# **Rapport de projet — Application de gestion de médiathèque**

## **1. Explication des choix techniques**

Le projet a été développé avec **Python** et le framework **Django** car celui-ci permet de gérer facilement :

* La séparation logique entre le modèle, la vue et le contrôleur (MVT)
* La persistance des données via une base relationnelle (SQLite)
* L’interface d’administration intégrée pour les bibliothécaires
* Le routage et les vues dédiées pour les membres et le personnel

J’ai aussi utilisé :

* virtualenv pour isoler l’environnement
* pytest/unittest pour les tests
* requirements.txt pour la reproductibilité

## **2. Étude du code initial**

Le code fourni initialement consistait en des classes simples sans encapsulation ni logique métier, par exemple :

class livre():  
 name = ""  
 auteur = ""

Problèmes détectés :

* Nommage non conforme (pas de majuscule, noms en français)
* Pas de séparation en modèles communs
* Répétition du même schéma de données
* Aucune logique de validation métier

Ce que j’ai conservé : la **volonté de séparer les types de médias**.

Ce qui a été refondu :

* Création d’un modèle Media générique avec un champ type\_media
* Introduction des classes Emprunteur et Emprunt avec logique métier intégrée

## **3. Fonctionnalités mises en place**

### **Application bibliothécaire**

* Ajouter / modifier / supprimer un **emprunteur**
* Ajouter / modifier / supprimer un **média**
* Créer un **emprunt** (si l'utilisateur est éligible)
* Retour d’un emprunt
* Liste des membres, des médias, et des emprunts
* Interface admin via /admin

### **Application membre**

* Accès à un **catalogue de médias**
* Filtrage par **disponibilité**
* Barre de **recherche**
* Affichage clair des statuts de chaque média

### **Contraintes métiers respectées :**

* Pas plus de **3 emprunts actifs**
* Un **retard bloque** les emprunts
* Les jeux de plateau ne peuvent pas être empruntés
* Délai d’un **emprunt = 1 semaine**

## **4. Stratégie de tests**

Objectifs des tests pour membres :

* Vérifier le fonctionnement des **vues membres**
* Tester la **recherche** et le **filtrage**
* Vérifier les règles métier (nombre max, retard, etc.)

Objectifs des tests bibliothécaires

* Un membre peut emprunter **si tout est OK**
* Un membre **ne peut pas emprunter** s’il a :
  1. plus de 3 emprunts actifs
  2. un retard
* Vérifier que le retour d’un emprunt met bien à jour l’état

Commandes utilisées :

python manage.py test

Les tests se trouvent dans membre/tests.py et bibliothecaire/tests.py

## **5. Contenu de la base de données test**

Quelques jeux de données sont ajoutés pour le test :

* 3 médias : un livre, un CD, un DVD (avec disponibilité variée)
* 1 emprunteur
* Quelques emprunts (actifs ou rendus)

## **6. Instructions d’exécution**

### **Prérequis : Python 3.10+**

### **1. Cloner le projet**

git clone https://github.com/Marion802/mediatheque.git  
cd mediatheque

### **2. Créer un environnement virtuel**

python -m venv env  
source env/bin/activate # Sur Linux/macOS  
env\Scripts\activate.bat # Sur Windows

### **3. Installer les dépendances**

pip install -r requirements.txt

### **4. Appliquer les migrations**

python manage.py migrate

### **5. (Optionnel) Créer un superutilisateur**

python manage.py createsuperuser

### **6. Lancer le serveur**

python manage.py runserver

Puis ouvrir le navigateur à http://127.0.0.1:8000/

## **Structure du projet**

mediatheque/  
├── bibliothecaire/  
├── membre/  
├── templates/  
├── db.sqlite3  
├── manage.py  
└── requirements.txt