Darnaudat Nathalie N° Elève : F449811W

Rapport de Travail :

Logiciel de gestion

SOMMAIRE

I- Etude et correctif du code fourni

Au sein du développement de mon projet, je me suis appuyée sur le code préalablement fourni.

J'ai en effet utilisé les différentes classes proposées :

- Livre,
- Dvd,
- Cd,
- JeuDePlateau,
- Emprunteur.

J'ai néanmoins adapté ces classes.

Tout d'abord, je les ai renommées car par convention, en Python, il est recommandé l'utilisation de l'anglais plutôt que du français. Les classes sont donc devenues :

- Book pour Livre,
- Board pour JeuDePlateau.

De plus, afin d'éviter des répétitions dans les classes Livre, Dvd, cd et JeuDePlateau, j'ai créé une classe Media qui est la classe « Mère » où j'ai renseigné les différents attributs communs à ces différentes classes. Ceci me permet d'utiliser l'héritage sur ces dernières.

Pour les classes « Enfants », j'ai rajouté des attributs qui me semblaient importants pour la conception du logiciel de gestion de la médiathèque.

Concernant le classe Emprunteur, je l'ai scindée en 2 :

- Une classe Member pour les membres,
- Une classe Loan pour les emprunts.

La classe Member reprend les différentes données concernant la personne telle que le nom, le prénom, l'email, le numéro de téléphone ainsi que la date de création du membre au sein de la base de données.

La classe Loan a comme attributs la date d'emprunt, la date de retour prévue ainsi que la date de retour effective. La date de retour prévue est calculée grâce à la fonction « get_default_loan_date » qui retourne le résultat de la date d'emprunt enregistrée plus 7 jours (le timedelta). Cette classe est également liée aux classes Member et Media par le biais d'une « ForeignKey ».

Les contraintes comme le fait de bloquer un membre s'il y a un retard de retour ou s'il a plus de 3 emprunts en même temps à son actif ne sont pas incluses dans les classes comme le suggère la classe Emprunteur dans le code fournit mais son définit dans des méthodes d'instance qui sont des constructeurs que je vais détailler :

- La méthode « __str__ » de la classe Member permet le retour du nom et prénom du membre.

- La méthode « __str__ » de la classe Media permet le retour du nom du média.

- La méthode « __str__ » de la classe Loan permet le retour du nom du média emprunt et de l'emprunteur.

- La méthode « check_borrowing_limit » contrôle si le membre a atteint la limite des 3 emprunts à son actif et entraine l'impossibilité d'emprunter un nouveau média si cette condition est vérifiée et message indiquant la raison de ce refus sera affiché.

- La méthode « check_late_loans » vérifie si un membre a un retard sur un retour d'emprunt et si cette condition est vraie alors un message indiquant que l'emprunteur a un retard de retour s'affichera et il sera impossible de créer un nouvel emprunt temps que la condition n'est pas levée.

- La méthode « check_availability_media » permet de vérifier si le média est disponible ou non à l'emprunt. S'il n'est pas disponible, un message l'indiquera.

- Les méthodes « mark_media_as_available » et « mark_media_unavailable » permettent la mise à jour de la disponibilité ou non d'un média en cas d'emprunt de celui-ci.

- La méthode « save » permet la sauvegarde après avoir vérifié si un emprunt était en cours et combien et la disponibilité du média.

II- Mise en place des fonctionnalités demandées

Les fonctionnalités que j'ai utilisées dans mon fichier views.py et forms.py peuvent se découper en plusieurs parties :

- L'accueil,
- La gestion des membres,
- La gestion des médias,
- La gestion des emprunts,

Je vais expliquer le fonctionnement de chacune des fonctions de ces différentes parties.

L'accueil :

- La fonction « home » permet l'accès à la page d'accueil principale de la médiathèque permettant l'accès pour les bibliothécaires et les membres.

- La fonction « home_librarian » permet d'accéder à la page d'accueil pour les bibliothécaires par le biais du return. En revanche, l'accès à cet espace est restreint et il faut donc des identifiants pour y accéder.
  Cette restriction se fait par un formulaire faisant appel à un login et le fichier settings.py de notre application principale a été modifié pour permettre les autorisations.

- La fonction « member_home » permet l'accès à la page d'accueil pour les membres par le biais du return.

La gestion des membres :

- La fonction « listmembers » permet d'afficher la liste de tous les membres et pour chacun d'entre eux d'afficher s'ils ont un ou des emprunts en cours. Si une erreur se produit lors du chargement de la page car un membre n'est pas récupéré correctement, un message d'erreur apparaîtra.

