



15 janvier 2021

# RAPPORT PROJET GLOG 2020

[Génie Logiciel]

BINET Coline - PERRIER ALBAN

BORDEAUX INP

ENSC 2A – Groupe 2 & 1

# SOMMAIRE

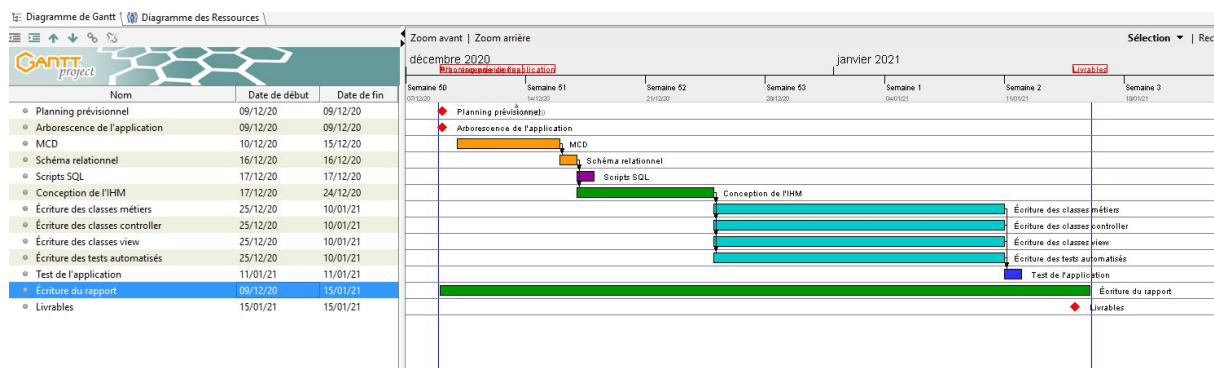
<b>INTRODUCTION</b>	<b>2</b>
Planning	2
Répartition des tâches	2
<b>NOTICE</b>	<b>5</b>
INSTALLATION	5
FONCTIONNALITÉS	7
<b>MODÉLISATION</b>	<b>9</b>
Diagramme des classes métier	9
Modèle Conceptuel de Données	11
Schéma relationnel	12
<b>CONCEPTION IHM</b>	<b>13</b>
<b>IMPLÉMENTATION</b>	<b>15</b>
Architecture technique de l'application	15
Tests automatisés	16
<b>CONCLUSION</b>	<b>24</b>
Difficultés rencontrées	24
Perspectives & améliorations	25
<b>ANNEXES</b>	<b>26</b>
Annexe 1 : Maquette visualisation Album	26
Annexe 2 : Maquette Formulaire d'inscription	26
Annexe 3 : Maquette Connexion	27
Annexe 4 : Maquette Profil Utilisateur	27
Annexe 5 : Maquette BDTheque : écran principal	28

# INTRODUCTION

Dans le cadre du module de Génie logiciel, nous avons effectué une application de gestion de collection de BD.

L'objectif de ce rapport est de présenter notre travail et justifier nos choix de conception. Il détaillera ainsi notre gestion de projet, une notice d'installation du projet, la modélisation du projet et de ses données, notre conception de l'IHM, les différentes fonctionnalités que nous avons implémentées ainsi que, pour conclure, un bilan.

## Planning



Notre planning pour ce projet est classique, il suit la conception attendue d'un projet logiciel avec la conception et modélisation en premières étapes, puis l'écriture du code correspondant.

## Répartition des tâches

La répartition des tâches s'est effectuée naturellement. Ayant l'habitude de travailler ensemble nous connaissons chacun nos forces et faiblesses, de fait, et pour changer, chacun a priorisé ce sur quoi il avait des difficultés. Ceci nous a permis d'avancer efficacement tout en nous permettant de nous entraider au besoin. Les tâches sont décrites dans l'ordre chronologique de début de réalisation.

A → Alban | C → Coline

La case "Qui" décrit la ou les personnes qui ont **principalement** travaillé sur l'aspect du projet décrit dans la case "Quoi". En effet, nous avons beaucoup fonctionné par pair programming où tout du moins nous communiquons beaucoup ensemble même si l'autre n'était pas directement devant le code.

