Dossier du projet "Créer un logiciel de gestion de médiathèque"

Table des matières

A. PRESENTATION

1. Contexte

La médiathèque "Notre livre, notre média" souhaite moderniser son système de gestion interne en l'informatisant. Cette gestion comprend celles des médias, des membres et des emprunts.

2. Les objectifs du site

Le site doit permettre au public de consulter la liste des médias de la médiathèque.
Une partie accessible uniquement aux bibliothécaires doit leur permettre de gérer les médias et les membres c'est-à-dire qu'ils doivent pouvoir en créer, en consulter la liste, les modifier et en supprimer. Ils doivent aussi pouvoir gérer les emprunts c'est-à-dire en créer, consulter la liste et effectuer des retours d'emprunts.

3. Les cibles

L'accessibilité est un point majeur car la plateforme doit être accessible à toute personne quelque soit leur âge ou leur situation de handicap le cas échéant. L'application devra donc être conforme aux règles énoncées dans la WCAG 2.1. Le site doit être intuitif et facile à prendre en main. Cela en particulier pour la partie réservée au public pour la consultation des médias. Quant aux bibliothécaires, ils recevront une formation pour la compréhension et l'utilisation de la partie de la plateforme qui leur est réservée, elle doit néanmoins rester la plus intuitive et ergonomique possible.

4. Périmètre du projet

Cette plateforme sera développée à l'aide du framework Django qui permet de gérer l'interface front-end de l'application ainsi que l'interface back-end.
De cette façon, les médias, les membres et les emprunts seront enregistrés dans une base de données relationnelle gérée par le SGBD MariaDB afin de pouvoir sécuriser l'accès à la base de données.

Le projet se limite au développement de l'interface et des fonctionnalités métiers qui découlent de la gestion d'une médiathèque. Ainsi les graphismes, styles et mises en forme ne seront pas abordés dans ce projet.

Cette plateforme sera composée de deux applications, une application membre et une application bibliothécaire. La partie de l'application membre est accessible à tout public alors que la partie bibliothécaire est strictement réservée aux bibliothécaires. Il faut donc mettre en place une interface de connexion avec un nom d'utilisateur et un mot de passe.

En cas de tentative d'accès d'un utilisateur non connecté à une URL nécessitant d'être connecté, l'utilisateur sera alors redirigé vers la page de connexion.

La médiathèque propose à ses membres d'emprunter des médias comme les livres, les CD ou les DVD mais pas les jeux de plateaux.

Un membre peut avoir au maximum 3 emprunts simultanément.

Les emprunts doivent être retournés à la médiathèque au bout d'une semaine sinon l'emprunteur est considéré comme bloqué et ne peut pas emprunter de nouveau un média. Le membre n'est plus bloqué à partir du moment où il a retourné le média pour lequel il était en retard.

B. GRAPHISME ET ERGONOMIE

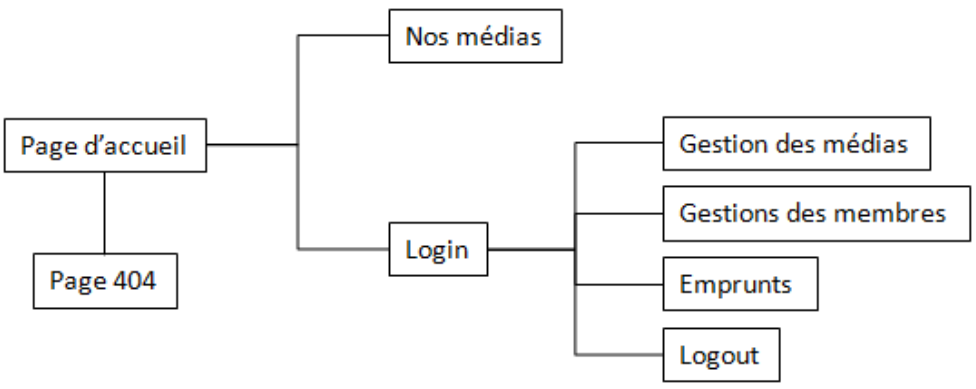
1. *La charte graphique*

La charte graphique n'est pas encore mise en place par lé client. Seule la favicon a été choisi par le client :

2. *Maquettage*

Le client préfère que les fonctionnalités soient d'abord développées pour s'assurer que leur utilisation lui convient avant de les intégrer dans des maquettes. Le maquettage sera donc réalisé par la suite et prendra en compte les différentes fonctionnalités mises en place au cours de ce projet.

3. *Arborescence du site web*



C. SPECIFICITES ET LIVRABLE

1. *Le contenu du site*

a. Page d'accueil

En tête

L'en-tête doit contenir la barre de navigation, elle est différente selon que l'utilisateur est connecté en tant que bibliothécaire ou non.

Lorsqu'on n'est pas connecté :

- Accueil : permet de revenir à la page d'accueil
- Nos médias : permet d'accéder à la page de consultation de l'ensemble des médias de la médiathèque
- Connexion : permet de se loguer à l'application en tant que bibliothécaire

Lorsqu'on est connecté en tant que bibliothécaire:

- Accueil
- Nos médias
- Bibliothécaire :
    - Gestion des médias : permet d'accéder à la page de gestion des médias
    - Gestion des membres : permet d'accéder à la page de gestion des membres
    - Emprunts : permet d'accéder à la page de gestion des emprunts
- Déconnexion : cliquer dessus permet de se déloguer de l'application et redirige l'utilisateur vers la page de connexion

Par défaut, cet en-tête sera présent sur toutes les pages de la plateforme.

