Projet Médiathèque

RAMOS PINTO Tânia | Développeur web | 29 oct. 25

1. INTRODUCTION

Le projet **Médiathèque** a pour objectif de développer une application web permettant la gestion complète des médias (livres, CD, DVD et jeux de plateau), des membres et de leurs emprunts.

Initialement basé sur un code Python très simplifié, le projet a été entièrement refondu à l'aide du framework **Django**, afin d'obtenir une application robuste, maintenable et sécurisée.

Ce rapport présente successivement :

- L'étude du code initial et les correctifs apportés,
- La mise en place des fonctionnalités dans Django,
- La stratégie de tests et la validation fonctionnelle,
- La conclusion sur les résultats obtenus.

2. ÉTUDE ET CORRECTIFS DU CODE FOURNI

2.1 Contexte initial

Le code de départ était un simple script Python comportant quelques classes (*livre, dvd, cd, jeuDePlateau, Emprunteur*) et une fonction *menu().*
Ce prototype n'était pas exploitable dans un environnement applicatif moderne. Une analyse approfondie a donc permis d'identifier ses faiblesses avant la refonte complète du système.

2.2 Problèmes identifiés

Type d'erreur	Description	Conséquence
Nom des classes non conforme	Les classes (livre, dvd…) n'étaient pas en majuscule selon la convention PEP8.	Lisibilité réduite, non-respect des standards Python.
Attributs statiques	Les variables de classe étaient partagées entre toutes les instances.	Données incohérentes.
Absence de constructeur *__init__*	Aucun mécanisme d'initialisation des objets.	Impossible d'avoir des instances uniques.
Pas d'héritage	Chaque média redéfinissait les mêmes champs.	Redondance et maintenance difficile.
Aucune logique métier	Pas de gestion des emprunts, retours ou retards.	Fonctionnalités inexistantes.
Menus incomplets	Simple affichage console.	Aucune interactivité.
Aucune persistance des données	Pas de stockage ni de base de données.	Données perdues à chaque exécution.

2.3 Objectifs de la refonte

1. Appliquer les principes de **programmation orientée objet**.
2. Introduire une **hiérarchie par héritage** pour les médias.
3. Intégrer une **logique métier complète** (emprunt, retour, blocage).
4. Mettre en place une **base de données relationnelle** via Django ORM.
5. Créer une **interface web interactive et sécurisée**.

2.4 Correctifs techniques appliqués

➢ Refonte des modèles

Les anciennes classes ont été remplacées par une classe abstraite Media, héritée par :

- Livre
- CD
- DVD
- JeuDePlateau

Exemple :

```
class Media(models.Model):
    titre = models.CharField(max_length=100)
    disponible = models.BooleanField(default=True)
    date_emprunt = models.DateField(null=True, blank=True)
```

Les sous-classes ajoutent leurs champs spécifiques (auteur, artiste, réalisateur, créateur).

➢ Ajout de la logique métier

Des méthodes ont été introduites pour gérer le cycle de vie des médias :

- *emprunter()*
- *retourner()*
- *est_en_retard()*

Elles assurent automatiquement la mise à jour du statut des médias et le suivi des retards.

➢ Création du modèle Emprunteur

Lié à un *Membre* :

```
class Emprunteur(models.Model):
    membre = models.OneToOneField(Membre,
on_delete=models.CASCADE)
```

```
bloque = models.BooleanField(default=False)
```

Il gère :

- La vérification des retards,
- La limitation à trois emprunts,
- Le blocage automatique.

Gestion des emprunts

Un modèle *Emprunt* assure la traçabilité :

```
class Emprunt(models.Model):
    emprunteur = models.ForeignKey(Emprunteur, on_delete=models.CASCADE)
    livre = models.ForeignKey(Livre, on_delete=models.CASCADE, null=True,
    blank=True)
    cd = models.ForeignKey(CD, on_delete=models.CASCADE, null=True,
    blank=True)
    dvd = models.ForeignKey(DVD, on_delete=models.CASCADE, null=True,
    blank=True)
    date_emprunt = models.DateField(auto_now_add=True) date_retour =
    models.DateField(null=True, blank=True
```

Chaque emprunt est enregistré et suivi dans le temps.

