

Rapport de Projet : Gestion de Médiathèque

1. Étude et Correctifs du Code Fourni

1.1 Analyse du Code Existant

Le code fourni était une implémentation en Python pur avec une structure rudimentaire .

- **Classes et Attributs** : Les classes `livre`, `dvd`, `cd`, `jeuDePlateau`, et `Emprunteur` étaient définies sans constructeur (`__init__`), ce qui rendait leur utilisation et leur manipulation difficiles. Les attributs étaient définis directement dans la classe, ce qui est contraire aux bonnes pratiques de programmation orientée objet.
- **Indentation et Structure** : Le code manquait d'une indentation correcte, ce qui affectait sa lisibilité et sa maintenabilité.
- **Fonctionnalités** : Les fonctions de menu étaient rudimentaires et n'implémentaient pas les fonctionnalités nécessaires pour une application de gestion de médiathèque complète.

1.2 Correctifs Apportés

1. **Refactorisation des Classes** :
 - Chaque classe a été refactorisée pour utiliser des constructeurs (`__init__`) pour initialiser les attributs.
 - Les attributs de chaque classe sont désormais correctement encapsulés et sont définis à l'intérieur du constructeur.
2. **Correction de l'Indentation** :
 - Le code a été réindenté pour améliorer la lisibilité et la structure.

Version améliorée du code:

```
class Livre:
    def __init__(self, name, auteur, date_emprunt="",
disponible=True, emprunteur=None):
        self.name = name
        self.auteur = auteur
        self.date_emprunt = date_emprunt
        self.disponible = disponible
        self.emprunteur = emprunteur

    def afficher_details(self):
        return f"Livre: {self.name}, Auteur: {self.auteur},
Disponible: {self.disponible}"
```

```

class Dvd:
    def __init__(self, name, réalisateur, date_emprunt="",
disponible=True, emprunteur=None):
        self.name = name
        self.réalisateur = réalisateur
        self.date_emprunt = date_emprunt
        self.disponible = disponible
        self.emprunteur = emprunteur

    def afficher_details(self):
        return f"DVD: {self.name}, Réalisateur: {self.réalisateur},
Disponible: {self.disponible}"

class Cd:
    def __init__(self, name, artiste, date_emprunt="",
disponible=True, emprunteur=None):
        self.name = name
        self.artiste = artiste
        self.date_emprunt = date_emprunt
        self.disponible = disponible
        self.emprunteur = emprunteur

    def afficher_details(self):
        return f"CD: {self.name}, Artiste: {self.artiste},
Disponible: {self.disponible}"

class JeuDePlateau:
    def __init__(self, name, createur):
        self.name = name
        self.createur = createur

    def afficher_details(self):
        return f"Jeu de Plateau: {self.name}, Créateur:
{self.createur}"

class Emprunteur:
    def __init__(self, name, bloque=False):
        self.name = name
        self.bloque = bloque

    def afficher_details(self):
        return f"Emprunteur: {self.name}, Bloqué: {self.bloque}"

def menu():
    print("Menu Principal")
    print("1. Gestion des Médias")
    print("2. Gestion des Emprunteurs")
    print("3. Quitter")

def menuBibliotheque():
    print("Menu Bibliothécaire")
    print("1. Ajouter un Média")
    print("2. Ajouter un Emprunteur")
    print("3. Afficher les Médias")
    print("4. Afficher les Emprunteurs")
    print("5. Retourner un Emprunt")
    print("6. Quitter")

def menuMembre():
    print("Menu Membre")
    print("1. Consulter les Médias Disponibles")

```

```
print("2. Quitter")

if __name__ == '__main__':
    menu()
```

Par la suite, pour la création de l'application, j'ai transformé les classes en modèles Django, les fonctions en vues, et les menus en interfaces utilisateur avec des formulaires et des templates .

2.Mise en Place des Fonctionnalités Demandées

Fonctionnalités pour l'Application Bibliothécaire

Pour y accéder j'ai créé un utilisateur avec **python manage.py createsuperuser**

Nom : **utilisateur**

Mdp : **bibliothecaire**

J'ai également utilisé le décorateur **@login_required** en Django pour restreindre l'accès à une vue pour les utilisateurs non authentifiés.