Qui	Quoi	Quand	Durée
A + C	Planning prévisionnel	09/12	30min
A	Arborescence de l'application	09/12	30min
A + C	Diagramme des classes	10/12	1h
A + C	MCD + MDP	10/12	2h
A	Maquettage de l'application sous Axure	15/12	3h
C	Structure classes métiers / NHibernate dont Mapping & Exporteur	17/12	3h
A	Structure des Repository des classes métiers	26/12	1h
A+C	Formulaires de connexion et utilisateur de l'application	26/12	2h30
A	Implémentation des données sur les formulaires (méthodes des repository)	26/12	1h30
C	Correction Mapping + Création de contenu pour la BD	03/01	2h
A + C	Pair Programming : Résolution de problèmes	04/01	1h30
A	Correction des requêtes pour l'implémentation du contenu	05/01	4h
A	Se déconnecter + Supprimer Album + form rechercheAlbum et Souhaits + début EF 04, 05, 06, 07, 08	06/01	3h
C	Écriture du script SQL pour les données	06/01	2h
A+C	Pair programming en classe Implémentation des données dans la BD correction et finition des EF du 06/01	08/01	3h
C	Tests Unitaires Domain/DAL + correction SQL / LINQ sur tables jointes (accès genreParAlbumId, etc)	12/01	4h
C	Fin tests Unitaire DAL + suppression souhait/collection	13/01	2h

	+ ajouter dans collection implique suppression souhait		
A+C	Pair Programming : form administrateur + Ajouter nouvel Album	13/01	2h
A	Finition form ajout de contenu (album, genre, auteur, série, ...)	14/01	1h
A+C	Correction bugs + feedbacks ajout/suppression	14/01	2h
C	Finition tests unitaires	15/01	1h30
A	Corrections bugs & optimisation	15/01	1h

# NOTICE

## INSTALLATION

Pour installer notre application ainsi que sa base de données associées, vous devez des logiciels suivants (ou leurs équivalents) :

- [Visual Studio 2019](#)
- [Xampp](#)

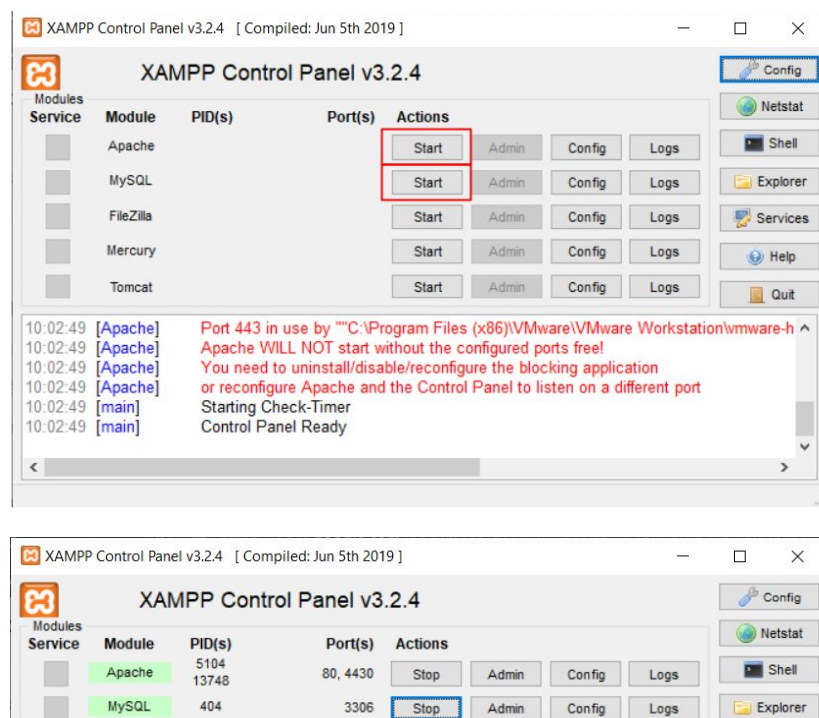
Une fois ces logiciels installés sur votre ordinateur, vous pouvez récupérer le code source de notre projet sur le lien github suivant (nécessite d'être connecté) :

<https://github.com/ensc-glog/projet-2020-coline-alban>

À partir de ce lien, clonez le dépôt ou bien téléchargez le en ZIP (une extraction du dossier sera nécessaire dans ce cas).