Corps de la page

Pour le moment simplement un titre :
Bienvenue à la médiathèque 'Notre livre, notre média'

Il y aura un texte de présentation de la médiathèque qui sera rajouté par la suite.

Pied de page

Pas de pieds de page pour le moment (et ce sur l'ensemble des pages) car il ne comprend pas de fonctionnalités liées à la gestion de la médiathèque, il sera rajouté par la suite sur la base du maquettage.

b. Page Nos médias

Cette page comprend la liste de tous les médias sous la forme d'un unique tableau avec plusieurs colonnes :

- Identifiant du média
- Type du média
- Titre du média
- Auteur / Réalisateur / Artiste / Créateur du média
- Disponibilité du média à l'emprunt
- Date maximale du retour

Il doit aussi y avoir un filtre permettant de choisir le type de média que l'on souhaite faire apparaître dans la liste. Ce filtre comprend les choix suivants :

- Tous les médias
- Les livres
- Les CD
- Les DVD
- Les jeux de société

Une barre de recherche permet de filtrer les différentes lignes du tableau. Pour chaque médias filtré, le contenu de la barre de recherche doit correspondre de façon exacte à du contenu présent dans la ligne du tableau correspondante.

c. Page Connexion

Elle doit contenir un formulaire avec les champs :

- Identifiant
- Password

Et un bouton 'Se connecter'

En cas de non reconnaissance de l'utilisateur, le message suivant s'affiche :
"Nom d'utilisateur ou mot de passe incorrect."

d. Page Gestion des médias

Dans cette page il doit y avoir 4 parties :

- Ajouter un média à la base de données
- Supprimer un média de la base de données
- Modifier un média de la base de données
- Liste des médias de la médiathèque

La partie 'Ajouter un média à la base de données' contient un formulaire permettant de sélectionner le type de média à ajouter (Livre, CD, DVD, Jeu de société). Il y a aussi un champ pour le nom du média et un autre champ pour le nom de l'auteur, artiste, réalisateur ou créateur. Les trois champs sont requis pour que le formulaire soit valide.
Après envoi du formulaire un message de succès ou d'échec de la requête doit informer le bibliothécaire de la situation.

La partie 'Supprimer un média de la base de données' contient un formulaire où il est possible de sélectionner le type de média à supprimer avec deux choix possibles :

- Livre / CD/ DVD
- Jeu de société

Un second champ permet de renseigner l'identifiant du média à supprimer. Une fois ces deux informations renseignées, le nom du média à supprimer apparaît de façon dynamique pour que le bibliothécaire puisse s'assurer qu'il s'agit du média correspondant.
Tous les champs sont requis pour valider l'envoie du formulaire.
Après envoi du formulaire un message de succès ou d'échec de la requête doit informer le bibliothécaire de la situation.

La partie 'Modifier un média de la base de données' contient un formulaire avec 4 champs :

- Type de média : deux choix possibles :
  - Livre / CD / DVD
  - Jeu de société
- Identifiant du média
- Nom du média
- Auteur / Artiste / Réalisateur / Créateur du média

Une fois les champs 'Type de média' et 'Identifiant' renseigné, le nom du média et l'auteur correspondant apparaissent dans les champs de façon dynamique. Il ne reste plus qu'à modifier les champs voulus (nom et/ou auteur) et à envoyer le formulaire.
Après envoi du formulaire un message de succès ou d'échec de la requête doit informer le bibliothécaire de la situation.

La liste des médias de la médiathèque correspond au même tableau et aux mêmes filtres que ceux présents sur la page 'Nos médias'

e. Page Gestion des membres

Dans cette page, il doit y avoir 4 parties :

- Créer un membre
- Modifier un membre
- Supprimer un membre
- La liste des membres

La partie 'Créer un membre' contient un formulaire avec deux champs :

- Nom
- Prénom

Ceux deux champs sont requis pour l'envoi du formulaire.
Après envoi du formulaire un message de succès ou d'échec de la requête doit informer le bibliothécaire de la situation.

La partie 'Modifier un membre' contient un formulaire avec 4 champs :

- Identifiant
- Nom
- Prénom
- Le membre est interdit d'emprunt (sélection oui/non)

Dès que l'identifiant est renseigné, les informations correspondantes du membre apparaissent de façon dynamique et il est possible de les modifier.
Il n'est cependant pas possible de déclarer un membre comme étant non interdit d'emprunt si ce dernier possède toujours un emprunt en retard à son nom.
Après envoi du formulaire un message de succès ou d'échec de la requête doit informer le bibliothécaire de la situation.

