

Rapport du Projet – Gestion Médiathèque

1. Étude et correctifs du code fourni

Le code initial présente à la fois des intentions louables en matière de structuration et plusieurs lacunes majeures à corriger pour garantir un fonctionnement correct et évolutif.

Points positifs

On constate tout d'abord une bonne volonté de modélisation du domaine : les entités métiers sont correctement identifiées avec des classes distinctes pour les livres, DVD, CD et jeux de plateau. De même, la distinction entre bibliothécaire et membre est amorcée via deux fonctions de menus séparés. Le code contient les champs essentiels attendus dans une gestion de médiathèque (nom, auteur, disponibilité, emprunteur, date d'emprunt), et présente un point d'entrée explicite via le bloc `if __name__ == "__main__"`, ce qui témoigne d'une compréhension des bonnes pratiques de base en Python.

Points négatifs

Cependant, le code souffre de nombreuses limitations. Les classes ne possèdent pas de constructeur `__init__` et se limitent à des structures de type liste ou dictionnaire, sans encapsulation ni logique métier. Il existe une forte redondance entre les différentes classes de média, ce qui aurait pu être évité grâce à l'héritage. Les fonctionnalités fondamentales sont absentes : aucune méthode ne permet de gérer les emprunts, les retours ou la vérification de disponibilité. De plus, aucune donnée n'est persistée : tout est perdu à la fermeture du programme. Les menus sont très rudimentaires, ils ne permettent pas une interaction fonctionnelle avec le système, et les objets ne sont pas reliés entre eux (par exemple, un emprunteur est représenté par une simple chaîne de caractères). Le nommage des classes ne respecte pas la convention PEP8 (les noms devraient être en PascalCase, comme Livre au lieu de livre), et enfin, tout le code est regroupé dans un seul fichier, ce qui rend sa lisibilité et sa maintenabilité plus difficile.

2. Mise en place des fonctionnalités demandées

2.1 Présentation générale du projet

Le projet consiste en une application web de gestion d'une médiathèque, développée avec Django. L'application permet de gérer les membres, les médias (livres, CDs, DVDs, jeux de plateau), ainsi que le prêt et le retour des médias. L'objectif est de fournir un outil simple et efficace pour le personnel de la médiathèque afin de suivre les emprunts, vérifier les retards et administrer la base de données des utilisateurs et des ressources.

2.2 Fonctionnalités principales

Gestion des membres

- Affichage de la liste complète des membres inscrits à la médiathèque.
- Ajout, modification et suppression des membres via des formulaires sécurisés.
- Validation des données des membres avec des formulaires Django personnalisés.
- Messages de confirmation ou d'erreur pour accompagner chaque action.

Gestion des médias

- Affichage de la liste de tous les médias disponibles, regroupant plusieurs types (livres, CDs, DVDs, jeux de plateau).
- Ajout de médias avec des formulaires spécifiques à chaque type de média (LivreForm, CDForm, DVDForm, JeuDePlateauForm).
- Suppression de médias de manière sécurisée.
- Recherche et récupération des médias via une fonction utilitaire pour gérer les différentes sous-classes de médias.

Emprunt et retour de médias

- Fonctionnalité d'emprunt d'un média par un membre, avec vérification des disponibilités et des données d'entrée.
- Retour des médias empruntés avec mise à jour des états dans la base de données.
- Gestion des emprunts en retard avec vérification automatique et blocage des membres en retard.
- Interface utilisateur permettant d'effectuer ces opérations via des formulaires et des actions POST, avec feedback visuel grâce au système de messages Django.

Sécurité et accessibilité

- Protection des vues sensibles avec des décorateurs de type `login_required` et contrôle d'accès via `user_passes_test` pour les actions réservées aux administrateurs (`superusers`).
- Gestion correcte des erreurs HTTP (ex. `HttpResponseBadRequest`) pour assurer la robustesse de l'application.
- Utilisation du framework Django pour le routage, la gestion des formulaires, et la validation des données.

2.3 Architecture technique

- Modèles : La structure des données repose sur un modèle abstrait Media et des sous-classes spécialisées (Livre, CD, DVD, etc.) pour représenter les différents types de médias.
- Vues : Les vues Django sont écrites en style fonctionnel, avec des contrôles précis des méthodes HTTP et des opérations de CRUD (Create, Read, Update, Delete).
- Formulaires : Chaque entité dispose d'un formulaire dédié, garantissant une saisie et une validation adaptées.
- Messages : Le système de messages Django est utilisé pour informer l'utilisateur du résultat des actions (succès, erreur, etc.).
- Gestion des erreurs : Les vues renvoient des erreurs claires en cas de données manquantes ou d'actions invalides.