Ensuite, ouvrez le fichier solution au format .sln : **"projet-2020-coline-alban.sln"** avec Visual Studio.

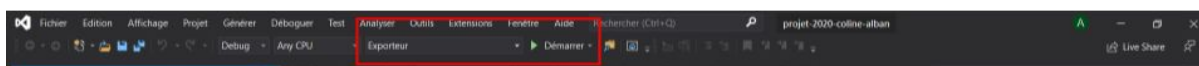
Puis lancez **Xampp**, et cliquez sur les boutons **"start"** des lignes correspondant à **"Apache"** et **"MySQL"**, après quelques secondes les deux modules devraient être de couleur verte.



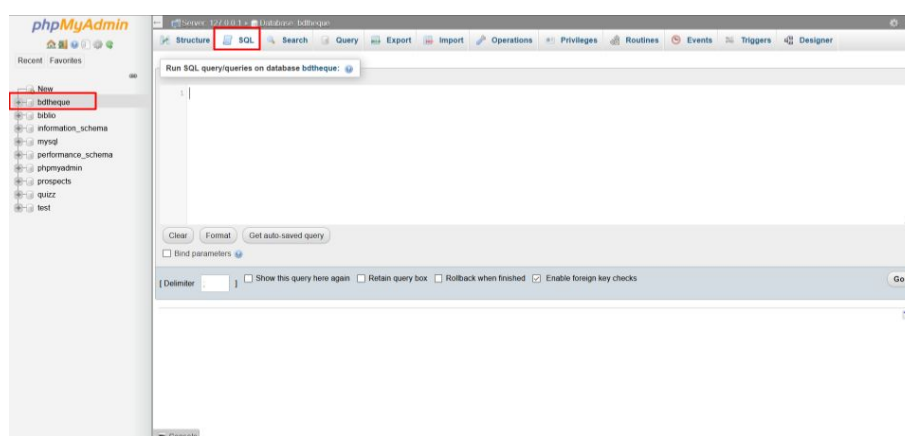
Vous pouvez désormais normalement accéder à cette URL dans votre navigateur préféré : <http://localhost/phpmyadmin/>

Ensuite, sur la solution chargée sur Visual Studio, vous pouvez ouvrir le fichier “[bdtheque\\_database.sql](#)” contenu dans le dossier *DAL/DB*, copiez-en le contenu. Retournez sur phpmyadmin (sur votre navigateur), et cliquez sur l’onglet en haut “SQL”. Collez le contenu du fichier sql copié, puis cliquez sur le bouton “Go” ou “Exécuter” (selon la version de langue) en bas à droite. Vous venez de créer votre base de données.

Désormais, vous pouvez lancer le projet “Exporteur” dans la solution Visual Studio.



Cela devrait vous ouvrir une fenêtre de console en écrivant les scripts SQL créant la structure de la base de données, laissez le s’exécuter jusqu’à ce qu’il affiche le message “Done!”, vous indiquant que vous pouvez désormais fermer la console. Maintenant, allez dans le fichier “[bdtheque\\_data.sql](#)” contenu dans le dossier *DAL/DB* et copié/collé en le contenu dans l’onglet SQL de phpmyadmin, une fois que vous aurez sélectionné la base de donnée nommée “bdtheque” dans le menu à gauche et ensuite l’onglet SQL en haut.



Vous venez de remplir votre base de données de contenu, maintenant vous pouvez utiliser l’application correctement.

Pour cela, il vous suffit de sélectionner le projet “App” dans Visual Studio et de le démarrer.

