



Centre Européen de Formation

# Dossier de Project

## **Créer un logiciel de gestion de médiathèque**

« Notre livre, notre média »

Tiago Machado

N° d'élève F471277A

Octobre 2024

## Index

1. Reprise du code existant
2. Structuration du projet
3. Vue d'ensemble des applications
4. Vue d'ensemble des Models
5. Vue d'ensemble des Views
6. Vue d'ensemble des Forms
7. Vue d'ensemble des URLs
8. Stratégie de tests
9. Étapes de l'installation du projet

## 1. Reprise du code existant

```
def menu():
    print("menu")

if __name__ == '__main__':
    menu()

class livre():
    name = ""
    auteur = ""
    dateEmprunt = ""
    disponible = ""
    emprunteur = ""

class dvd():
    name = ""
    realisateur = ""
    dateEmprunt = ""
    disponible = ""
    emprunteur = ""

class cd():
    name = ""
    artiste = ""
    dateEmprunt = ""
    disponible = ""
    emprunteur = ""

class jeuDePlateau :
    name = ""
    createur = ""

class Emprunteur():
    name = ""
    bloque = ""

def menuBibliotheque() :
    print("c'est le menu de l'application des bibliothécaire")

def menuMembre():
    print("c'est le menu de l'application des membres")
    print("affiche tout")
```

Le code existant présente plusieurs problèmes, tant au niveau du style que de la logique, surtout si l'on considère qu'il est destiné à être utilisé dans une application *Django*.

### 1. Organisation du code :

- Les classes et les fonctions sont mal formatées, manquent d'indentation et de structuration adéquate.
- La fonction *menu()* n'a aucune interaction utile avec le reste du code.
- La méthode *if \_\_name\_\_ == '\_\_main\_\_':* : elle est utilisée dans les scripts *Python* autonomes, mais dans un projet *Django*, le code est structuré différemment.

### 2. Pas d'utilisation de *models.Model* :

- Le code doit suivre l'architecture *MVC de Django*, où la couche de données est gérée par des classes qui héritent de *models.Model*. Les classes qui définissent les entités, telles que *livre*, *dvd*, *etc.*, doivent être des modèles *Django* avec des champs spécifiques (*CharField*, *DateField*, etc.).

### 3. Absence de méthodes utiles dans les classes :

- Les classes n'ont pas de méthodes ou de fonctionnalités associées. Dans le cas d'une application Django, vous devez ajouter des méthodes pertinentes (par exemple *save()*, *get\_absolute\_url()*, etc.) ou manipuler la logique requise pour chaque modèle.

#### **4. Absence d'internationalisation :**

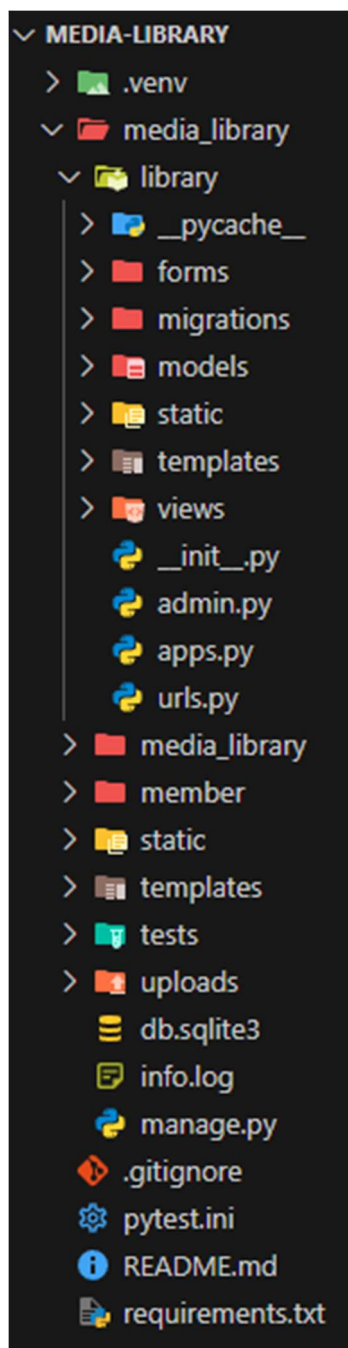
- Le nommage des classes et des méthodes est en Français. Par convention et bonne pratique, l'utilisation de l'Anglais est recommandée.

#### **5. Les méthodes de menu sont illogiques :**

- Les fonctions `menuBibliotheque()` et `menuMembre()` n'impriment que du texte, ce qui est insuffisant pour une application web. Il n'y a pas d'intégration avec les vues ou les modèles de *Django*.

Dans ce contexte, les points suivants montreront la structuration du projet, le fonctionnement général de l'application, et comment le code existant a été réutilisé et modifié afin de créer une application *Django* dynamique utilisant la méthodologie *CRUD (Create, Read, Update and Delete)* avec une base de données relationnelle *SQLite*.

## 2. Structuration du projet



La structure typique d'un projet *Django* suit un modèle organisé pour séparer les différents composants de l'application (tels que la configuration, les modèles, les vues, et les fichiers statiques).