2.4 Particularités et choix techniques

- Recherche des médias polymorphe : la récupération des objets média est réalisée en interrogeant successivement les modèles enfants pour retrouver l'instance correspondante à un identifiant donné.
- Centralisation des fonctionnalités critiques comme l'emprunt et le retour des médias au sein du modèle Emprunt, afin de garantir une logique métier cohérente.
- Utilisation de décorateurs personnalisés pour gérer la sécurité, combinant authentification et autorisation.

3. Stratégie de test

3.1 Objectif des tests

L'objectif principal des tests est de garantir que les fonctionnalités développées fonctionnent correctement et répondent aux exigences du projet. Ils permettent aussi de vérifier la robustesse du code face à des cas courants d'utilisation ainsi qu'à des scénarios limites.

3.2 Tests manuels

Dans un premier temps, j'ai réalisé des tests manuels en naviguant sur l'application web via le navigateur, afin de vérifier le bon fonctionnement des principales fonctionnalités :

- Création, modification, affichage et suppression des membres.

- Ajout, affichage et suppression des médias (livres, DVDs, etc.).
- Emprunt et retour des médias, avec mise à jour de leur disponibilité.
- Gestion des cas d'erreurs, comme la tentative d'emprunt d'un média déjà emprunté ou d'un membre bloqué.
- Contrôle de la sécurité, notamment l'accès restreint aux fonctions administratives.

Ces tests manuels ont permis de détecter et corriger rapidement les bugs évidents liés aux formulaires et aux opérations CRUD.

3.3 Tests automatisés

Pour assurer une couverture systématique et reproductible, j'ai développé un ensemble de tests automatisés avec le framework de tests intégré à Django (`django.test.TestCase`).

Ces tests couvrent :

- Modèles (Models) :
 - Création et validation d'instances Membre, Media et leurs sous-classes (Livre, DVD).
 - Vérification de l'héritage entre Media et ses sous-classes.
 - Suppression correcte des objets de la base de données.
- Vues (Views) :
 - Accès aux listes de membres et médias, en vérifiant la présence des données affichées dans les [templates](#).
 - Fonctionnalités d'ajout, modification et suppression des membres et médias via des requêtes [POST](#) et [GET](#).
 - Processus d'emprunt d'un média disponible par un membre non bloqué.
 - Retour d'un média emprunté avec mise à jour de l'état et des messages utilisateurs.
 - Vérification du statut [HTTP](#) des réponses (200 pour affichage, 302 pour redirections).
 - Gestion des messages flash (via le système de messages Django) confirmant les actions.
- Cas particuliers et erreurs :
 - Tentatives d'emprunt sur média indisponible.
 - Gestion des redirections après modification ou suppression.
 - Mise à jour correcte de la disponibilité des médias lors d'un emprunt ou d'un retour.

4. Base de données avec données de test

Pour tester et faire fonctionner l'application de gestion de médiathèque, j'ai utilisé plusieurs types de données représentant les membres et les médias.

Membres


J'ai créé plusieurs membres fictifs, avec des informations telles que le nom et l'email. Par exemple :

- Luc (luc@exemple.com)
- Marc (marc@exemple.com)
- Jean (jean@example.com)
- Carl (carl@exemple.com)

Ces membres permettent de tester la gestion des emprunts, la modification et la suppression des profils, ainsi que la gestion des accès.

Médias

J'ai utilisé différents types de médias, chacun avec un statut de disponibilité :

- Livres (ex. : LIVRE 1, LIVRE 2), certains empruntés (indiqués par )
- CDs (ex. : CD 1), disponibles
- DVDs (ex. : DVD 1), empruntés
- Jeux de plateau (ex. : JEU DE PLATEAU 1), disponibles

Ces données permettent de tester les fonctionnalités d'emprunt, de retour, d'ajout et de suppression des médias, ainsi que la mise à jour de leur statut.

Instructions pour exécuter le programme depuis n'importe quelle machine

Ce projet Django peut être exécuté facilement sans prérequis particuliers.

Pour lancer l'application, il suffit de suivre les étapes suivantes :

1. Cloner le dépôt GitHub sur la machine locale.
2. Installer Python (version 3.8 ou supérieure) si ce n'est pas déjà fait.
3. Créer un environnement virtuel et l'activer.
4. Installer les dépendances avec la commande : `pip install -r requirements.txt`
5. Appliquer les migrations de la base de données : `python manage.py migrate`
6. Démarrer le serveur de développement Django : `python manage.py runserver`
7. Ouvrir un navigateur et accéder à l'adresse suivante : <http://127.0.0.1:8000/>

Ces étapes permettent d'exécuter le programme sur n'importe quelle machine disposant de Python et d'un accès internet pour installer les dépendances.