La partie 'Supprimer un membre' contient un formulaire avec un champ 'Identifiant'. Une fois l'identifiant renseigné, les nom et prénom du membre s'affichent de façon dynamique afin que le bibliothécaire puisse s'assurer qu'il s'agit bien du bon membre.
Après envoi du formulaire un message de succès ou d'échec de la requête doit informer le bibliothécaire de la situation.

f. Page Emprunts

Cette page contient 3 parties :

- Nouvel emprunt
- Retour d'emprunt
- Liste des emprunts

La partie 'Nouvel emprunt' contient un formulaire avec deux champs :

- Identifiant du média
- Identifiant du membre

Lorsque le champ 'Identifiant du média' est rempli, le nom du média apparaît dynamiquement pour que le bibliothécaire s'assure qu'il s'agit bien du bon média. Même procédé pour le champ 'Identifiant du membre'.
Si le membre est interdit d'emprunt, un message d'erreur dynamique s'affiche pour informer le bibliothécaire. Si le formulaire est tout de même envoyé, l'emprunt n'est pas enregistré et un message d'erreur s'affiche. Même procédé si le membre possède déjà 3 emprunts à son nom.
Les deux champs sont requis pour que le formulaire soit valide.

Après envoi du formulaire un message de succès ou d'échec de la requête doit informer le bibliothécaire de la situation.

La partie 'Retour d'emprunt' contient un formulaire avec 2 champs :

- Identifiant du membre
- Média à retourner

Le bibliothécaire renseigne l'identifiant du membre et la liste des emprunts s'affiche dynamiquement dans une sélection. Chaque choix contient l'identifiant, le nom, l'auteur et le type du média concerné.
Si le membre est en retard pour un emprunt, un message s'affiche pour informer le bibliothécaire qui peut alors informer le membre de la situation.
Si le membre est bloqué uniquement à cause de l'emprunt qu'il retourne alors il est débloqué après enregistrement du retour.
Les deux champs sont requis.
Après envoi du formulaire un message de succès ou d'échec de la requête doit informer le bibliothécaire de la situation.

g. Page 404

Cette page s'affiche si l'url renseigné ne renvoi aucune page HTML, l'utilisateur est informé que cette page est introuvable et il est invité à retourner sur la page d'accueil via un lien cliquable.

2. Spécificités techniques

La plateforme est développée à l'aide framework Django et on utilise le SGBD MariaDB (via Xampp) pour générer et gérer la base de données relationnelle afin de pouvoir la sécuriser l'accès par un identifiant et un mot de passe.

On utilisera Git et GitHub pour versionner le code source du projet.

a. Reprise du code existant

Concernant le code fourni, chaque nom de classe doit respecter le PascalCase et il doit y avoir des parenthèses() après chaque nom de classe sans espace entre le nom et les parenthèses. Ces parenthèses sont suivies de deux point ':' sans espace entre les parenthèse et les deux point : NomDeMaClasse():
Dans les classes, méthodes et d'une façon générale, l'indentation doit toujours être la même dans le code et de préférence de 4 espaces. Les noms de variables et de méthodes doivent être écrites en snake_case.

Globalement les mêmes classes ont été conservées. Cependant la classe BorrowableMedia() a été ajoutée :

```python
class BorrowableMedia(models.Model):  25 usages  fb.lb
    name = models.fields.CharField(max_length=150)
    borrowing_date = models.fields.DateField(null=True)
    return_date = models.fields.DateField(null=True)
    is_available = models.fields.BooleanField(default=True)
    borrower = models.ForeignKey(
        Member,
        on_delete=models.SET_NULL,
        null=True
    )
```

Les classes Book(), Cd() et Dvd() héritent de la classe BorrowableMedia() ce qui permet de déclarer dans ces classes uniquement le type du média et son auteur / artiste ou directeur :

```python
class Book(BorrowableMedia):  15 usages  fb.lb
    media_type = models.fields.CharField(max_length=150, default='livre')
    author = models.fields.CharField(max_length=150)


class Cd(BorrowableMedia):  2 usages  fb.lb
    media_type = models.fields.CharField(max_length=150, default='cd')
    artist = models.fields.CharField(max_length=150)


class Dvd(BorrowableMedia):  5 usages  fb.lb
    media_type = models.fields.CharField(max_length=150, default='dvd')
    director = models.fields.CharField(max_length=150)
```

Quant à la classe ParlourGame() elle n'hérite pas de la classe BorrowableMedia() car les jeux de société ne sont pas empruntables :

```python
class ParlourGame(models.Model):  13 usages  fb.lb
    media_type = models.fields.CharField(max_length=150, default='jeu de société')
    name = models.fields.CharField(max_length=150)
    creator = models.fields.CharField(max_length=150)
```

Il reste maintenant à définir la classe des membres :

```
class Member(models.Model):  26 usages  fb.lb
    first_name = models.fields.CharField(max_length=30)
    last_name = models.fields.CharField(max_length=30)
    nb_current_borrowings = models.fields.IntegerField(
        default = 0,
        validators = [
            MaxValueValidator(3),
            MinValueValidator(0)
        ]
    )
    is_blocked = models.fields.BooleanField(default=False)
```

Dans la classe BorrowableMedia(), borrower correspond à un champs de clé étrangère permettant de faire la relation entre un média empruntable et un membre qui se trouve alors être l'emprunteur du média en question.

b. Login, logout et restriction de views aux bibliothécaires

Login

Pour le login, il a tout d'abord fallut définir l'url faisant appel à la view générant le template de la page de connexion :

```
urlpatterns = [
   path('connexion/', auth_views.LoginView.as_view(template_name="login.html",
authentication_form=LoginForm), name='login')
]
```

Dans urlpatterns, on définit 'connexion/' comme étant le chemin permettant l'accès à la page de connexion. En accédant à cet url, on utilise auth_views.LoginView.as_view() (provenant du package django.contrib.auth) qui va utiliser un formulaire de connexion par défaut et accéder au template login.html que l'on va créer.