1. Création et Gestion des Membres

Création d'un Membre :

- Vue pour afficher un formulaire de création.
- Formulaire POST pour enregistrer les détails du membre dans la base de données.

```
@login_required
def creer_emprunteur(request):
    if request.method == 'POST':
        form = EmprunteurForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, 'Emprunteur créé avec succès.')
            return redirect('liste_emprunteurs')
        else:
            messages.error(request, 'Formulaire invalide.')
    else:
        form = EmprunteurForm()
        return render(request, 'bibliothecaire/creer_emprunteur.html',
            {'form': form})
```

Affichage de la Liste des Membres :

- Vue pour afficher tous les membres avec des options pour voir plus de détails.

```
@login_required
def liste_emprunteurs(request):
    emprunteurs = Emprunteur.objects.all()
    return render(request, 'bibliothecaire/liste_emprunteurs.html',
                  {'emprunteurs': emprunteurs})
```

Mise à Jour d'un Membre :

- Vue pour afficher un formulaire pré-rempli avec les informations du membre à mettre à jour.

```
@login_required
def mettre_a_jour_emprunteur(request, pk):
    emprunteur = get_object_or_404(Emprunteur, pk=pk)
    if request.method == 'POST':
        form = EmprunteurForm(request.POST, instance=emprunteur)
        if form.is_valid():
            form.save()
            messages.success(request, 'Emprunteur mis à jour avec succès.')
            return redirect('liste_emprunteurs')
        else:
            messages.error(request, 'Formulaire invalide.')
    else:
        form = EmprunteurForm(instance=emprunteur)
        return render(request,
                      'bibliothecaire/mettre_a_jour_emprunteur.html', {'form': form})
```

2. Gestion des Médias et des Emprunts

Affichage de la Liste des Médias :

- Vue pour afficher les différents types de médias.

```
@login_required
def liste_medias(request):
    livres = Media.objects.filter(type_media='Livre')
    dvds = Media.objects.filter(type_media='DVD')
    cds = Media.objects.filter(type_media='CD')
    jeux_de_plateau = Media.objects.filter(type_media='Jeu de Plateau')
    return render(request, 'bibliothecaire/liste_medias.html', {
        'livres': livres,
        'dvds': dvds,
        'cds': cds,
        'jeux_de_plateau': jeux_de_plateau
    })
```

Création d'un emprunt :

- Vue pour créer un emprunt .

```
@login_required
def creer_emprunt(request):
    if request.method == 'POST':
        form = EmpruntForm(request.POST)
        if form.is_valid():
            emprunt = form.save(commit=False)
            media = emprunt.media
            if media.disponible:
                media.disponible = False
                media.save()
                emprunt.save()
                messages.success(request, 'Emprunt créé avec succès.')
                return redirect('liste_emprunts')
            else:
                form.add_error(None, 'Le média n\'est pas disponible.')
        else:
            messages.error(request, 'Formulaire invalide.')
    else:
        form = EmpruntForm()
    return render(request, 'bibliothecaire/creer_emprunt.html',
                  {'form': form})
```

Retourner un emprunt :

- Vue pour retourner un emprunt .

```
@login_required
def retourner_emprunt(request, pk):
    emprunt = get_object_or_404(Emprunt, pk=pk)
    if request.method == 'POST':
        emprunt.date_retour = date.today()
        emprunt.media.disponible = True
        emprunt.media.save()
        emprunt.save()
        messages.success(request, 'Emprunt retourné avec succès.')
        return redirect('liste_emprunts')
    return render(request, 'bibliothecaire/retourner_emprunt.html',
                  {'emprunt': emprunt})
```

Affichage des Emprunts :

- Vue pour afficher tous les emprunts en cours avec les détails de retour.

```
@login_required
def liste_emprunts(request):
    emprunts = Emprunt.objects.all()
    return render(request, 'bibliothecaire/liste_emprunts.html',
                  {'emprunts': emprunts})
```

Fonctionnalités pour l'Application Membre

Consultation des Médias

Affichage de la Liste des Médias :

- Vue pour afficher les différents types de médias disponibles.

```
def liste_medias_disponibles(request):  
    medias = Media.objects.filter(disponible=True)  
    return render(request,  
        'consultation/liste_medias_disponibles.html', {'medias': medias})
```

- Template : liste_medias.html

```
<!DOCTYPE html>  
<html>  
<head>  
    <title>Médias Disponibles</title>  
</head>  
<body>  
    <h1>Médias Disponibles</h1>  
    <ul>  
        {% for media in medias %}  
            <li>{{ media.titre }}</li>  
        {% endfor %}  
    </ul>  
</body>  
</html>
```

3.Stratégie de Tests

• Tests de Modèles :

- Vérification de la création et de la validité des instances des modèles (Livre, Dvd, Cd, JeuDePlateau, Emprunteur).
- Test de la gestion des emprunts pour s'assurer que les emprunts sont correctement associés aux membres et aux médias.

• Tests de Vues :

- Vérification que les vues renvoient les données correctes pour les listes de membres et de médias.
- Test de la fonctionnalité d'emprunt et de retour pour s'assurer que les emprunts sont correctement créés et traités.