## FONCTIONNALITÉS

Toutes les exigences fonctionnelles ont été respectées, elles sont listées dans le tableau suivant :

Code	Description
EF_01	En tant qu'utilisateur, je peux me connecter à l'application grâce à mes identifiants (login/mot de passe).
EF_02	En tant qu'utilisateur, je peux consulter la liste de mes albums.
EF_03	En tant qu'utilisateur, je peux afficher des informations détaillées sur un album : image de couverture, nom, série, auteur(s), catégorie (BD/manga/comic/...), genre (fantasy/polar/jeunesse/...), éditeur.
EF_04	En tant qu'utilisateur, je peux effectuer une recherche dans la liste des albums du marché. Cette recherche peut être basée sur les critères suivants : nom (ou partie du nom), série, auteur, genre.
EF_05	En tant qu'utilisateur, je peux ajouter un ou plusieurs album(s) du marché à la liste de mes albums.
EF_06	En tant qu'utilisateur, je peux ajouter des albums du marché à ma liste de souhaits. Cette liste est mise à jour en cas d'achat d'un album.
EF_07	En tant qu'utilisateur, je peux consulter la liste de mes souhaits.
EF_08	En tant qu'utilisateur, je peux retirer un ou plusieurs album(s) de la liste de mes souhaits.
EF_09	En tant qu'utilisateur, je peux me déconnecter de l'application pour revenir à l'écran d'accueil permettant de s'y connecter.
EF_10	En tant qu'administrateur, je peux me connecter à l'application grâce à des identifiants spécifiques (login/mot de passe).
EF_11	En tant qu'administrateur, je peux ajouter un album à la liste des albums du marché.



À celles-ci, nous avons rajouté des exigences non fonctionnelles :

Code	Description
ENF_01	En tant qu'utilisateur, je peux supprimer un album de la liste de ceux que je possède.
ENF_02	En tant qu'utilisateur, je peux consulter des informations sur moi (nom et prénom).
ENF_03	En tant qu'administrateur, je peux ajouter un genre, une catégorie, une série, un auteur dans la liste de ceux qui existent déjà, cette liste est mise à jour.
ENF_04	Il n'est pas possible de s'inscrire avec un login déjà existant.
ENF_05	Il n'est pas possible d'ajouter un album à sa liste de souhaits ou sa collection s'il en fait déjà partie.
ENF_06	Les saisies de l'utilisateur dans les controleur <i>textBox</i> sont exempts d'accents et de caractères spéciaux.
ENF_07	Les noms des éléments des WinForms respectent la norme <a href="#">Hungarian Notation Prefixes</a> .