On créé donc le template login.html qui contient les éléments du formulaire qui nous intéressent. On y définit les éléments un à un pour que les messages d'erreur par défaut de Django n'apparaissent pas et on personnalise notre message d'erreur à l'aide d'un bloc if :

```
<form action="" method="post">
    {% csrf_token %}
    <div>
        {{ form.username.label_tag }}
        {{ form.username }}
    </div>
    <div>
        {{ form.password.label_tag }}
        {{ form.password }}
    </div>
    <button type="submit">Se connecter</button>
</form>
```

```
{% if form.errors %}
<p style="color: red;">Nom d'utilisateur ou mot de passe incorrect.</p>
{% endif %}
```

Après avoir créé un utilisateur dans la base de données il est maintenant possible de se connecter à l'application.
Une fois l'utilisateur connecté, il doit être redirigé vers la page d'accueil, pour cela on va dans settings.py et on définit :

```
LOGIN_REDIRECT_URL = '/'
```

Restrictions des views aux utilisateurs connectés

Pour qu'il y ait un intérêt à ce que l'utilisateur soit connecté, il faut que certaines views ne soient accessibles que si l'utilisateur est connecté. Pour cela il faut ajouter le décorateur @login_required (importé de le package django.contrib.auth.decorators) à la ligne précédent la définition d'une fonction retournant un template non accessible à un utilisateur non connecté :

```
@login_required  1 usage   fb.lb
def members_management(request):...
```

Si jamais un utilisateur non connecté tente d'accéder à cette view, il faut qu'il soit redirigé vers la page de connexion et pour cela on va dans settings.py et on définit :

```
LOGIN_URL = reverse_lazy('login')
```

Logout

Pour le logout, on définit dans urlpatterns de urls.py le chemin suivant :

```
path('deconnexion/', auth_views.LogoutView.as_view(next_page='login'), name='logout'),
```

De cette façon le chemin 'deconnexion/' fait appel à la view générée par auth_views.LogoutView.as_view() (importé du package django.contrib.auth) qui va gérer la

déconnexion de l'utilisateur et le paramètre `next_page='login'` permet de rediriger l'utilisateur vers la page de connexion après avoir été déconnecté.

c. Interface CRUD des médias

Create

Le formulaire pour l'ajout d'un média contient :

- Une balise <select> pour choisir le type de média
- Un <input> de type texte pour renseigner le nom du média
- Un <input> de type texte pour renseigner le nom de l'auteur / artiste / réalisateur / créateur

Une fois les informations renseignées on clique sur 'Enregistrer' ce qui envoie le formulaire au server par la méthode POST.
Le serveur reçoit une requête provenant de l'url http://localhost:8000/bibliothecaire/gestion-des-medias/, analyse le chemin /bibliothecaire/gestion-des-medias/. Tout d'abord dans le urls.py de la mediatheque_app c'est la partie /bibliothecaire/ qui est reconnu ce qui redirige vers le fichier urls.py de l'application librarians_app qui reconnaît alors la partie /gestion-des-medias/ via la variable urlspatterns où un path indique qu'il faut exécuter la méthode media_management() de la views de librarians_app.

Cette méthode media_management() prend en argument la requête donc on a media_management(request). On teste si la méthode est GET ou POST, ici comme c'est POST et qu'on trouve 'submit_create_media' dans le POST (placé dans l'attribut 'name' de l'input de type submit afin de distinguer chacun des formulaires de la page) alors on lance la méthode create_media().

Dans create_media() on récupère ce qui a été posté dans la requête :

```
create_media_request = CreateMedia(request.POST)
```

Il faut d'abord vérifier que le formulaire est valide avec :

```
if create_media_request.is_valid():
```

Sinon on renvoie la page avec le formulaire en l'état et django informe l'utilisateur que les champs sont requis.
(il ne faut pas oublier le novalidate dans la balise <form> pour préciser à Django que le formulaire doit être validé avant de pouvoir en extraire les valeurs)

Puis selon le type de média, on créé un nouveau média correspondant et on y définit les valeurs des différents attributs du média puis on le sauvegarde. Pour un livre cela donne :

```
book = Book()
book.name = create_media_request.cleaned_data['name']
book.author = create_media_request.cleaned_data['author']
book.save()
```

En suite on redirige l'utilisateur vers cette même page en y précisant le paramètre success dans l'url pour que côté client il puisse y avoir un affichage d'un message de succès qui s'affiche grâce à :

```
{% if request.GET.success == 'create_media' %}
<div class="success">Nouveau média enregistré</div>
{% endif %}
```

Read

Lorsque l'utilisateur tente d'accéder au serveur avec l'url http://localhost:8000/bibliothecaire/gestion-des-medias/ par la méthode GET alors l'url est reconnu de la même façon que pour le processus 'Create' ci-dessus et exécute la méthode media_management() du fichiers views.py de Librarians_app.

Tout d'abord, tous les médias empruntables sont récupérés puis les jeux de sociétés aussi

```
medias = BorrowableMedia.objects.all()
parlour_games = ParlourGame.objects.all()
```

Pour tous les médias dans medias je défini les propriétés media.type et media.author pour que les valeurs soit accessibles de la même façon quelque soit le type de média.