- La fonction « addmember » permet l'ajout d'un nouveau membre par le biais d'un formulaire avec comme champs le nom, prénom, email, numéro de téléphone et la date de création de la fiche de celui-ci. Seul le champ email n'est pas obligatoire pour valider la création du membre. Si un champ obligatoire n'est pas complété, une notification s'affichera pour informer quel champ doit être rempli pour valider la mise à jour et si l'ajout se fait correctement un message de succès apparaîtra.

- La fonction « memberupdate » permet la mise à jour des informations concernant un membre grâce à son id. Un formulaire prérempli avec les informations déjà enregistrées permettra cette fonction. Au préalable, une vérification de l'existence du membre sera effectuée et si le membre n'existe pas, un renvoi sur une page erreur 404 sera fait. Sur le même principe, un message de succès s'affichera lors de la mise à jour du membre.

- La fonction « membredelete » permet la suppression d'un membre. Celui-ci est récupéré dans la base de données par son id. Un message de succès s'affichera à la suppression de la fiche et l'utilisateur sera redirigé vers la page de la liste des membres.

- La fonction « list_medias_member » de l'application 'app_membre' permet d'afficher la liste des médias configurées depuis l'espace bibliothécaire en mentionnant si le média est disponible à l'emprunt ou non.

La gestion des médias :

- La fonction « listmedia » permet d'accéder la liste de tous les médias présents au sein de la médiathèque en affichant pour chacun d'eux s'ils sont disponibles ou non à l'emprunt.
  Si un média est en cours d'emprunt, le nom de l'emprunteur ainsi que la date d'emprunt et la date de retour limite qui seront indiqués.

- La fonction « addmedia » permet l'accès au template « ajoutmedia ».

- Les fonctions « add_book », « add_dvd », « add_cd » et « add_board » permettent l'ajout d'un nouveau média à l'aide d'un formulaire. Lors de l'ajout d'un nouveau média, un message de succès s'affichera et une redirection vers la page de tous les médias sera faite.

La gestion des emprunts :

- La fonction « create_loan » permet la création d'un emprunt toujours à l'aide d'un formulaire. Le paramètre catégorie est récupéré afin de pouvoir filtrer les médias selon leur type. A l'actualisation du formulaire après la sélection de la catégorie, on pourra sélectionner le nom du membre qui souhaite faire cet emprunt via une liste déroulante, sélectionner le média a emprunté et entrer la date de l'emprunt. A la validation du formulaire, un message de succès s'affichera et la date de retour prévue sera automatiquement calculée avec le 'timedelta' configuré à 7 jours.
  Le média sera marqué comme indisponible au niveau de la liste des médias en mentionnant les nom et prénom de l'emprunteur ainsi que les dates d'emprunt et de retour prévues.
  Le média emprunté sera noté sur la fiche de l'emprunteur.
  Le média sera aussi marqué comme indisponible à l'emprunt sur la liste des médias côté Membre.

- La méthode « __init__ » de la classe LoanForm permet la sélection de la catégorie dans la fonction « create_loan » et le filtrage des médias en fonction de la catégorie choisie.

- La méthode « clean_loan_date » permet de vérifier le format de la date et de rentrer une date antérieure à la date du jour.

- La fonction « return_loan » permet dans un premier temps de sélectionner un emprunteur via une liste déroulante.
  Lorsque l'emprunteur est sélectionné, apparait la liste des emprunts en cours pour celui-ci avec la date d'emprunt et à quelle date il doit faire le retour de l'emprunt au plus tard. Si l'id du membre sélectionné n'est pas reconnu alors une redirection vers la page erreur 404 sera faite.
  Si la demande de retour de l'emprunt est faite, un formulaire prérempli contenant le nom du média, la date d'emprunt et la date de retour prévue sera affiché. Pour valider le retour, il faudra entrer la date effective de celui-ci et confirmer le retour. Un message de succès apparaitra à la suite de cette confirmation.
  La validation du retour d'emprunt entrainera la mise à jour de la disponibilité du média. La liste des médias actualisée et le média sera à nouveau marqué comme disponible à l'emprunt et dans la liste des membres, l'emprunt en cours sera effacé.
  La liste des médias côté Membre sera également mise à jour.

  - La méthode « __init__ » de la classe ReturnLoanForm permet de préremplir le formulaire lors du retour d'un emprunt, après la sélection du membre.

  - La méthode « clean_effective_return_date » permet la validation de la date de retour effective lors de 'enregistrement du retour de la fonction « return_loan » en vérifiant le format de la date, que la date de retour sélectionnée soit bien postérieure à la date de l'emprunt et que le média n'a pas déjà été retourné.

- La méthode « save » permet l'enregistrement du retour de l'emprunt ainsi que la mise à jour de la date de retour du média en le rendant de nouveau disponible.