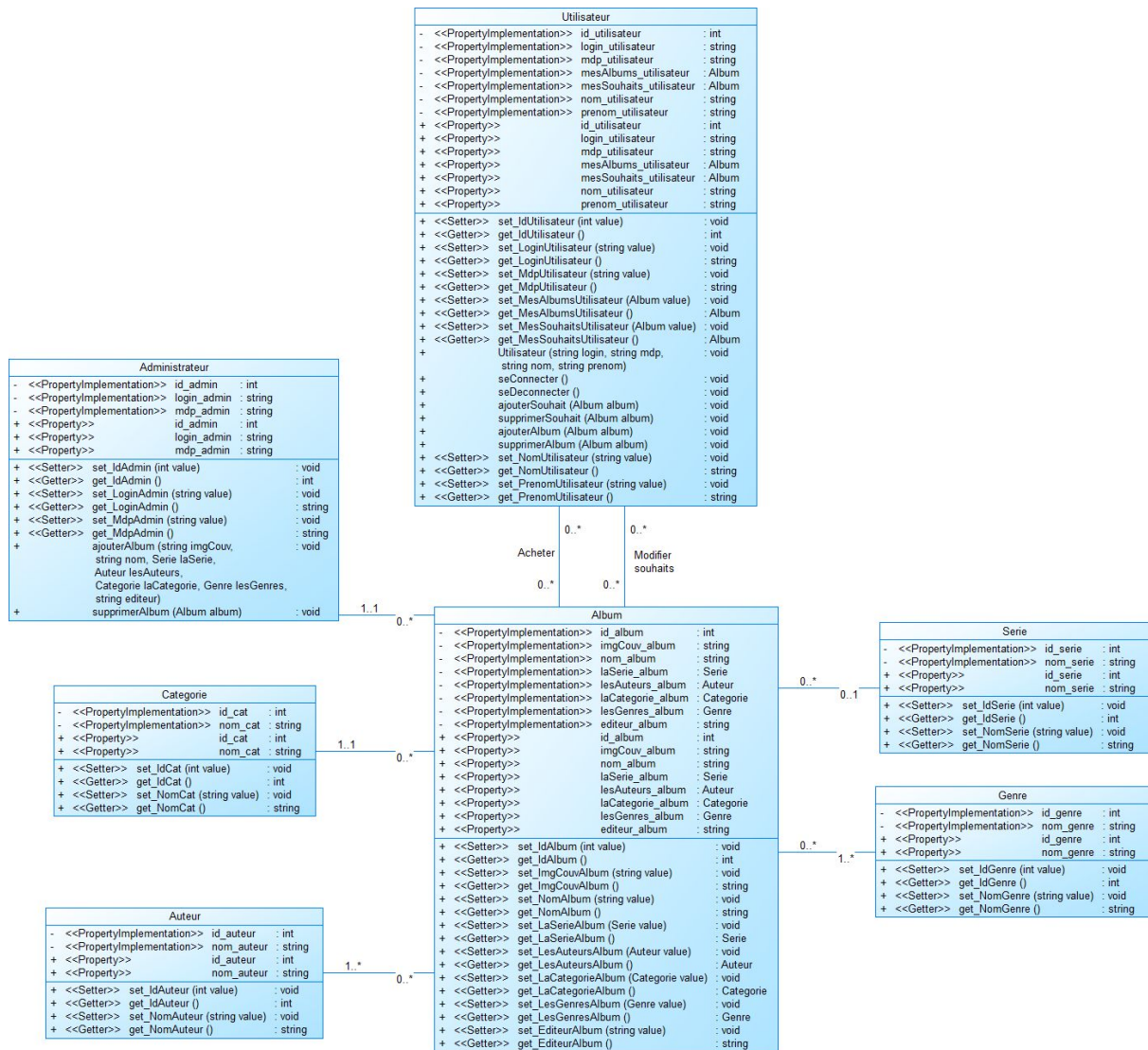
# MODÉLISATION

## Diagramme des classes métier

Une classe correspond à un objet, cet objet peut être défini par plusieurs attributs.

C'est ce que nous avons modélisé sur ce diagramme à l'aide du logiciel

PowerDesign.



Un objet de la classe **Album** est donc caractérisé par un identifiant (*id\_Album*), une image de couverture (*imgCouv\_album*), un nom (*nom\_album*), un éditeur (*editeur\_album*), mais appartient aussi à une certaine catégorie (*laCategorie\_album*), à une certaine série (*laSerie\_album*), à un ou plusieurs genres (*lesGenres\_album*) et a été écrit par un ou plusieurs auteurs (*lesAuteurs\_album*). Ces quatre attributs sont donc issus des classes correspondantes : **Categorie**, **Serie**, **Genre** et **Auteur**.

