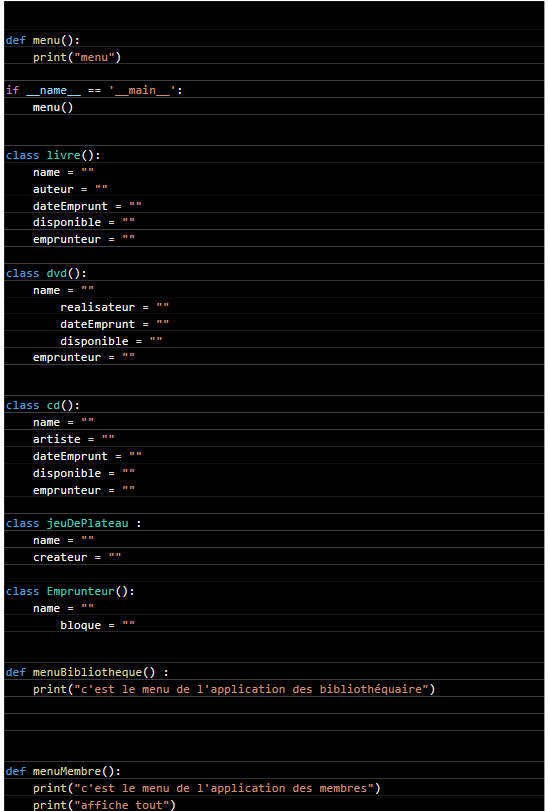
**Présentation et modification du code fourni**

Le code fourni est composé de plusieurs classes dans un même fichier comme suit :



Ces classes représentent une base pour les modèles à utiliser pour le projet mais il se trouve, en plus que leurs données sont vides, que certaines d’entre elles se répète dans les données qu’elles appellent, tel que livre, dvd et cd.

Pour pallier à ce problème, un modèle parent (Media) a été créé pour regrouper les données en commun entre Livre, Dvd et Cd. Ce modèle Media est un modèle abstrait et n’apparait donc pas en base de données.

Un autre problème est qu’aucun modèle n’utilise “models.Model” comme classe parente. Ce qui rend plus compliqué la création, modification, ou suppression de données en base de données.

Pour ces raisons, les model ont reçu comme classe parente “models.Model”, en réalisant un import de “models” à partir de “django.db”.

Exemple :

from django.db import models  
  
  
class Emprunteur(models.Model):

Grâce à cela on peut également attribuer à leurs variables des options et types de champs, des clés primaires, des clés étrangères.

Plus précisément :

Charfield renvoi un champ de caractère ayant un nombre de caractères max égal à 150, IntegerField un nombre ou encore BooleanField un booléen (true or false) avec par défaut la valeur faux.

Ensuite, les modèles ont été séparé par famille, dans différents dossiers, contenant chacun un fichier models.py. On retrouve ainsi les dossiers suivants :

* Membre : avec le model Emprunteur
* Media : avec les modèles Livre, Dvd, Cd, Jeu de Plateau
* Emprunt : avec de nouveaux modèles créés pour lier les modèles Emprunteur et les Médias

Le dossier Emprunt contient les modèles EmpruntLivre, EmpruntCd et EmpruntDvd. Ces 3 modèles ont variables la date d’emprunt qui était initialement dans les modèles Medias, l’id des Media et l’id d’Emprunteur. Pour permettre cette liaison un champ “membres” a été ajouté aux modèles Medias, avec un type de champ “ManyToManyField.

**La mise en place des fonctionnalités demandées**

*Créer un membre-emprunteur :*

def ajoutmembre(request):  
 if request.method == 'POST':  
 creationmembre = CreationMembre(request.POST)  
 if creationmembre.is\_valid():  
 membre = Emprunteur()  
 membre.name = creationmembre.cleaned\_data['name']  
 membre.save()  
 membres = Emprunteur.objects.all()  
 return render(request, 'Membres/list.html',  
 {'membres': membres})  
 else:  
 creationmembre = CreationMembre()  
 return render(request,  
 'Membres/ajoutMembre.html',  
 {'creationmembre': creationmembre}  
 )

Si la méthode “post” du formulaire présent dans le template ajoutMembre.html est appelé, alors on applique les choses suivantes :

* On appelle le formulaire créé dans notre forms.py à travers la variable creationmembre
* Si le formulaire est valide :

- On appelle le model Emprunteur à travers la variable membre

- On attribue le champ de formulaire name (de creationmembre) au nom de l’Emprunteur

- On sauvegarde la valeur du champ en base de données

On affiche ensuite la page de liste des membres.