Cette structure permet de séparer le code par fonctionnalité, ce qui rend le développement plus organisé et plus modulaire

J'ai utilisé la création d'un environnement virtuel (*.venv*) dans *Python* pour isoler les dépendances du projet afin qu'elles n'interfèrent pas avec d'autres installations *Python* sur le système. Un environnement virtuel est une copie indépendante de l'interpréteur *Python*, où nous pouvons installer des paquets spécifiques pour le projet.

|                              |                   |
|------------------------------|-------------------|
| <b>python3 -m venv .venv</b> | <b>Unix/macOS</b> |
| <b>py -m venv .venv</b>      | <b>Windows</b>    |

Commandes pour créer un environnement virtuel

|                                  |                   |
|----------------------------------|-------------------|
| <b>source .venv/bin/activate</b> | <b>Unix/macOS</b> |
| <b>.venv\Scripts\activate</b>    | <b>Windows</b>    |

Commandes pour activer un environnement virtuel

Le fichier *requirements.txt* contient les dépendances du projet. Pour les installer, utilisez la commande suivante

|   |                   |
|---|-------------------|
| <b>pip3 install -r requirements.txt</b> | <b>Unix/macOS</b> |
| <b>pip install -r requirements.txt</b>  | <b>Windows</b>    |

### 3. Vue d'ensemble des applications

Le projet Django est divisé en deux applications principales « Membre » et « Bibliothèque » :

#### Page Membre

- L'application membre, destinée aux utilisateurs ordinaires, nous amène à la page principale de l'application, où tous les médias sont listés. L'utilisateur a la possibilité de filtrer les médias par type et peut également effectuer une recherche par nom. Les médias répertoriés sont cliquables et il est possible de les visualiser en détail et de vérifier leur disponibilité.

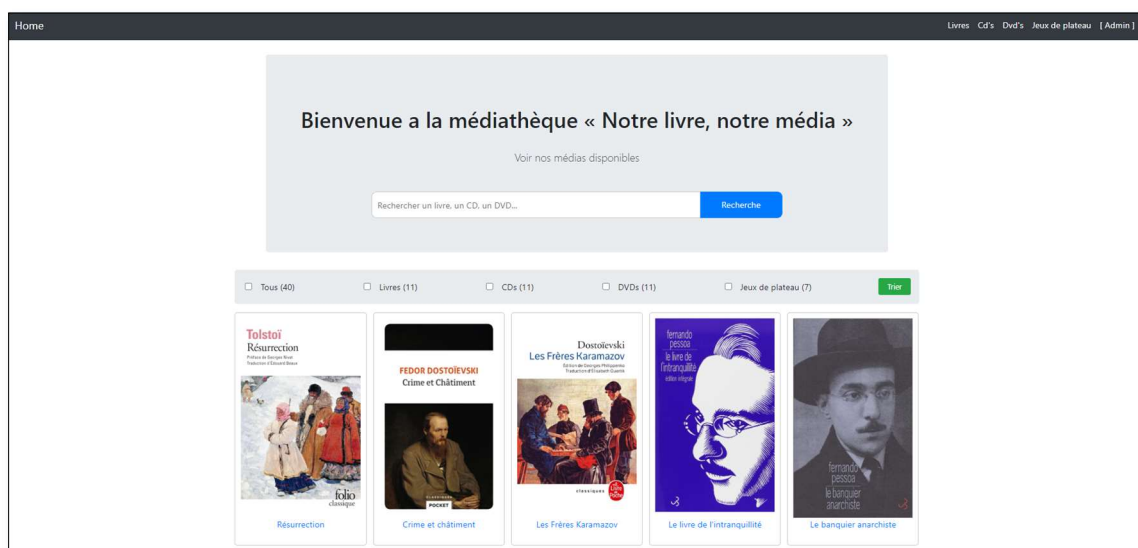


Figure 1 - Home page

- L'application **Bibliothèque** permet d'accéder à la partie administrative où l'on peut créer, lister, supprimer et mettre à jour différents types de médias ainsi que gérer les membres et les demandes de médias.

Cette application est protégée par un contrôle d'accès via un formulaire de connexion créé à l'aide du **système d'authentification de Django**. Ce système offre à la fois l'authentification et l'autorisation. Ces services sont appelés « **système d'authentification** » car ces fonctions sont en quelque sorte interconnectées.

<https://docs.djangoproject.com/en/5.1/topics/auth/default/>

L'accès se fait par le menu **Admin** de la page d'accueil, et nous serons redirigés vers la page de connexion. Par défaut, les identifiants d'accès ont été conservés dans les « **inputs** » afin de faciliter l'accès à des fins académiques, alors qu'ils devraient être supprimés si nous étions dans une phase de déploiement.

