}}</td>
                                <td>{{media.media_type}}</td>
                                <td>{{media.livre_auteur}}</td>
                            </tr>
                            {% endfor %}
                        </tbody>
                    </table>
                </div>
            </div>
        </div>
```

Médiathèque

Membre

Liste des Jeux de plateau disponibles

Jeu id	Nom du jeu	Créateur du jeu
2	Jeu de l'oise	Inconnu
3	Jeu de dame	Inconnu

Liste des Médias disponibles

Media id	Titre	Type de média	Auteur/Réalisateur/Artiste
1	La cinquième vague	CD	Adèle
1	Le cinquième élément	DVD	Luc Besson
2	Harry Potter et la chambre des secrets	LIVRE	JK Rollings
2	Un autre monde	CD	Téléphon

Conclusion

[Créer un logiciel de gestion de médiathèque](#)

A l'issue de l'étude des différents points du projet, on peut considérer que le logiciel répond aux problématiques formulées par le client en termes de fonctionnalités.

Dans un second temps, la partie front-end sera développée pour compléter la réponse aux besoins..