- La fonction « mediadelete » permet la suppression d'un média et à la validation de celui-ci, un message de succès s'affichera.

III- Stratégie de tests

Afin de garantir la fiabilité du code ainsi que la maintenabilité et la qualité de celui-ci, j'ai réalisé différents tests sur mes fonctions principales ainsi que des logs que je vais détailler.

Afin de fluidifier la gestion de mes tests, je les ai placés dans un dossier 'tests' et j'ai séparé les tests de mes models et les tests de mes views.
J'ai également créé un dossier permettant de notifier l'historique des logs.

Afin de pouvoir réaliser mes tests, je dois permettre l'accès à ma base de données et c'est le décorateur '@pytest.mark.django_db' qui le permet.

Test_models :

- Le « test_member_creation » permet de tester la création d'un membre en créant un membre test, de vérifier que celui-ci a bien été enregistré dans la base de données de test et que la méthode « __str__ » de la classe Member fonctionne correctement.

- Le « test_media_creation » permet de tester la création d'un média en créant un média test, de vérifier que celui-ci a bien été enregistré dans la base de données test, que la méthode « __str__ » de la classe Media fonctionne correctement et que le média est bien disponible.

- Le « test_loan_creation » permet de tester la création d'un emprunt et l'indisponibilité du média emprunté en vérifiant qu'un emprunt a bien été créé, que l'emprunteur et le média sont bien liés et que l'emprunt change l'état du média en le marquant comme indisponible.

- Le « test_borrowing_limit » permet de tester si la limite des 3 emprunts est atteinte en créant un membre et 4 médias tests puis en créant l'emprunts de 3 de ces médias par le membre test. On vérifie ensuite que lors de la création d'un quatrième emprunt pour ce membre, le message d'erreur attendu s'affiche grâce à 'ValueError'. Si la création de l'emprunt réussi alors le test sera mis en échec.

- Le « test_late_loans » permet de tester l'impossibilité pour un membre d'emprunter un nouveau média s'il n'a pas rendu un média avant la date prévue en créant un membre et un média test puis en lui attitrant l'emprunt sachant qu'il a emprunté le média test depuis 10j et qu'il ne l'a pas retourné. On vérifie qu'à la création d'un nouvel emprunt, le message d'erreur attendu s'affiche grâce à 'ValueError'. Si la création de l'emprunt réussi, le test sera mis en échec.

- Le « test_media_as_unavailable » permet de tester qu'un emprunt ne peut se réaliser si le média est marqué comme indisponible en créant un membre et un média test qui lui est noté indisponible. On vérifie qu'à la création d'un emprunt, le message d'erreur attendu s'affiche grâce à 'ValueError'. Si la création de l'emprunt réussi, le test sera mis en échec.

Test_views :

- Afin de pouvoir simuler des requêtes http de type GET ou POST dans mes tests views, j'utiliserai le paramètre 'client' qui est intégré à pytest-django.

- Le « test_add_member » permet de tester la création d'un membre en vérifiant qu'un utilisateur (pour moi, la bibliothécaire) puisse ajouter un membre en envoyant un formulaire valide, qu'ensuite la redirection vers une autre page se fait correctement puis que le membre est bien ajouté dans la base de données.
  Si le membre n'est pas créé ou que la redirection vers la page souhaitée ne se font pas, alors le test échouera.

- Le « test_member_update » permet de tester la mise à jour d'un membre en vérifiant que l'on peut mettre à jour le formulaire d'un membre existant, que la réponse POST est prise en compte et que les valeurs mises à jour sont bien enregistrées dans la base de données.
  Si les informations du membre ne sont pas correctement mises à jour alors le test échouera.

- Le « test_member_delete » permet de tester la suppression d'un membre en vérifiant qu'un membre qui existe puisse être supprimé via une requête POST, qu'après la suppression, la redirection vers la page souhaitée se fasse et que le membre n'existe plus dans la base de données.

- Le « test_add_book » permet de tester l'ajout d'un livre en vérifiant que l'ajout d'un livre peut se faire via une requête POST, qu'à la validation de l'ajout, la redirection vers la page souhaitée se fasse correctement et que le livre est bien présent dans la base de données.

- Le « test_media_delete » permet de tester la suppression d'un média en vérifiant qu'un média créé puisse être supprimé via une requête POST et qu'après la suppression la média n'existe plus dans la base de données et que la redirection vers la page souhaitée se fasse.

- Le « test_create_loan » permet de tester la création d'un emprunt en vérifiant que l'emprunt d'un média est bien réalisé par un membre, qu'à la suite de l'emprunt, la redirection vers la page souhaitée se fasse, que l'emprunt existe bien dans la base de données et que le média soit ensuite bien marqué comme indisponible à l'emprunt.