Puis je les transmets dans le paramètre de context de render :

```
return render(request, template_name: 'mediaManagement.html', context: {
    'create_media': CreateMedia(),
    'delete_media': DeleteMedia(),
    'update_media': UpdateMedia(),
    'medias': medias,
    'parlour_games': parlour_games
})
```

Ensuite dans le template medias et parlour_games sont utilisés pour que côté client, les informations relatives aux médias s'affichent dans un tableau. Pour medias, cela donne :

```
{% for media in medias %}
<tr class="{{ media.type|slugify }}">
    <td>{{ media.pk }}</td>
    <td>{{ media.type }}</td>
    <td>{{ media.name }}</td>
    <td>{{ media.author }}</td>
    <td>{% if media.is_available %}Oui{% else %}Non{% endif %}</td>
    <td>{% if not media.is_available %}{{ media.return_date }}{% else %}{% endif %}</td>
</tr>
{% endfor %}
```

Update

Le formulaire de modification d'un média contient quatre champs :

- Le type du média
- L'identifiant
- Le nom du média
- L'auteur / artiste / réalisateur / créateur du média

Le type du média ne contient que deux choix possibles :

- Livre / CD / DVD
- Jeu de société

J'ai choisi cela car les livres, CD et DVD héritant tous de la classe model BorrowableMedia ils ont tous des identifiants différents ce qui permet de les regrouper tous ensemble, l'identifiant peut les distinguer les uns des autres.
Par contre les jeux de société n'héritent pas de la classe model BorrowableMedia donc ils peuvent avoir des identifiants communs à un livre, CD ou DVD d'où l'importance de les distinguer.

Une fois les quatre champs renseignés, on peut envoyer le formulaire côté serveur via la méthode POST. La requête est reconnue de la même façon que dans les deux exemples ci-dessus.

La méthode media_management() est donc exécutée et comme la requête de la méthode est POST et qu'elle contient 'submit-update-media' (placé dans l'attribut 'name' de l'input de type submit) alors c'est la méthode update_media() qui est exécutée.

Dans cette méthode, on récupère le formulaire, on vérifie qu'il est valide puis selon le type du média on récupère le média en question et on modifie ses attributs par les valeurs renseignées dans les champs du nom et de l'auteur/artiste/réalisateur/créateur puis on sauvegarde le média. Pour la mise à jour des valeurs des propriétés d'un jeu de société cela donne :

```
id = update_media_request.cleaned_data['id']
parlour_game = ParlourGame.objects.get(pk=id)
parlour_game.name = update_media_request.cleaned_data['name']
parlour_game.creator = update_media_request.cleaned_data['author']
parlour_game.save()
```

Ensuite l'utilisateur est redirigé vers cette même page avec un message de succès (de la même façon que la partie create ci-dessus) :

```
return redirect(reverse('media_management') + '?success=update_media#update-media-section')
```

Delete

Pour le formulaire de la suppression d'un média, il y a deux champs :

- Le type du média
- L'identifiant du média

Pour les mêmes raisons que précisées ci-dessus, il y a deux choix pour le type du média :

- Livre / CD / DVD
- Jeu de société

Une fois le formulaire envoyé par la méthode POST, la méthode media_mangement() est de nouveau exécuté et comme la méthode de la requête est POST et qu'il y a 'submit-delete-media' dedans alors c'est la fonction delete_media() qui est exécutée.

Dans cette méthode, on récupère le formulaire envoyé, on vérifie qu'il est valide puis selon le type du média on récupère le media et exécute la méthode .delete() pour le supprimer. Puis on redirige l'utilisateur sur cette même page avec un message de succès. Pour un média de type livre, Cd ou DVD, cela donne (sans rencontrer d'erreur) :

```
def delete_media(request, medias, parlour_games):  1 usage   fb.lb
    delete_media_request = DeleteMedia(request.POST)
    if delete_media_request.is_valid():
        media_type = delete_media_request.cleaned_data['media_type']
        if media_type == 'media':
            try:
                media = BorrowableMedia.objects.get(pk=delete_media_request.cleaned_data['id'])
                media.delete()
                return redirect(reverse('media_management') + '?success=delete_media#delete-media-section')
```

d. Les tests

Pour les tests j'ai voulu tester le bon déroulement du login, du logout, la création / l'affichage / modification / suppression d'un média, d'un membre et la création / le retour d'un emprunt.

Pour la connexion j'ai testé avec des identifiants valides puis avec des identifiants invalides pour m'assurer qu'une personne avec de mauvais identifiants ne puissent pas se connecter.

Pour les nouveaux emprunts j'ai testé aussi le cas où l'utilisateur possède 3 emprunts à son nom et le cas où l'un des emprunts de l'utilisateur est en retard sur le retour.

