**Rapport de projet**

**Introduction :**

Une médiathèque portant le nom de « Notre livre, notre média » m’a contacté afin de moderniser son système de gestion interne qui est pour l’instant géré à l’ancienne par des fiches papiers. L’objectif est d’avoir un système fonctionnel, sécurisé, utilisable par les bibliothécaires et également par le public.

La médiathèque propose à la consultation et à l’emprunt des livres, des Cds, des Dvds. Seuls les jeux de plateaux sont disponibles en consultation

Deux applications à déployer :

* Une application principale qui ne sera accessible qu’aux bibliothécaires,
* Une application de consultation accessible aux emprunteurs.

**I – Étude et correctifs du code fourni**

1. **Étude du code fourni**

Dans un premier temps, on remarque une méthode « menu » qui affiche donc un menu.

Nous pouvons également retrouver dans ce code, différentes classes constituées de variables.

Enfin, nous avons 2 méthodes qui permettent d’afficher soit l’application bibliothécaire soit l’application membre.

Voici un tableau récapitulatif des principales normes à respecter et si le code fourni les respecte :

|  |  |  |
| --- | --- | --- |
|  | La norme de convention à appliquer | Est-ce respecté par le code fourni ? |
| Pour les attributs | Le symbole ‘\_’ (tiret bas) se place en séparateur de chaque mot. Il s’agit de la norme Snake case. | Non utilisé |
| Un nom d’attribut doit commencer par une lettre minuscule, soit un tiret bas (jamais un chiffre). | **Oui** |
| Un nom de variable doit idéalement être écrit en anglais. | **Non**, il y a un mélange d’anglais et de français |
| Le nom d’un attribut reflète son contenu et soit explicite. | **Oui** |
| Les conventions PEP8 | Un espace autour des opérateurs d’affectation et opérateur logique (sauf pour les valeurs par défaut) | **Oui** |
| Pas d’espace entre une fonction et ses arguments | **Oui** |
| Pas d’espaces entre les parenthèses d’une fonction/d’une classe et les deux-points | **Non**, de temps en temps un espace a été mis |
| Pas d’espace après une parenthèse ouvrante | **Oui** |
| Deux sauts de ligne après la définition d’une classe ou d’une fonction | **Non**, 3 sauts de lignes ont été utilisé entre 2 méthodes |
| Un seul saut de ligne avant la définition d’une méthode | **Non**, 2 sauts de lignes ont été effectué |
| La longueur d’une ligne de code doit être limitée à 79 caractères | **Oui** |
| Utilisation de 4 espaces pour indenter les blocs de code | **Non**, certaines variables ont 8 espaces au lieu de 4 |
| Pour les classes | Le mot-clé « class » pour définir une classe | **Oui** |
| Le nom d’une classe commence toujours par une majuscule et doit être écrit en CamelCase | **Non** |
| Les attributs caractéristiques d’une classe sont déclarés en premier, suivent ensuite les méthodes | **Oui** |
| Les méthodes | Commencent par le mot clé « def » | **Oui** |
| Une méthode doit avoir un nom et des ( ) | **Oui** |
| Tout le code à l’intérieur d’une méthode devra être indenté de 4 espaces à partir du mot réservé “def”, en dessous de la signature | **Oui** |
| Chaque méthode possède son propre nom | **Oui** |
| Le nom d’une méthode doit être écrit en snake\_case | **Non** |

1. **Les correctifs du code fourni**

A corriger :

* Mise en place des majuscules pour les noms des classes et écrire en CamelCase,
* Mise en place du nom des classes et attributs en anglais,
* Mise en place du nom des méthodes en snake\_case,
* L’indentation correcte des attributs,
* Mise en place de l’héritage entre Media et Book, Dvd et Cd,
* Les espaces avant les deux points ont été supprimés,
* Tous les mêmes noms des attributs ont été mis dans la classe Media pour pouvoir le faire hériter ensuite dans les classes filles et ainsi éviter d’avoir trop lignes de code.