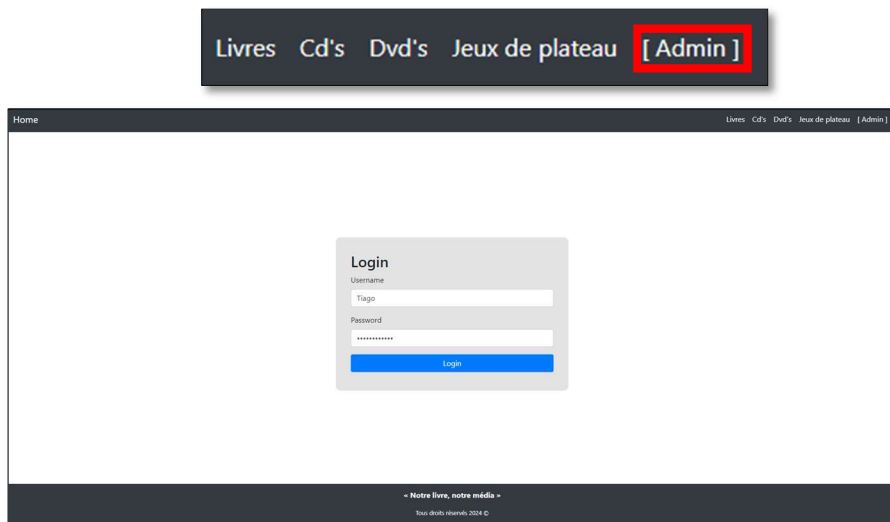


Figure 2- Formulaire de connexion



Dans la partie administrative, j'aurais pu créer une page pour l'enregistrement de nouveaux utilisateurs, mais comme il s'agit d'une partie sensible du projet, j'ai laissé cette possibilité de côté et il n'est possible de créer de nouveaux utilisateurs que par l'intermédiaire de [l'utilisateur principal](#).

Cet utilisateur est responsable de la [gestion de la base de données](#) à travers [l'interface](#) fournie par [Django](#) et gérer les profils qui peuvent accéder à la partie administrative, il est responsable de ne pas mettre en danger l'intégrité des données stockées.

Pour accéder à cette [interface](#), vous devez vous rendre à [URL](#) fourni et vous authentifier avec les identifiants suivants :

<http://127.0.0.1:8000/admin/>

**Username** : Admin

**Password** : 3#\_45aPgG\*89

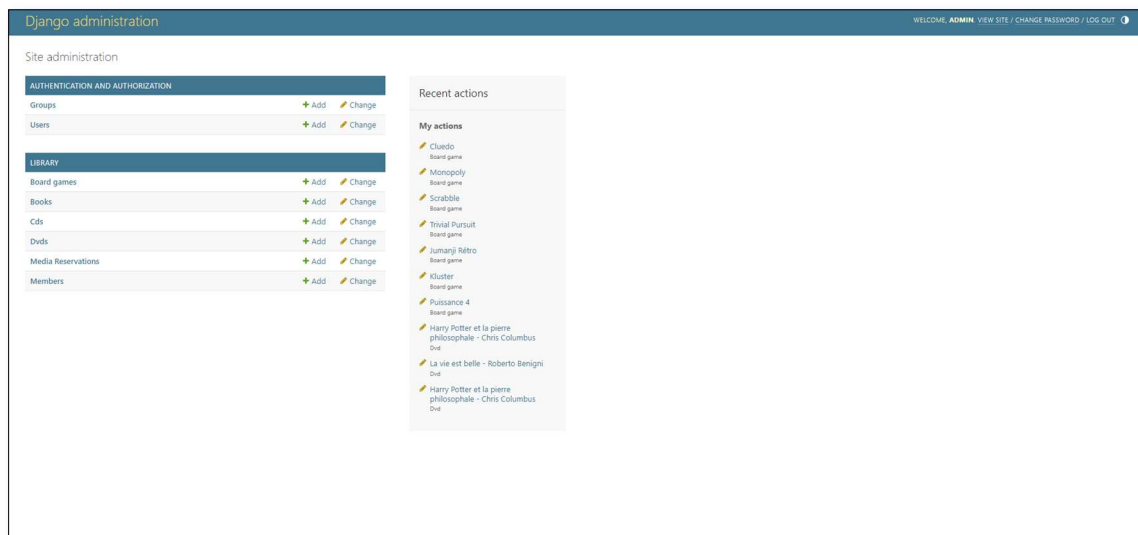


Figure 3- Interface d'administration de la base de données

## Page Bibliothèque

Sur la page principale de l'application [Bibliothèque](#), nous serons présents à une série de boutons dont la fonction est de nous rediriger vers chaque type de support, vers la page des membres ou vers la page des emprunts, où nous pourrions effectuer nos opérations de gestion.