Exemple du login

Pour le test je vérifie d'abord qu'il est possible d'accéder à la page de connexion grâce à la méthode reverse() et client.get() puis je vérifie que le code de la réponse est bien 200 et qu'il y a le mot "Connexion", "Nom d'utilisateur" et "Mot de passe" dans le contenu de la réponse puisque ce sont des éléments spécifiques de la page de connexion.

```python
@pytest.mark.django_db  fb.lb *
def test_login_view(client):
    # Test getting login page
    url = reverse('login')
    response = client.get(url)
    assert response.status_code == 200
    assert b"Connexion" and b"Nom d'utilisateur" and b"Mot de passe" in response.content
```

Pour créer un utilisateur valide, il faut que les tests aient accès à la base de données, pour cela il faut utiliser le décorateur @pytest.mark.django_db sur la ligne précédent la définition de la méthode. De cette façon on peut créer un utilisateur :

```python
# Test of connexion with valid username and password
user = User.objects.create_user(username='testunsername', password='testpassword')
response = client.post(url, {
    'username': 'testunsername',
    'password': 'testpassword'
})
```

Après connexion l'utilisateur est rédirigé vers la page d'accueil, on peut donc vérifier que le code de la réponse est bien 302 et que le chemin de l'url de redirection est bien celui de la page d'accueil '/'

```python
assert response.status_code == 302
assert response.url == '/'
```

Pour tester un login avec des identifiants incorrects, on utilise des identifiants que l'on a sur aucun utilisateur, on vérifie que cela retourne bien la même page du formulaire de connexion avec le code 200 et que la page contient bien le message d'erreur "Nom d'utilisateur ou mot de passe incorrect."

```python
# Test of connexion with invalid username and password
response = client.post(url, {
    'username': 'invalidunsername',
    'password': 'invalidpassword'
})
assert response.status_code == 200
assert b"Nom d'utilisateur ou mot de passe incorrect." in response.content
```

Exemple d'un nouvel emprunt

Pour un nouvel emprunt, je créé un utilisateur valide car il doit pouvoir avoir accès à l'application librarians_app pour enregistrer un nouvel emprunt puis je me connecte pour avoir accès à l'application librarians_app.
Il faut ensuite créer un membre, un livre et un dvd puis je vérifie que le livre est bien disponible et qu'il n'y a pas d'emprunteur pour être sur que la création s'est bien déroulée :

```python
@pytest.mark.django_db  fb.lb
def test_new_borrowing(client):
    # Connexion of librarian and creation of a member, a book and a dvd
    user = User.objects.create_user(username='testuser', password='testpassword')
    client.login(username='testuser', password='testpassword')
    member = Member.objects.create(first_name='Prénom Membre', last_name='Nom Membre')
    book = Book.objects.create(name='Livre test', author='Auteur test')
    dvd = Dvd.objects.create(name='DVD test', director='Réalisateur test')
    assert book.is_available == True
    assert book.borrower is None
```

Ensuite je simule un emprunt valide et je vérifie que le code de la réponse est bien 302 puisqu'il s'agit d'une redirection, que le membre est bien l'emprunteur du livre et que le livre n'est plus disponible :

```python
# Test of a valid new borrowing creation
response = client.post(reverse('borrowings'), {
    'submit_borrowing': "Enregistrer l'emprunt",
    'media_id': book.id,
    'member_id': member.id
})
book_borrowed = Book.objects.get(pk=book.id)
assert response.status_code == 302
assert book_borrowed.borrower == member
assert book_borrowed.is_available == False
```

Ensuite je teste le cas où l'emprunteur à déjà trois emprunts à son nom, je modifie donc la valeur de member.nb_current_borrowings à 3 et je simule un emprunt avec le DVD puis je vérifie que le code de la réponse est bien 200, que la page contient bien le message d'erreur, que le DVD n'a pas d'emprunteur et qu'il est toujours disponible :

```python
# Test that member who have 3 currents borrowings can't have a new one
response = client.post(reverse('borrowings'), {
    'submit_borrowing': "Enregistrer l'emprunt",
    'media_id': dvd.id,
    'member_id': member.id
}, follow=True)
dvd_no_borrowed = Dvd.objects.get(pk=dvd.id)
assert response.status_code == 200
assert "Ce membre ne peut pas emprunter car il a déjà 3 emprunts à son nom" in response.content.decode()
assert dvd_no_borrowed.borrower is None
assert dvd_no_borrowed.is_available == True
```

Enfin je teste le cas où l'emprunteur est en retard sur l'un de ses emprunts. Je prends tout d'abord soin de remettre son nombre d'emprunt à 1. Puis je modifie la date de retour d'emprunt pour qu'elle soit antérieure à la date du jour. Je peux alors simuler l'emprunt du DVD et vérifier que le code de la réponse est bien 200, que la page contient bien le message d'erreur correspondant, que le membre est alors considéré comme bloqué, que le DVD n'a pas d'emprunteur et qu'il est toujours disponible :

```
# Update member's number of borrowing at 1
member.nb_current_borrowings = 1
member.save()

# Update return date to block member
book_borrowed.return_date -= timedelta(days=8)
book_borrowed.save()

# Test at the time that late member is blocked and can't have a new borrowing
response = client.post(reverse('borrowings'), {
    'submit_borrowing': "Enregistrer l'emprunt",
    'media_id': dvd.id,
    'member_id': member.id
}, follow=True)
member_blocked = Member.objects.get(pk=member.id)
assert response.status_code == 200
assert b"Ce membre ne peut pas emprunter car il est interdit d'emprunt" in response.content
assert member_blocked.is_blocked == True
assert dvd_no_borrowed.borrower is None
assert dvd_no_borrowed.is_available == True
```

e. Les fixtures

Pour les fixtures j'ai créé trois fichiers :