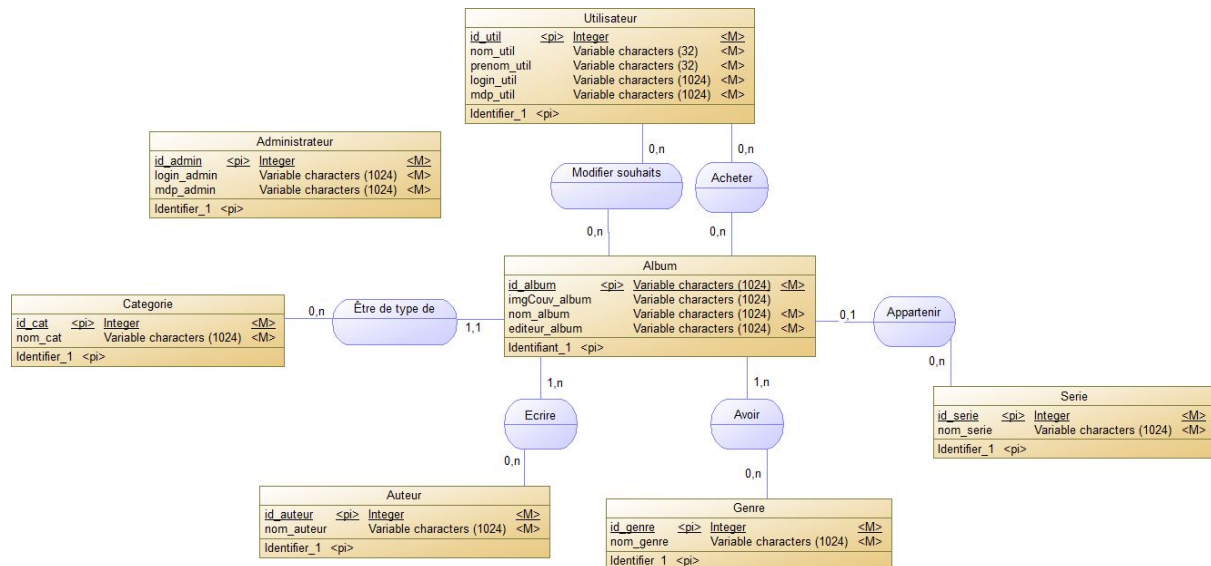
*laSerie\_album* et *laCategorie\_album* correspondent à des objets alors que *lesGenres\_album* et *lesAuteurs\_album* sont des listes d'objet(s) de ces classes. Elles sont toutes les trois construites de la même manière avec un identifiant (*id\_classe*) et un nom (*nom\_classe*) (à l'exception de l'auteur à qui on a attribué un prénom par la suite).

Nous avons ensuite une classe **Utilisateur**, caractérisée par un identifiant (*id\_utilisateur*), un nom (*nom\_utilisateur*), un prénom (*prenom\_utilisateur*), un login (*login\_utilisateur*) et un mot de passe (*mdp\_utilisateur*) pour se connecter. Un objet de la classe **Utilisateur** peut ajouter des objets de type **Album** dans une liste de souhaits, nous lui avons donc ajouté un attribut *mesSouhaits\_utilisateur*, contenant une liste d'objets de type **Album**. De la même manière, un objet de la classe **Utilisateur** peut posséder des objets de type **Album**, correspondant à sa collection, nous lui avons donc ajouté un attribut *mesAlbums\_utilisateur*, contenant une liste d'objets de type **Album**.

Concernant l'administrateur de l'application, nous avons fait le choix d'en faire une classe à part plutôt que de la faire hériter de la classe **Utilisateur**, cela afin d'éviter d'avoir à lui instancier des attributs inutiles.

## Modèle Conceptuel de Données

À partir de ce diagramme de classes, nous avons modélisé le MCD associé :



Chaque classe a été modélisée par une entité portant son nom : **Utilisateur**, **Administrateur**, **Album**, **Catégorie**, **Auteur**, **Genre** et **Série**. Leurs attributs sont devenus des propriétés avec chacune un identifiant unique et servant de clé primaire.