**II – Mise en place des fonctionnalités demandées**

1. **Les models (models.py)**

Ma première étape était de définir les models de l’application. En effet, j’ai donc repris le code fourni et je l’ai modifié comme vue précédemment.

Ici, il fallait mettre en héritage la classe mère Media avec les classes filles Book, Dvd et Cd. Ceci est faisable grâce à « models.Model » qui permet de représenter des tables dans la base de données avec des relations et des méthodes. Le fait de mettre ensuite « Media » dans les classes filles permet de faire le lien et donc de pouvoir réutiliser les variables de la classe mère. Chaque attribut correspond à une colonne dans cette table mère.

Après avoir consulté la documentation de django, j’ai pu insérer différents types de champs aux attributs :

* CharField : pour les chaînes de caractères courtes ou longues
* DateField : pour une date
* BooleanField : pour un champ vrai/faux
* ForeignKey : pour une relation many-to-one 🡪 cette colonne est en relation avec un autre modèle.

Les méthodes « def \_\_str\_\_(self) » permettent d’afficher l’objet sous forme de texte.

1. **La vue (views.py)**

En ce qui concerne les méthodes dans la vue, j’ai utilisé les docstrings. Ainsi, il est plus facile d’expliquer chaque méthode du projet.

J’ai d’abord commencé par créer la fonctionnalité « Créer un membre-emprunter ». Ensuite, j’ai créé la fonctionnalité « Afficher la liste des membres » et toutes les modifications qu’on pouvait apporter à ce membre-emprunteur : mettre à jour et supprimer le membre. Puis, j’ai créé la fonctionnalité « Afficher la liste des médias » et comme on ne peut pas avoir une liste sans média, j’ai créé « Ajouter un média » avec un bouton d’ajout pour chaque type de média. Enfin, les deux dernières qui vont ensemble : « Créer un emprunt pour un média disponible » et « rentrer un emprunt ».

A cela s’ajouter les différentes contraintes :

* Un membre ne peut pas avoir plus de 3 emprunts à la fois,
* Un emprunt doit être retourné au bout d’une semaine,
* Un membre ayant un emprunt en retard ne peut plus emprunter,
* Les jeux de plateaux ne sont pas concernés par les emprunts.

1. **Les templates (fichiers.html)**

Pour plus de légèreté dans la lecture du code, j’ai décidé de créer plusieurs fichiers html. En effet, il était plus simple de gérer les fonctionnalités comme ceci.

J’ai rencontré quelques problèmes lors de la création d’ajout des médias car mon premier objectif était de créer une seule page html pour l’ajout des médias, sélectionner le média et lors de la sélection, il y a l’affichage du formulaire avec les bons champs. Cependant, en masquant les autres formulaires, il ressortait une erreur car il fallait que je remplisse les champs obligatoires des autres formulaires alors que je ne les voyais pas. Impossible de résoudre le problème car je voulais rester sur un formulaire django. J’ai donc décidé de créer un bouton pour chaque type de média qui me renverrait vers la page qui contient le formulaire correspondant.

Il était donc plus simple de créer une page html pour chaque fonctionnalité.

1. **Les urls (urls.py)**

Afin d’afficher les pages html créées dans le dossier « templates », il faut que je crée le chemin. C’est donc dans le fichier urls.py que j’ai pu le créer. Certains chemins demandaient d’inclure « <int/member\_id> » car il fallait que je sélectionne le membre sur lequel j’avais cliqué pour faire des modifications uniquement sur celui-ci. De plus, il fallait que j’associe la vue que j’avais créé dans views.py à chaque chemin pour que les fonctionnalités soient associées.

1. **Les formulaires (forms.py)**

J’ai inséré un fichier forms.py pour gérer les formulaires pour la création d’un membre, l’ajout d’un livre, cd et dvd. J’ai donc créé plusieurs classes auquel j’ai associé plusieurs champs afin qu’ils apparaissent sur le formulaire.