- medias.json
- members.json
- users.json

J'ai initialement créé ces données avec mon application puis je les ai exportées au format json. Pour les médias, cela donne avec la commande :
python -X utf8 manage.py dumpdata librarians_app.borrowablemedia librarians_app.book librarians_app.cd librarians_app.dvd librarians_app.parlourgame --indent 4 > medias.json

Ensuite pour les utiliser il faut d'abord créer la base de données avec chacune des tables puis on peut alors importer les données :
python manage.py loaddata librarians_pp/fixtures/medias.json

Dans ces médias j'ai créé au moins un média de chaque type. J'ai aussi créé plusieurs membres avec lesquels j'ai simulé des emprunts. Un des membres (Jacques Dupont) est en retard sur l'un de ces emprunts et il en possède deux en cours, il est donc possible de vérifier qu'il ne peut pas effectuer un nouvel emprunt. Un second membre (Johnny Smith) possède 3 emprunts en cours, il est donc possible de vérifier qu'il ne peut pas faire de nouvel emprunt.

Quant à la fixture users.json, elle permet de créer un super utilisateur qui a accès à la partie admin de l'application django et un utilisateur qui à la possibilité d'accéder à la partie bibliothécaire de l'application pour pouvoir simuler les emprunts notamment (cf fin du Readme.md pour les identifiants et mot de passe associés).

f. Exemple d'utilisation de filtres pour la recherche de médias

J'ai décidé de mettre en place des filtres pour faciliter la recherche d'un média dans la liste des médias. Le premier filtre permet de sélectionner le type de média recherché et le second correspond à une recherche textuelle.

La liste des médias étant entièrement chargée au chargement de la page, j'ai décidé de réaliser ce tri uniquement en javascript côté client pour ne pas charger le serveur inutilement et pour conserver un maximum de fluidité côté client.

Sélection du type de média

Tout d'abord dans mon template mediasList.html j'ai ajouté une classe à chacune des lignes contenant un média, le nom de cette classe étant le type de média :

```
<tr class="{{ media.type|slugify }}">
```

J'ai aussi mis une balise <select> avec les différentes options correspondant aux différents types de médias :

```
<select name="media_filter" id="media-filter">
    <option value="all">Tous les médias</option>
    <option value="livre">Les livres</option>
    <option value="cd">Les CD</option>
    <option value="dvd">Les DVD</option>
    <option value="jeu-de-societe">Les jeux de société</option>
</select>
```

Ensuite côté javascript j'ai ajouté un eventlistener sur la balise <select> à chaque changement d'option. Je récupère alors tous les éléments <tr> et j'ajoute la classe hidden-select aux éléments <tr> dont la classe est différente de celle de la valeur de l'option sélectionné en prenant soin de ne pas prendre en compte la première ligne du tableau :

```
// Add event listener on select element which filter each line by their type
const selectElement = document.getElementById('media-filter');
selectElement.addEventListener("change", (event) => {
    const optionSelected = event.target.value;
    const allTrElements = document.querySelectorAll('tr');
    for (const tr of allTrElements) {
        tr.classList.remove('hidden-select');
        if (tr.classList.length == 0) {
            continue;
        } else if (optionSelected == 'all') {
            continue
        } else if (tr.classList.contains(optionSelected)) {
            continue
        } else {
            tr.classList.add('hidden-select');
        }
    }
});
```

Comme j'utilise un fichier javascript contenu dans le dossier static au chargement de ce template, il faut charger les fichiers static avec {% load static %} en haut du template après {% extends 'base.html' %} :

```
{% extends 'base.html' %}
{% load static %}
```

En bas du template, on peut ajouter le block contenant la balise <script>

```
{% block script_medias_list %}
<script src="{% static 'js/medias_list.js' %}" defer></script>
{% endblock %}
```

Recherche textuelle

Concernant la recherche textuelle, j'ai mis un <input> de type texte dans mon template pour avoir une barre de recherche :

```
<label for="media-search-bar">Rechercher</label>
<input type="text" name="media_search_bar" id="media-search-bar">
```

Côté javascript, j'ai ajouté un eventlistener sur l'élément <input> pour chaque changement de caractère. J'ai mis en place un compte à rebours de façon à ce que chaque changement de caractère inférieur à 500 millisecondes ne déclenche pas la fonction qui suit. Lorsque le temps de changement de caractère est supérieur à 500 millisecondes alors je récupère tous les éléments <tr> et j'ajoute la classe hidden-search aux lignes dont le contenu de la barre de recherche n'est pas inclus dans le contenu de la ligne. Je prends soin de ne pas prendre en compte la première ligne et de mettre en

lowercase la valeur de la barre de recherche ainsi que le contenu de la ligne pour ne pas prendre en compte la casse dans la recherche.

```
// Add event listener on search bar input which filter each line by their content
const inputElement = document.getElementById('media-search-bar');
let debounceTimeout;

inputElement.addEventListener("input", (event) => {
    clearTimeout(debounceTimeout);
    debounceTimeout = setTimeout(() => {
        const searchValue = event.target.value.toLowerCase();
        const allTrElements = document.querySelectorAll('tr');
        for (const tr of allTrElements) {
            tr.classList.remove('hidden-search');
            if (tr.classList.length == 0) {
                continue;
            } else if (!tr.innerText.toLowerCase().includes(searchValue)) {
                tr.classList.add('hidden-search');
            }
        }
    }, 500)
});
```

g. Exemple de l'affichage dynamique de données dans un formulaire

Prenons le formulaire de modification d'un média pour l'exemple.
Ce formulaire contient :