Si la méthode post n’est pas appelé alors on affiche la page ajoutMembre.html avec notre champ de formulaire.

*Afficher la liste des membres :*

def listemembre(request):  
 membres = Emprunteur.objects.all()  
 return render(request, 'Membres/list.html',  
 {'membres': membres})

Pour créer une liste des membres, on crée une variable membres, avec pour valeur tous les objets du model Emprunteur. La fonction retourne ici le template list.html dans le dossier Membre et y affiche tous les membres.

*Mettre à jour un membre :*

def updatemembre(request, id):  
 if request.method == 'POST':  
 membre = Emprunteur.objects.get(pk=id)  
 update\_membre = UpdateMembre(request.POST)  
 if update\_membre.is\_valid():  
 membre.name = update\_membre.cleaned\_data['name']  
 membre.save()  
 membres = Emprunteur.objects.all()  
 return render(request, 'Membres/list.html',  
 {'membres': membres})  
 else:  
 update\_membre = UpdateMembre()  
 return render(request,  
 'Membres/updateMembre.html',  
 {'updatemembre': update\_membre}  
 )

Ici, comme pour l’ajout d’un membre, si une méthode post est appelé alors on attribue la valeur name de notre model Emprunteur à notre champ de formulaire ayant pour valeur ‘name’ et on sauvegarde en base de données. À la différence que l’Emprunteur est déjà existant en base de données. Il est appelé à travers son id. En effet, en mettant un id en fin d’url alors on peut retrouver les objets déjà existant en base de données et ainsi les modifier ou les supprimer.

*Afficher la liste des médias :*

def listemedia(request):  
 livres = Livre.objects.all()  
 dvds = Dvd.objects.all()  
 cds = Cd.objects.all()  
 jdps = Jdp.objects.all()  
 return render(request, 'Media/list.html',  
 {'livres': livres, 'dvds' : dvds, 'cds' : cds, 'jdps' : jdps})

La liste des médias fonctionne comme l’affichage de la liste des membres. On attribue à une variable, respectivement, tous les objets des model “média”, puis on les affiche à travers un template html.

*Créer un emprunt pour un média disponible :*

def empruntmedia(model, model2, related\_models, request, \_id):  
 *"""*  
 *Méthode de base pour les emprunts.*  
 *"""*  
if request.method == 'POST':  
 media = model.objects.get(pk=\_id)  
 creationemprunt = CreationEmprunt(request.POST)  
 if creationemprunt.is\_valid():  
 value = creationemprunt.cleaned\_data['membre']  
 membre = Emprunteur.objects.get(name=value)  
 if media.disponible and membre.NbEmprunt <= 2 and not membre.bloque:  
 membre.NbEmprunt += 1  
 membre.save()  
 media.disponible = False  
 media.save()  
 if related\_models == 'livre':  
 membre.livres.add(media)  
 elif related\_models == 'cd':  
 membre.cds.add(media)  
 elif related\_models == 'dvds':  
 membre.dvds.add(media)  
 else:  
 membre.bloque = True  
 membre.save()  
 return render(request, 'Emprunt/Erreur/Error\_livre.html')  
 emprunteurs = model2.objects.all()  
 return render(request, 'Emprunt/Emprunt/listEmprunt.html',  
 {'emprunteurs': emprunteurs})  
 else:  
 creationemprunt = CreationEmprunt()  
 return render(request,  
 'Emprunt/Emprunt/Empruntlivre.html',  
 {'creationemprunt': creationemprunt}  
 )

Ici, si la méthode utilisée n’est pas un post alors cela affichera la page html Emprunlivre.html. Mais si ça en est un, alors :

* On récupère la PK du media grâce à l’url
* On récupère notre objet membre grâce au champ de saisie de notre formulaire.

Puis SI le media dont l’id est égal à l’id dans notre url est disponible, MAIS également si le membre appelé à l’aide de notre champ de formulaire n’a pas plus de 3 emprunts, ALORS :

* On met le média en indisponible et on le sauvegarde.
* On incrémente son nombre d’emprunt de 1 et on le sauvegarde.
* De plus en fonction de la valeur du paramètre related\_models alors la jointure sera appliquée sur tel ou tel modèle.

SINON on bloque le membre, on sauvegarde et on affiche une page html d’erreur.