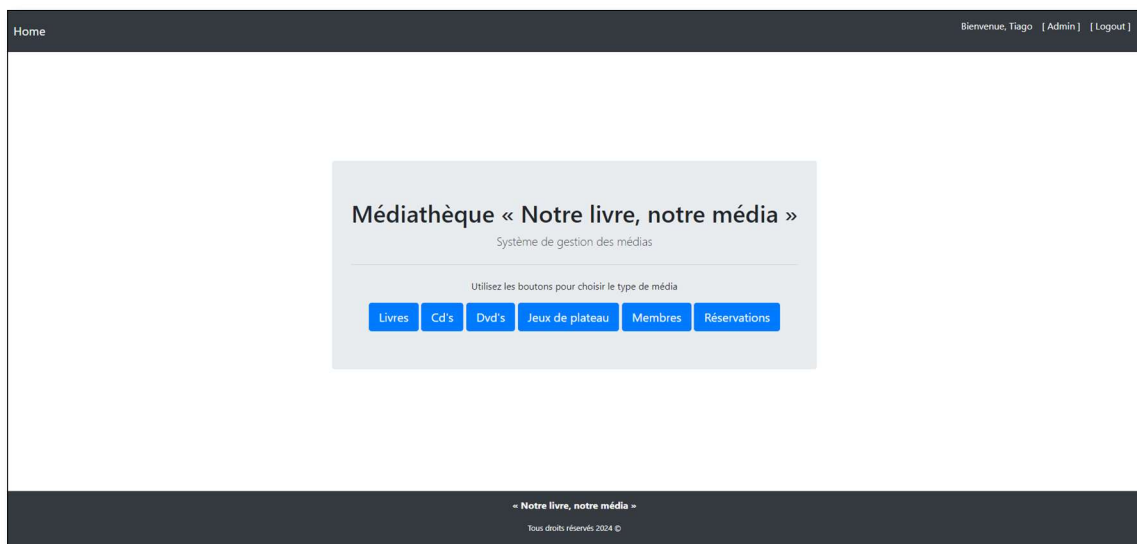


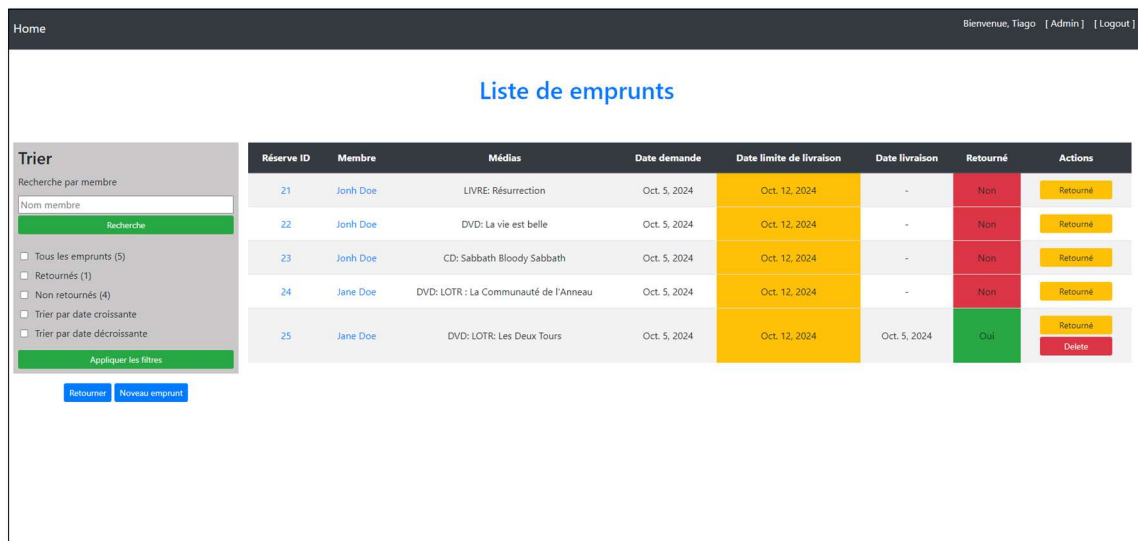
Figure 4 - Home page bibliothèque

## Page emprunts

Sur la page de [gestion des emprunts](#), la partie gauche présente un certain nombre de filtres à appliquer pour rendre les différents emprunts, ainsi qu'une recherche par membre pour voir tous les emprunts associés à un membre spécifique. La partie inférieure est occupée par les boutons de [retour](#) et de [création d'un emprunt](#).

Au centre et à droite de la page se trouve la [liste des emprunts](#), où l'on peut voir les données associées à chacun d'eux, à [savoir l'identifiant, le membre, le média demandé, la date de la demande, la date limite de livraison, la date réelle de livraison, si le média en question a été livré ou non](#).

La dernière colonne sert à marquer le prêt comme [retourné](#), en enregistrant la date et en marquant le média en question comme [disponible](#), et ce n'est qu'à partir de ce point que nous pouvons [supprimer](#) un prêt.



| Réserve ID | Membre   | Médias                                | Date demande | Date limite de livraison | Date livraison | Retourné | Actions   |
|------------|----------|---------------------------------------|--------------|--------------------------|----------------|----------|---|
| 21         | Jonh Doe | LIVRE: Résurrection                   | Oct. 5, 2024 | Oct. 12, 2024            | -              | Non      | <a href="#">Retourné</a>                        |
| 22         | Jonh Doe | DVD: La vie est belle                 | Oct. 5, 2024 | Oct. 12, 2024            | -              | Non      | <a href="#">Retourné</a>                        |
| 23         | Jonh Doe | CD: Sabbath Bloody Sabbath            | Oct. 5, 2024 | Oct. 12, 2024            | -              | Non      | <a href="#">Retourné</a>                        |
| 24         | Jane Doe | DVD: LOTR : La Communauté de l'Anneau | Oct. 5, 2024 | Oct. 12, 2024            | -              | Non      | <a href="#">Retourné</a>                        |
| 25         | Jane Doe | DVD: LOTR: Les Deux Tours             | Oct. 5, 2024 | Oct. 12, 2024            | Oct. 5, 2024   | Oui      | <a href="#">Retourné</a> <a href="#">Delete</a> |

Figure 5 - Page gestion emprunts

## Création d'un emprunt

Pour créer un [emprunt](#), nous sommes redirigés vers un [formulaire](#) contenant les champs nécessaires à sa réalisation.