- Une balise <select> avec les options suivantes :
  - Livre / CD / DVD (valeur : media)
  - Jeu de société (valeur : parlour_game)
- Un <input> de type texte pour l'identifiant du média
- Un <input> de type texte pour le nom du média
- Un <input> de type texte pour l'auteur / artiste / réalisateur ou créateur du média

J'ai décidé de regrouper livre, CD et DVD dans une même option de la sélection car ils héritent de la classe BorrowableMedia et ont donc tous des identifiant (primary_key) différents. Quant aux jeux de société, ils n'héritent pas de cette classe, par conséquent ils ont leur propre identifiant qui peuvent être communs avec ceux de livre, CD et DVD d'où l'importance de faire la distinction.

Dans ce formulaire, il suffit de renseigner le type du média à changer et de renseigner son identifiant. Il y a un eventlistener côté javascript sur l'input de l'identifiant. A chaque changement de la valeur de l'identifiant, un requête fetch est lancé à ce chemin :

```
fetch(`/bibliothecaire/get-media-details-management/?media_id=${mediaId}&media_type=${mediaType}`)
```

On peut voir dans le chemin ${mediaId} et ${mediaType} qui permettent de transmettre les valeurs des variables correspondantes à travers le chemin.
Le chemin get-media-details-management/ est reconnu par urls.py :

```
path('get-media-details-management/', views.get_media_details_management, name='get_media_details_management')
```

De cette façon, on peut faire appel à la view get_media_details_management qui va tout d'abord récupérer les valeurs des variables mediaId et mediaType transmises dans le chemin de cette façon :

```
media_id = request.GET.get('media_id')
media_type = request.GET.get('media_type')
```

Ensuite selon le type du média, je vais récupérer son nom et son auteur/artiste/réalisateur/créateur que je vais mettre dans un objet data que je retourne en réponse à la requête :
Pour un livre, data correspond à cela :

```
data = {
   'title': 'nom du média',
   'author': 'auteur du livre'
}
return JsonResponse(data)
```

Côté javascript, je récupère la réponse et la convertit au format JSON pour récupérer les informations de data. Je peux maintenant récupérer les éléments adéquats et y définir leur valeur avec celles des propriétés de data.
Si du côté views.py je rencontrais une erreur alors data correspondait à cela :

```
data = {
   'error': "descriptif de l'erreur rencontrée"
}
```

Donc après avoir récupéré data côté javascript, je teste d'abord s'il contient la propriété 'error' pour adapter la valeur des éléments concernés :

```
const mediaInputUpdate = document.getElementById("media-id-update");
const mediaInfoUpdate = document.getElementById("media-error-update");
const mediaNameUpdate = document.getElementById("media-name-update");
const mediaAuthorUpdate = document.getElementById("media-author-update");

mediaInputUpdate.addEventListener("input", function() {
    const mediaId = this.value;
    const mediaTypeElement = document.getElementById("media-type-update");
    const mediaType = mediaTypeElement.value;

    if (mediaId) {
        fetch(`/bibliothecaire/get-media-details-management/?media_id=${mediaId}&media_type=${mediaType}`)
            .then(response => response.json())
            .then(data => {
                if (data.error) {
                    mediaInfoUpdate.innerHTML = `<p style="color:red;">${data.error}</p>`;
                    mediaNameUpdate.value = "";
                    mediaAuthorUpdate.value = "";
                } else {
                    mediaInfoUpdate.innerHTML = "";
                    mediaNameUpdate.value = data.title;
                    mediaAuthorUpdate.value = data.author;
                }
            });
    } else {
        mediaInfoUpdate.innerHTML = "";
        mediaNameUpdate.value = "";
        mediaAuthorUpdate.value = "";
    }
});
```

3. *Les livrables*

Le projet est disponible via le dépôt GitHub contenant le code source du projet :
https://github.com/fb-lb/CEF_devoirs_mediatheque

Besoins	Livrables
Site web abordable pour tous	Un site web répondant aux critères de la WCAG 2.1 Présence d'une barre de navigation pour accéder facilement à toutes les pages
Consultation des médias par le public	Page d'affichage des médias avec filtre sur le type du média et barre de recherche
Partie accessible uniquement aux bibliothécaires	Page de connexion avec restriction aux views de l'application bibliothécaire
Gestion des médias	Possibilité de créer / afficher / modifier / supprimer les médias dans l'application bibliothécaire
Gestion des membres	Possibilité de créer / afficher / modifier / supprimer les membres dans l'application bibliothécaire
Gestion des emprunts	Possibilité de créer / afficher / supprimer les emprunts dans l'application bibliothécaire + respects des conditions métiers*

*conditions métiers :

- 3 emprunts maximum pour un membre,
- 1 semaine de délai pour retourner un emprunt sinon l'utilisateur est bloqué
- pas d'emprunt si bloqué,
- jeux de plateaux non concerné par les emprunts.