➢ Refonte des menus → Interfaces web Django

Les anciens *print()* sont remplacés par des vues Django :

- *menuBibliotheque()* → interface bibliothécaire
- *menuMembre()* → interface membre

Ces vues s'appuient sur des templates HTML et des formulaires.

➢ Sécurisation

L'accès aux fonctionnalités sensibles est restreint grâce à *@login_required.*

2.5 Résultats après correction

Aspect	Avant	Après
Structure du code	Classes isolées	Hiérarchie claire (héritage Django)
Gestion des médias	Texte statique	Base de données relationnelle
Logique d’emprunt	Absente	Complète et automatisée
Interface	Menu console	Application web ergonomique
Sécurité	Aucune	Authentification Django
Maintenance	Difficile	Code propre et modulaire

3. MISE EN PLACE DES FONCTIONNALITES DEMANDEES

3.1 Structure du projet Django

L'application suit le modèle **MVT (Model – View – Template)** :

- **Models** : structure et logique métier
- **Views** : actions et contrôleurs
- **Templates** : interface web

3.2 Modèles de données principaux

➢ *Media (classe abstraite)*

Base commune des médias empruntables :

- titre, disponible, date_emprunt, emprunteur.
- Méthodes : emprunter(), retourner(), est_en_retard().

➢ *Livre, CD, DVD*

Sous-classes avec leurs champs propres (auteur, artiste, réalisateur).

➢ *JeuDePlateau*

Non empruntable — méthode *emprunter()* lève une exception.

➢ *Membre et Emprunteur*

Membre : informations de base.

Emprunteur : statut d'emprunt, blocage, vérification de retard.

- *Emprunt*

Historique complet des prêts (dates, emprunteur, média).

3.3 Vues principales

Vue	Rôle
liste_media()	Affiche et filtre les médias.
ajouter_media() / modifier_media()	Gestion du catalogue.
liste_membres()	Affiche les membres et leurs statuts.
emprunter_media()	Gère les emprunts avec contrôle des conditions.
retourner_media()	Enregistre le retour et met à jour le statut.
custom_login() / custom_logout()	Authentification des utilisateurs (bibliothécaire).

3.4 Sécurité

Toutes les opérations (emprunt, retour, suppression, modification, création) sont protégées par

@login_required.

Les erreurs sont gérées avec des messages d'alerte.

3.5 Données de test

Une base de données JSON a été créée pour initialiser le système avec :

- **8 livres**, **9 CD**, **12 DVD**, **10 jeux de plateau**, et **8 membres**.

Ces données facilitent la démonstration et la validation du fonctionnement de l'application.

4. STRATEGIE DE TESTS

4.1 Objectif

Assurer la fiabilité et la stabilité du système complet :

- logique métier,
- intégrité des données,
- interface web.

4.2 Outils

- Pytest
- Django Test Client
- Base temporaire générée automatiquement.

4.3 Types de tests

➢ *Tests unitaires*

- Création de Membre et Emprunteur.
- Vérification des attributs par défaut.

➢ *Tests de logique métier*

- Emprunt / retour d'un média.
- Blocage en cas de retard > 7 jours.
- Limitation à 3 emprunts.
- Interdiction d'emprunter un jeu.

➢ *Tests d'intégration (vues web)*

- Connexion / déconnexion d'utilisateur.
- Accès aux listes (membres, médias).
- Ajout, modification, suppression via formulaires.

4.4 Résultats

Domaine	Résultat
Création de membres et emprunteurs	✅ Réussi
Emprunts et retours	✅ Réussi
Blocage automatique	✅ Réussi
Limitation à 3 emprunts	✅ Réussi
Jeux non empruntables	✅ Réussi
Authentification	✅ Réussi
Affichage des vues	✅ Réussi

Les tests ont confirmé la **stabilité et la cohérence** de l'ensemble du système.

6. CONCLUSION GENERALE

Le projet Médiathèque est passé d'un prototype Python basique à une **application web Django complète et professionnelle**, offrant :

- Une **gestion centralisée** des médias,
- Un **suivi automatisé** des emprunts et retards,
- Une **interface utilisateur sécurisée**,
- Et une **structure modulaire et maintenable**.

Cette transformation illustre la montée en compétence technique et la compréhension globale des principes de développement web moderne avec Django.