Un membre qui est [bloqué](#) ou qui a [3 emprunts en même temps](#) ne pourra pas faire une autre demande, si l'un de ces cas se produit, un message sera affiché à l'utilisateur. De même qu'un membre dont la Media est en retard sera bloqué.

Il est également nécessaire de sélectionner au moins un type de média pour que l'emprunt soit ajouté avec succès. Les champs [date de requête](#) et [date de retournement](#) sont remplis automatiquement et ne sont pas modifiables. Si l'emprunt est terminé avec succès, nous serons redirigés vers la page de [gestion des emprunts](#).

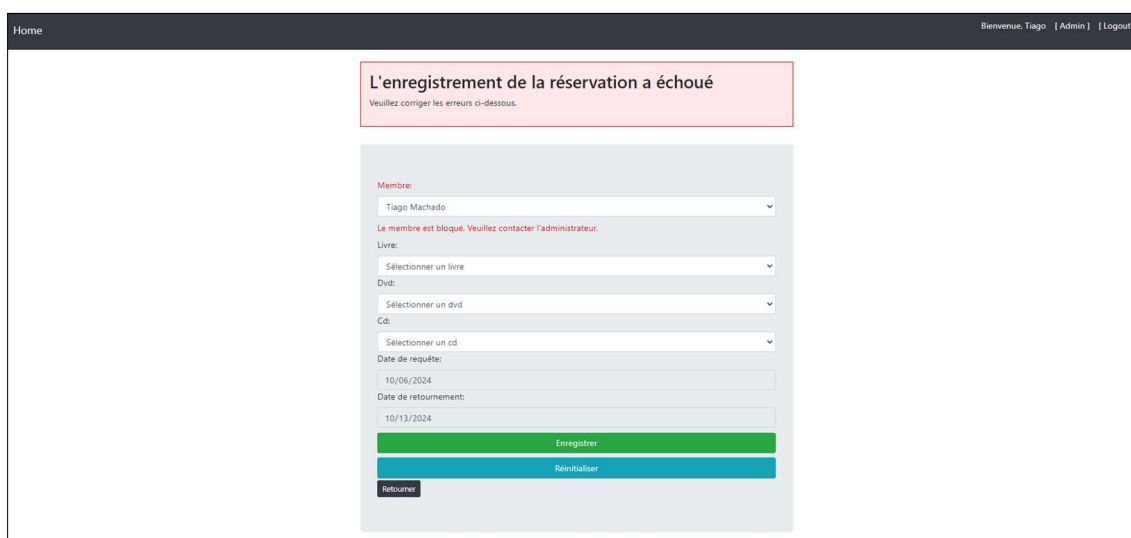
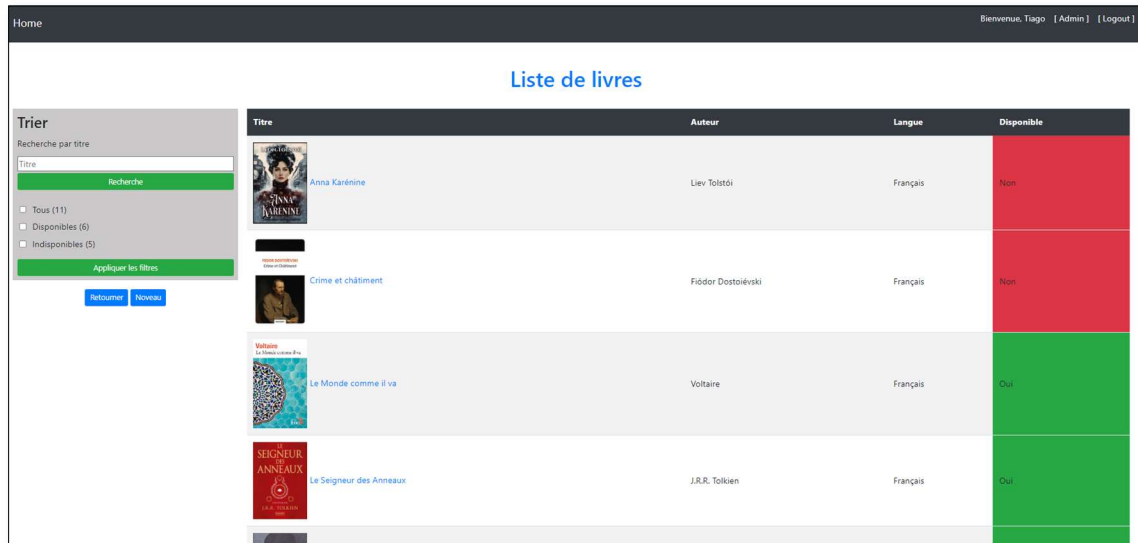


Figure 6 - Formulaire création emprunt

## Gestion des livres

La méthodologie utilisée pour gérer les [livres](#) est la même que celle utilisée pour les autres [médias](#) et [membres](#). La page de gestion des livres présente à un bloc de [filtrage](#) et une liste des livres présents dans la [base de données](#). Cette liste comprend les [champs titre, auteur, langue et disponible](#), les livres non disponibles étant mis en évidence en rouge dans la colonne correspondante.