Tests dans bibliothécaire\tests.py

Test de Création d'un Média :

```
class MediaModelTests(TestCase):
    fixtures = ['initial_data.json'] # Spécifie le fichier de fixture à
    utiliser

    def test_creation_livre(self):
        livre = Media.objects.create(type_media="Livre", name="1984")
        self.assertEqual(livre.name, "1984")
        self.assertEqual(livre.type_media, "Livre")
        self.assertTrue(livre.disponible) # Vérifie que le média est
        disponible par défaut

    def test_creation_dvd(self):
        dvd = Media.objects.create(type_media="DVD", name="Inception")
        self.assertEqual(dvd.name, "Inception")
        self.assertEqual(dvd.type_media, "DVD")
        self.assertTrue(dvd.disponible)
```

Test de Création d'un Emprunteur :

```
class EmprunteurModelTests(TestCase):
    fixtures = ['initial_data.json'] # Spécifie le fichier de fixture à
    utiliser

    def test_creation_emprunteur(self):
        emprunteur = Emprunteur.objects.create(name="John Doe")
        self.assertEqual(emprunteur.name, "John Doe")
        self.assertFalse(emprunteur.bloque) # Vérifie que l'emprunteur
        n'est pas bloqué par défaut

    def test_emprunt_media(self):
        emprunteur = Emprunteur.objects.create(name="Jane Doe")
        livre = Media.objects.create(type_media="Livre", name="1984")
        emprunteur.emprunts.add(livre)

        # Vérifie que l'emprunteur a bien emprunté le livre
        self.assertIn(livre, emprunteur.emprunts.all())
```

Lancement des tests : python manage.py test bibliothécaire

Test dans consultation\tests.py

Test du modèle consultation

```
from django.test import TestCase
from .models import Consultation
from bibliothecaire.models import Media, Emprunteur

class ConsultationModelTests(TestCase):
    fixtures = ['second_data.json'] # Charger les données initiales pour
    le test
```

```
def test_creation_consultation(self):
    consultation = Consultation.objects.get(pk=1)

    # Vérifier que la consultation a bien été créée avec les bonnes
    données
    self.assertEqual(consultation.emprunteur.name, "John Doe")
    self.assertEqual(consultation.media_consulte.name, "Python Basics")
    self.assertEqual(str(consultation.date_consultation), "2024-08-27")
    self.assertEqual(str(consultation), "John Doe a consulté Python
Basics")
```

Lancement des tests : python manage.py test consultation

4. Base de Données avec des Données de Test

1. Configuration de la Base de Données

Django utilise par défaut SQLite pour la base de données.

Configuration pour SQLite

```
# mediatheque/settings.py

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

2. Création et Migration de la Base de Données

Appliquer les migrations :

```
python manage.py makemigrations
python manage.py migrate
```

Pour insérer des données tests, j'ai ajouté des données test via un fichier de fixture JSON dans bibliothecaire/fixtures/initial_data.json et consultation/fixtures/second_data.json

.

Fichier initial_data.json

```
[
  {
    "model": "bibliothecaire.media",
    "pk": 1,
    "fields": {
      "type_media": "Livre",
      "name": "Python Basics",
      "date_emprunt": null,
      "disponible": true
    }
  }
]
```



```

    }
  },
  {
    "model": "bibliothecaire.emprunteur",
    "pk": 1,
    "fields": {
      "name": "John Doe",
      "bloque": false
    }
  }
]

```

Fichier second_data.json

```

[
  {
    "model": "consultation.media",
    "pk": 1,
    "fields": {
      "titre": "Python Basics",
      "disponible": true
    }
  },
  {
    "model": "consultation.media",
    "pk": 2,
    "fields": {
      "titre": "Advanced Python",
      "disponible": false
    }
  },
  {
    "model": "consultation.media",
    "pk": 3,
    "fields": {
      "titre": "Python Expert",
      "disponible": true
    }
  }
]

```

Les données ont été chargées avec :

python manage.py loaddata consultation/fixtures/second_data.json

J'ai alors vérifié les données dans Django Shell :

from consultation.models import Media

medias = Media.objects.filter(disponible=True)

print(medias)

Instructions pour Exécuter le Projet

1. Cloner le Repository :

git clone <https://github.com/DS30190/POO.git>

2. Naviguer dans le Répertoire du Projet :

cd POO

3. Créer un Environnement Virtuel :

python -m venv env

4. Activer l'Environnement Virtuel :

.\env\Scripts\activate

5. Appliquer les migrations :

python manage.py migrate

6. Démarrer le serveur de développement :

python manage.py runserver

7. Accéder à l'application :

<http://127.0.0.1:8000/>