**III – Stratégie de tests**

1. **Les logs**

Dans ce projet, nous devions mettre en place des logs. Ils permettent de suivre le déroulement de la méthode et de diagnostiquer les problèmes. Plusieurs types de logs ont été insérer et dont la description de la documentation django est :

* Log info : « information système générale. »
* Log debug : « information système de bas niveau à des fins de débogage. »
* Log warning : « information décrivant la présence d’un problème mineur. »
* Log error : « information décrivant la présence d’un problème majeur. »

1. **Les tests (tests.py)**

**Pour la création d’un membre-emprunteur :**

J’ai choisi de mettre un test pour vérifier que la vue traite correctement les données valides. En effet, ça permet de vérifier que le membre a bien été créé dans la base de données quand le formulaire est envoyé avec des données valides. De plus, il vérifie que les données du membre sont bien celles envoyées dans le formulaire. Enfin, il vérifie que l’utilisateur est correctement redirigé.

**Pour l’affichage de la liste des membres :**

J’ai choisi de mettre un test pour vérifier si la vue est accessible, si elle affiche les correctement les membres existants en utilisant le bon template. En effet, ça permet de vérifier le respect de la logique de récupération des membres.

**Pour mettre à jour un membre :**

J’ai choisi de mettre un test pour valider le fonctionnement de la vue pour mettre à jour un membre. En effet, il vérifie que les données du membre sont mises à jour avec succès, que la redirection fonctionne comme prévu et que la base de données aura les nouvelles données du membre.

**Pour supprimer un membre :**

J’ai choisi de mettre un test pour vérifier que la suppression d’un membre se fait correctement. En effet, il vérifie donc que le membre sera supprimé de la base de données, qu’il y a une redirection de l’utilisateur vers la liste des membres et qu’aucun objet Borrower avec l’ID supprimé n’est pas récupérable.

**Pour afficher la liste des médias :**

J’ai choisi de mettre un test pour vérifier que la vue renvoie correctement la liste contenant tous les médias avec leurs informations. En effet, le test assure que la vue renvoie tous les médias disponibles dans la base de données peu importe leur type et il confirme que les noms des médias sont bien affichés sur la page HTML .

**Pour créer un emprunt pour un média disponible :**

J’ai choisi de mettre un test pour vérifier que la vue permet la création d’un emprunt pour un média disponible correctement. En effet, le test vérifie que l’utilisateur peut emprunter un média disponible, qu’il y a une mise à jour ensuite de la disponibilité du média pour pas que les membres puissent l’emprunter tant qu’il n’a pas été retourné et que l’utilisateur soit bien redirigé après la création de l’emprunt.

**Pour ajouter un média :**

J’ai choisi de mettre un test pour chaque méthode d’ajout de média (livre, dvd, cd) pour vérifier que chaque type de média a été correctement ajouté à la base de données. En effet, le test vérifie que les données soumises par le formulaire ont correctement été traités et qu’elles sont valides. Ensuite, il vérifie que la base de données à bien été mise à jour et que la redirection est correcte.

**Pour rentrer un emprunt :**

J’ai choisi de mettre un test pour vérifier que la fonctionnalité de retour d’un emprunt fonctionne correctement. En effet, le test assure la cohérence des données en marquant l’emprunt comme retourné pour que le média soit à nouveau disponible. Il vérifie les étapes de retour et empêche donc les erreurs.

**IV – Base de données avec des données tests**

Pour accéder à la base de données avec des données tests, il suffit d’aller dans le terminal de PyCharm et taper : python manage.py test bibliothécaire. Django permet d’utiliser une base de données temporaire dédiée pour exécuter les tests de l’application.

**V – Instructions d’installation**

Les instructions ont directement été mis dans le fichier README.md.