The screenshot shows a web application titled 'Liste de livres'. On the left is a sidebar with a search bar and filters. The main area contains a table with the following data:





| Titre   | Auteur             | Langue   | Disponible |
|---|--------------------|----------|------------|
|  Anna Karénine           | Liev Tolstói       | Français | Non        |
|  Crime et châtiment      | Fiodor Dostoïévski | Français | Non        |
|  Le Monde comme il va    | Voltaire           | Français | Oui        |
|  Le Seigneur des Anneaux | J.R.R. Tolkien     | Français | Oui        |

Figure 7 - Gestion des livres

En cliquant sur l'image de chaque livre, on accède au livre sélectionné en [détail](#), où l'on peut voir les autres champs appartenant à ce type de média et où apparaît la possibilité [d'éditer le média](#).

Vous accéderez au [formulaire](#) de création contenant les [données](#) du livre sélectionné afin de pouvoir l'éditer. C'est également sur cette page que l'on peut [éliminer](#) le média sélectionné, en étant redirigé vers la [page de confirmation de l'élimination](#). Un média ou un membre avec des emprunts actifs ne peut pas être éliminé.

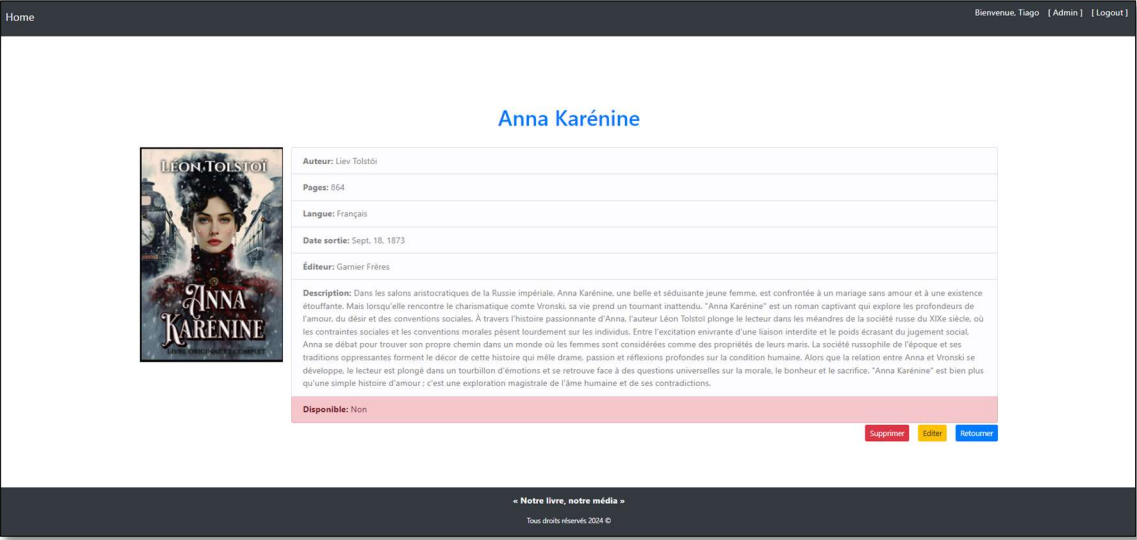


Figure 8 - Livre en détail



Figure 9 - Confirmation suppression

## 4. Vue d'ensemble des « Models »

Les modèles dans [Django](#) sont une partie fondamentale du cadre, responsable de la définition de la structure des données de l'application. Chaque [modèle](#) représente une table de la base de données et les champs définis dans le modèle sont les colonnes de cette table. [Django](#) offre une [API](#) de haut niveau pour interagir avec les données de la base de données, permettant des requêtes, des insertions, des mises à jour et des suppressions de manière simple et efficace.

Compte tenu du code existant, les classes de modèles suivantes ont été créées :

- classe `BoardGame(models.Model)`
- classe `Media(models.Model)`
- classe `Book(Media)`
- classe `Cd(Media)`
- classe `Dvd(Media)`
- classe `MediaReservations(models.Model)`
- classe `Member(models.Model)`

### Héritage

Les classes [Book](#), [Cd](#) et [Dvd](#) héritent de la classe [Media](#) les attributs qu'elles peuvent avoir en commun, ce qui nous permet d'éviter de répéter le code. J'ai décidé de créer cette [classe \(Media\)](#) en tant que [classe abstraite](#), de sorte que [Django](#) ne créera pas de table dans la base de données, ce sera une table de moins à consulter, ce qui peut améliorer les performances de l'application.