- Le « test_return_loan » permet de tester le retour d'un emprunt et sa disponibilité en vérifiant l'accès à la page de retour d'emprunt, que l'on peut sélectionner un membre et que l'on soit redirigé vers la liste des emprunts concernant le membre sélectionné auparavant, que la liste des emprunts s'affiche bien, que l'on peut enregistrer l'emprunt et que la redirection vers la page souhaitée se fasse.

- Le « test_access_home_librarian_unauthenticated » permet de tester l'accès à la page bibliothécaire en vérifiant qu'un membre ou personne n'ayant pas les identifiants ne puisse accéder à cette page, que si une tentative est faite, il sera redirigé vers la page de connexion et que si l'authentification réussi alors la redirection se fait bien vers la page demandée.

Les logs :

- La création d'un logger facilite le débogage, de tracer et surveiller l'exécution du code et d'enregistrer les erreurs d'où la création d'un dossier 'logs'.
  La création du 'logger' dans les views permet ensuite l'utilisation de différents messages de type debug, info, warning, error ou critical suivant le niveau.

- Dans la fonction « home_librarian », le « logger.info » permet d'enregistrer un message à chaque fois qu'une page est visitée.

- Dans la fonction « listmembers » :

  - Le « logger.debug » permet d'enregistrer un message indiquant le nombre de membres dans la base de données ce qui permet de comparer ce nombre au nombre réel et s'il y a discordance, permettre de résoudre plus facilement le problème.

  - Le « logger.error » permet d'enregistrer un message d'erreur s'il y a un problème lors de la récupération des membres dans la base de données lors de l'affichage de la liste des membres.

- Dans la fonction « addmember » :

  - Le « logger.info(soumission…) » permet d'enregistrer un message à chaque fois que la fonction « addmember » est appelée.

  - Le « logger.info(membre…) » permet d'enregistrer un message à chaque fois qu'un membre est ajouté avec succès.

  - Le « logger.error » permet d'enregistrer un message si une erreur se produit pendant l'ajout d'un membre.

  - Le « logger.warning » permet d'enregistrer un message d'avertissement lorsque le formulaire pour ajouter un membre est invalide.

  - Le « logger.info(affichage…) permet d'enregistrer un message à chaque fois qu'un formulaire vide pour l'ajout d'un membre est affiché.

- Dans la fonction « memberdelete » :

  - Le « logger.info(tentative…) permet d'enregistrer un message avant toute tentative de suppression d'un membre.

- Le « logger.info(membre…) permet d'enregistrer un message lorsque le membre a été supprimé avec succès.

- Le « logger.error » permet d'enregistrer un message si une erreur apparait lors de la suppression d'un membre.

➢ Dans la fonction « listmedia » :

- Le « logger.info » permet d'enregistrer un message chaque fois qu'on utilise la fonction « listmedia ».

- Le « logger.debug » permet d'enregistrer un message concernant le nombre de fois qu'une catégorie de médias est récupérée dans la base de données.

- Le « logger.error » permet d'enregistrer un message s'il y a un problème lors de la récupération d'un média pendant l'affichage de cette liste.

IV-Base de données et données tests

Comme mentionné dans les tests, j'ai créé des données tests pour permettre leurs exécutions.

Pour la base de données du projet, elle est dans le fichier db.sqlite3 qui se trouve dans le projet indiqué dans le lien GitHub :

https://github.com/Nathi33/my_mediatheque_project.git

V- Instructions d'exécution du programme

Voici les instructions afin de pouvoir exécuter le projet :

- Dans le terminal, cloner le nom du projet en entrant la commande :
  git clone https://github.com/Nathi33/my_mediatheque_project.git

- Aller dans le répertoire du projet en entrant :
  Cd my_mediatheque_project

- Vérifier que Python soit bien installer en tapant :
  python --version
  (si ce n'est pas le cas, télécharger Python via le site officiel :
  https://www.python.org/downloads/)

- Créer un environnement virtuel en tapant :
  python -m venv env

- Activer l'environnement virtuel :
  .\env\Scripts\activate

- Vérifier que Django soit bien installé en tapant :
  python -m django --version
  (si ce n'est pas le cas, télécharger Django en tapant : pip install django)

- Faire la migration de la base de données en tapant :
  python manage.py makemigrations
  Puis :
  python manage.py migrate

- Lancer le projet en tapant :
  python manage.py runserver

- Pour accéder à l'espace bibliothécaire, entrer les identifiants :
  nom d'utilisateur : bibliothecaire
  mot de passe : biblio33