Projet Médiathèque Django

SOMMAIRE

1. Introduction

2. Étude du code à reprendre

3. Modélisation et règles métiers

4. Tests

5. Instructions d’exécutions

1. Introduction

Ce projet a été réalisé dans le cadre de ma formation de développeur web et web mobile.
Il s'agit d'une application Django permettant de gérer une médiathèque avec deux espaces distincts : un espace public, destiné aux membres pour consulter les médias, et un espace staff, réservé aux bibliothécaires pour gérer les emprunteurs, les médias et les emprunts.

2. Étude du code à reprendre

Le document fourni contenait une première version du projet en Python "classique", sans framework.

- ## Gestion des menus :
  - Le code initial utilisait des fonctions de menu (menu, menuBibliothèque, menuMembre) pour naviguer entre les fonctionnalités. Dans Django, cette logique est remplacée par les **views** et les **URLs**, qui permettent de gérer la navigation directement via le navigateur et les **templates HTML**.
  - Le script se terminait par une condition d'exécution :

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

Dans Django, ce mécanisme est intégré à **manage.py**, qui utilise :

```
if __name__ == "__main__":
    main()
```

- ## Défauts relevés dans les classes :

Le code initial comportait plusieurs points non conformes aux bonnes pratique de Python :

  - Les classes doivent commencer par une majuscule (convention PEP8) or, seule la classe Emprunteur respectait cette règle.
  - Dans Python, les parenthèses ne sont nécessaires que dans le cas d'un héritage. Certaines classes avaient été déclarées avec des parenthèses inutiles.

- **Problèmes d'indentation :**
  - Python repose sur l'indentation pour délimiter les blocs de code. Des erreurs étaient présentes sur plusieurs attributs de classe, ces attributs étaient décalés de 4 espaces en trop, rendant le code invalide.

- **Duplication du code :**
  - Les classes Livre, Dvd et Cd définissaient toutes les mêmes attributs en plus de leurs attributs spécifiques. La solution retenue a été de créer une classe mère Media regroupant ces champs communs. Les classes spécialisées héritent de cette base, réduisant la duplication et améliorant la lisibilité.

3. Modélisation et règles métiers

Au cœur du projet se trouve une classe abstraite **Media**, qui contient les champs communs à tous les médias : un titre et un booléen indiquant si le média est disponible ou non.

Cette classe est spécialisée en plusieurs modèles :

- **Livre**, qui ajoute l'attribut auteur ;
- **Dvd**, avec l'attribut réalisateur ;
- **Cd**, avec l'attribut artiste ;
- **JeuDePlateau**, qui constitue un cas particulier : il est consultable mais ne peut jamais être emprunté.

Les utilisateurs de la médiathèque sont représentés par le modèle **Emprunteur**, défini par un nom et un attribut indiquant s'il est bloqué. Un emprunteur peut avoir plusieurs emprunts en cours, mais certaines règles strictes doivent être respectées.

Le modèle **Emprunt** relie ainsi un emprunteur à un seul média (livre, DVD ou CD) avec une date d'emprunt et une date de retour. Ce modèle applique les règles métiers suivantes :

- un emprunteur ne peut pas avoir plus de trois emprunts simultanés ;
- la durée maximale d'un emprunt est de 7 jours, au-delà l'emprunteur est bloqué ;
- un emprunt ne peut se faire que sur un média disponible ;
- les jeux de plateau ne sont pas empruntables.

4. Tests

Afin de vérifier le bon fonctionnement de chaque fonctionnalité, une série de tests unitaires a été mise en place.
Chaque application contient un dossier tests/, avec par exemple :

- **core/tests/** : couvre la logique métier.
- **staff/tests/** : couvre les vues côté bibliothécaire.
- **member/tests/** : couvre l'accès aux pages accessible par les membres.

Cette séparation permet de tester chaque module de manière indépendante.

5. Instructions d'exécutions

Toutes les instruction d'installation et d'exécution sont réunies dans le fichier README.md du projet que vous pouvez trouvez ici:

https://github.com/MaitreGobz/Projet-mediateque-Django/blob/main/README.md