## Les relations

Dans la classe `MediaReservations`, des relations ont été créées avec `Member`, `Book`, `Dvd` et `Cd` en utilisant `ForeignKey (one-to-many)` : Utilisé pour lier ce modèle à un autre.

## Méthodes utiles dans les modèles

Méthodes personnalisées étaient ajoutées pour une logique spécifique. Par exemple, dans le modèle `MediaReservations`, existent méthodes telles que `get_media_items` et `return_item`, qui encapsulent des fonctionnalités utiles liées à cet objet, méthode `is_late()` pour vérifier si un membre a une média en retard et, si c'est le cas, le bloquer. Méthode `__str__` définit la manière dont l'objet sera représenté lorsqu'il sera converti en chaîne de caractères, ce qui est utile pour rendre les instances du modèle lisibles dans les interfaces d'administration ou de débogage.

<https://docs.djangoproject.com/fr/4.2/topics/db/models/>

## 5. Vue d'ensemble des « Views »

Dans ce projet, j'ai décidé d'utiliser les « `Class-Based Views` », essentiellement pour les raisons suivantes :

### 1. Réutilisation du code

Les `CBV` favorisent la réutilisation du code, puisqu'il est possible de créer des classes génériques et réutilisables qui peuvent être héritées et adaptées à différentes situations. `Django` propose déjà plusieurs `CBV` génériques telles que `ListView`, `DetailView`, `CreateView`, etc., qui facilitent les tâches courantes telles que l'énumération, la visualisation et la création d'objets.



## 2. Organisation et maintenance

Comme le code est structuré dans une classe, il est plus modulaire et plus facile à organiser. Des méthodes telles que `get_queryset`, `get_context_data`, `post`, etc., peuvent être utilisées pour encapsuler différentes parties de la logique de la vue.

## 3. Méthodes prédéfinies

Les CBV sont livrés avec des méthodes prédéfinies pour les opérations les plus courantes, telles que `get()`, `post()`, `put()`, `delete()`. Ces méthodes peuvent être facilement remplacées pour personnaliser le comportement.

## 4. Meilleure intégration avec les formulaires

Les CBV tels que `CreateView` et `UpdateView` sont déjà intégrés à `Django Forms`, ce qui facilite la manipulation des formulaires, la validation et l'enregistrement des données sans avoir à implémenter ces fonctionnalités manuellement.

## 5. Utilisation de mixins

Les CBV peuvent être combinés avec des `mixins`, qui sont de petites classes avec des fonctionnalités spécifiques qui peuvent être réutilisées dans différentes vues. Les `mixins`, tels que `LoginRequiredMixin` permettent d'ajouter facilement des fonctionnalités spécifiques sans dupliquer le code.

Par exemple, pour garantir qu'un utilisateur doit être authentifié pour accéder à une vue, il suffit d'ajouter `LoginRequiredMixin` en tant qu'héritage dans CBV.

<https://docs.djangoproject.com/en/5.1/topics/class-based-views/>

## 6. Vue d'ensemble des « Forms »

`ModelForm` est un outil puissant dans `Django` car il permet de générer automatiquement un formulaire basé sur un modèle `Django`, ce qui facilite la manipulation des objets du modèle dans les formulaires `HTML`. Permet d'avoir :

- **Labels personnalisés**

Le dictionnaire des `labels` permet de définir des labels personnalisés pour les champs du formulaire. Par exemple, le champ « Titre » apparaîtra dans le formulaire avec l'étiquette « Titre » au lieu de « Title ». Cette fonction est utile pour fournir à l'utilisateur final des `labels` plus descriptifs ou traduits.

- **Messages d'erreur personnalisés**

Le dictionnaire `error_messages` permet de définir des messages d'erreur personnalisés pour différents champs et différents types d'erreur

- **Widgets**

Sont utilisés pour personnaliser le code `HTML` généré pour chaque champ de formulaire. Ils sont utilisés pour ajouter des attributs tels que `placeholder`, la classe et, le type de date pour transformer le champ en un sélecteur de date en `HTML5`.

- **Validation automatique**

`ModelForm` est déjà doté d'une validation automatique basée sur le modèle. Cela signifie que si un champ du modèle est obligatoire (`blank=False`), le formulaire marquera également ce champ comme obligatoire.

<https://docs.djangoproject.com/en/5.1/topics/forms/modelforms/#modelform>

## 7. Structure des URL

Le projet met en œuvre une organisation claire et concise des [URL](#), ce qui facilite la navigation et la gestion des différentes fonctionnalités du système. Les [URL](#) sont définies de manière à correspondre aux vues correspondantes, ce qui fournit une interface intuitive permettant aux utilisateurs d'interagir avec les ressources disponibles.

- **URL des médias**

Les [URL](#) associées aux médias ([livres](#), [CD](#) et [DVD](#)) permettent d'accéder à diverses opérations, telles que la création, la mise à jour et la suppression de ressources. Par exemple, l'[URL create-book](#) permet de créer de nouveaux livres, tandis que l'[URL book-update](#) facilite la mise à jour des informations existantes. Cette structure simplifie la gestion de l'inventaire des médias.