*Ajouter un media :*

L’ajout d’un média se fait de la même façon que l’ajout d’un membre mais en utilisant les modèles Livre, Cd, Dvd et Jeu de Plateau.

*Rentrer un emprunt :*

def retourmedia(model, model2, related\_models, request, \_id):  
 *""" Méthode permettant défaire une jointure existante entre un emprunteur et un livre."""*  
if request.method == 'POST':  
 emprunt = model2.objects.get(pk=\_id)  
 retouremprunt = RetourEmprunt(request.POST)  
 if retouremprunt.is\_valid():  
 value1 = retouremprunt.cleaned\_data['membre']  
 membre = Emprunteur.objects.get(name=value1)  
 value2 = retouremprunt.cleaned\_data['media']  
 media = model.objects.get(name=value2)  
 if not media.disponible:  
 membre.NbEmprunt -= 1  
 if membre.bloque:  
 membre.bloque = False  
 membre.save()  
 media.disponible = True  
 media.save()  
 if date.today() > emprunt.dateEmprunt\_max:  
 membre.bloque = True  
 membre.save()  
 if related\_models == 'livre':  
 membre.livres.remove(media)  
 elif related\_models == 'cd':  
 membre.cds.remove(media)  
 elif related\_models == 'dvds':  
 membre.dvds.remove(media)  
 emprunteurs = model2.objects.all()  
 return render(request, 'Emprunt/Emprunt/listEmprunt.html',  
 {'emprunteurs': emprunteurs})  
 else:  
 retouremprunt = RetourEmprunt()  
 return render(request,  
 'Emprunt/Retour/retourlivre.html',  
 {'retouremprunt': retouremprunt}  
 )

Ici, si la méthode utilisée n’est pas un post alors cela affichera la page html retourlivre.html. Mais si ça en est un, alors :

* On récupère la PK de la table intermédiaire d'Emprunteur et les médias grâce à l’url
* On appelle le formulaire créé dans notre forms.py à travers la variable retouremprunt
* Si le formulaire est valide on appelle un membre et media à l’aide des champs de formulaire.
* Si le média appelé est indisponible, le membre appelé voit son nombre emprunt réduit de 1. De plus, si ce membre est bloqué, on le débloque et on sauvegarde.
* Le média appelé devient ensuite disponible et est sauvegardé.
* Si la date du jour dépasse la date max d’emprunt alors le membre est bloqué.

**Accéder au projet depuis votre machine**

1. **Ouvrir PyCharm** : Lancez PyCharm.
2. **Cloner le Dépôt GitHub** :
   * **Accédez à File > New > Project from Version Control**.
   * Sélectionnez Git dans la liste des systèmes de contrôle de version.
   * Dans le champ URL, entrez l'URL du dépôt GitHub (par exemple, <https://github.com/username/repository.git>).
   * Choisissez un répertoire de destination sur votre machine où le projet sera cloné.
   * Cliquez sur Clone pour cloner le dépôt.
3. **Créer un Environnement Virtuel** :
   * **Ouvrir le Projet Cloné** : Une fois le dépôt cloné, PyCharm ouvrira automatiquement le projet. Si ce n’est pas le cas, allez dans File > Open et ouvrez le répertoire cloné.
   * **Configurer l’Environnement Virtuel** :
     + **Accédez à File > Settings (ou PyCharm > Preferences sur macOS)**.
     + Dans la section Project: <project\_name>, sélectionnez Python Interpreter.
     + Cliquez sur l'icône de roue dentée (⚙️) et sélectionnez Add....
     + Choisissez Virtualenv Environment.
     + Sélectionnez New environment et choisissez un emplacement pour votre environnement virtuel. Vous pouvez également choisir de créer l'environnement dans le répertoire du projet.
     + Cliquez sur OK pour créer l'environnement virtuel.

**Stratégie de test**

Pour vérifier que le CRUD est bien respecté avec notre modèle principal qui est Emprunteur, on vérifie à l’aide de tests que les points suivants fonctionnent correctement : la récupération, création, la modification, la suppression des données.

Pour les modèles Media on procède de la même façon, on test si on peut créer des objets à partir de nos modèles puis on vérifie si, d’abord il existe, puis si on peut les récupérer (pour pouvoir les afficher dans une liste).

Enfin pour les modèles intermédiaires, les tests ont surtout consisté à vérifier si la liaison entre les modèles Emprunteur et les Media se faisaient comme convenu.

**Base de données**

La base de données est accessible dans le repository github, qui est le suivant :

https://github.com/Dada-nol/Django-project.git