- **URL de réservation**

Les réservations sont gérées via des [URL](#) dédiées, telles que [gest-reservations](#), qui affichent une liste de toutes les réservations effectuées. Ce point d'accès centralise la visualisation et la gestion des réservations, permettant aux utilisateurs de filtrer et de consulter rapidement leurs informations. L'[URL return-item](#) est responsable du traitement des retours de médias, ce qui permet aux utilisateurs de gérer facilement les articles qu'ils possèdent.

- **URL d'authentification**

L'inclusion d'[URL](#) pour l'authentification, comme la connexion et la déconnexion, fournit une couche de sécurité au système, garantissant que seuls les utilisateurs autorisés peuvent accéder à certaines fonctionnalités. Ces [URL](#) sont essentiels pour la protection des données et la personnalisation de l'expérience de l'utilisateur.

- **Une structure cohérente**

La structure des [URL](#) est cohérente et organisée, conformément aux bonnes pratiques en matière de dénomination et de hiérarchie. Chaque [URL](#) est clairement associé à une fonctionnalité spécifique, ce qui facilite la maintenance et l'évolutivité du projet.

- **Des URL « User Friendly »**

Les [slugs](#) sont utilisés pour créer des [URL plus intuitifs et conviviaux](#), facilitant l'identification du contenu par les utilisateurs et les moteurs de recherche. Par exemple, au lieu d'avoir une [URL telle que book-update/123/](#), l'utilisation du slug permet de former des URL telles que [book-update/new-book/](#), ce qui les rend plus lisibles et plus mémorables.

<https://docs.djangoproject.com/en/5.1/topics/http/urls/>

## 8. Stratégie de tests

Les tests ont utilisé une stratégie complète et structurée, incorporant diverses bonnes pratiques pour garantir la qualité et la fonctionnalité des vues. Voici quelques points forts de la stratégie de test appliquée :

- **Test fonctionnel**

Tests a [la fonctionnalité des vues](#) dans des [scénarios réels](#), tels que le [rendu des pages](#), le [filtrage des réservations](#) et la [manipulation des réservations](#) ([création](#), [mise à jour](#), [suppression](#)). Cela permet de s'assurer que la logique commerciale et l'interaction de l'utilisateur avec l'interface utilisateur fonctionnent comme prévu.

- **Utilisation de dispositifs**

L'utilisation de [fixtures](#), telles que [authenticated\\_client](#), [book](#), [dvd](#), [cd](#), [member](#) et [reservations](#), permet de créer un environnement de test cohérent et isolé. Il est ainsi plus facile de créer les données nécessaires aux tests sans

avoir à réécrire le code pour créer des objets pour chaque test. Les [fixtures](#) contribuent à rendre les tests plus propres et plus lisibles.

- **Vérification des réponses et des contextes**

Les tests vérifient non seulement que les [réponses](#) ont les [codes d'état attendus](#) (tels que 200 ou 302), mais ils valident également le contenu du [contexte renvoyé](#). Par exemple, ils vérifient que les réservations filtrées correspondent au membre recherché ou que le nombre total de réservations est correct. Cela permet de s'assurer que la logique de filtrage et de comptage fonctionne correctement.

- **Tests d'erreur et cas limites**

Test des [scénarios](#) dans lesquels il n'y a pas de résultats, par exemple lors de la recherche d'un membre qui n'existe pas. Ceci est important pour s'assurer que l'application se comporte correctement et fournit un retour d'information adéquat à l'utilisateur dans des situations inattendues.

- **Tests de formulaires**

Les tests qui vérifient la [création de réservations](#) avec des données valides et non valides garantissent que la [logique du formulaire](#) fonctionne correctement. Ils vérifient à la fois les redirections en cas de succès et les erreurs de validation lorsque les données ne sont pas valides. Cela permet de s'assurer que les [données saisies par l'utilisateur sont traitées correctement](#).

- **Tests de manipulation des données**

Le test d'opérations telles que le renvoi et la [suppression de réservations](#) permet de s'assurer que les [opérations de modification](#) de la base de données fonctionnent correctement et que les réservations sont [mises à jour](#) ou [supprimées](#) comme prévu.

- **Organisation structurelle**

Les tests sont clairement organisés au sein d'une classe de test unique, en utilisant des méthodes qui décrivent clairement ce que chaque test vérifie. Cela facilite la lecture et la maintenance du code de test.

## 9. Étapes de l'installation du projet

**Lien GitHub** - <https://github.com/masterxamss/media-library.git>

```
git clone git@github.com:masterxamss/media-library.git
```

```
cd media-library
```

### Créer un environnement virtuel

```
python3 -m venv .venv - Unix/macOS
```

```
py -m venv .venv - windows
```

### Activer l'environnement virtuel

```
source .venv/bin/activate - Unix/macOS
```

```
.venv\Scripts\activate -windows
```

### Installer les dépendances du projet

```
pip3 install -r requirements.txt - Unix/macOS
```

```
pip install -r requirements.txt - windows
```

```
cd media-library
```

### Run Server

```
python3 manage.py runserver - Unix/macOS
```