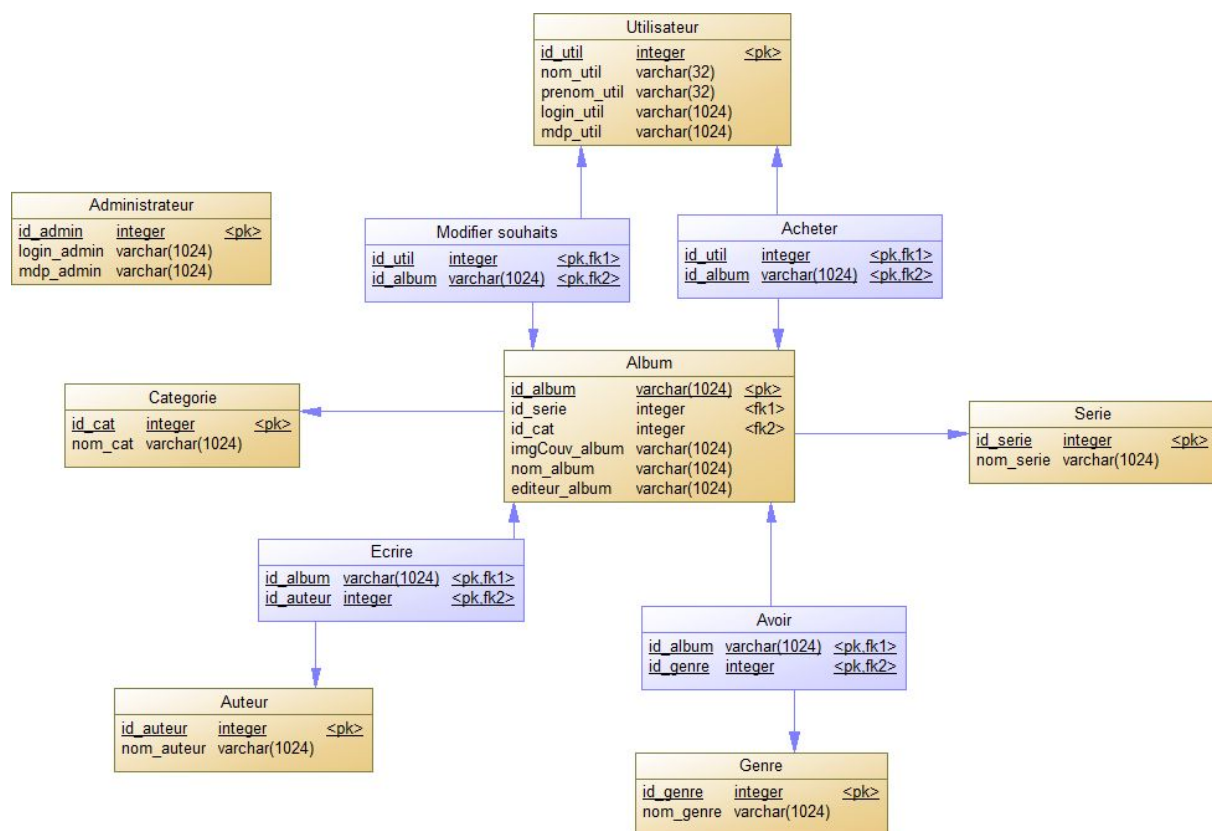
Concernant les liens entre les classes réalisées avec des attributs représentant des listes d'objets expliqués précédemment, nous les avons ici représentés par des associations entre les différentes entités :

- Dans la classe **Album** :
  - L'attribut *lesAuteurs* : un album peut être écrit par un ou plusieurs auteurs (1,n)
  - L'attribut *laSerie* : un album peut faire partie de 0 ou 1 série (0,1) et une série peut avoir 0 ou plusieurs albums (1,n)
  - L'attribut *laCatégorie* : un album fait partie d'une et une seule catégorie (1,1), et une catégorie peut représenter 0 ou plusieurs albums (0,n)
  - L'attribut *lesGenres* : un album appartient à un ou plusieurs genres (1,n)

- Dans la classe **Utilisateur** :
  - L'attribut *mesSouhaits* : un utilisateur peut souhaiter un ou plusieurs albums (0,n) et un album peut être souhaité par un ou plusieurs utilisateurs
  - L'attribut *mesAlbums* : un utilisateur peut posséder un ou plusieurs albums (0,n) et un album peut être possédé par un ou plusieurs utilisateurs

## Schéma relationnel

À partir de ce MCD, nous avons généré le schéma relationnel suivant, représentant les différentes clés primaires et secondaires des entités.

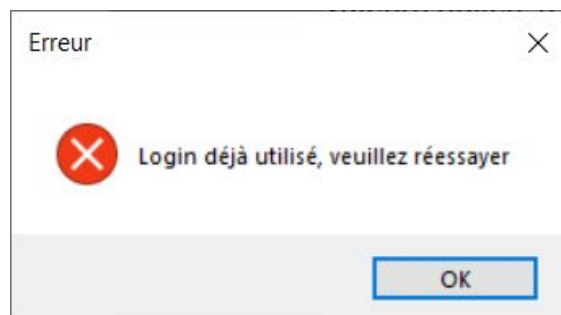


## CONCEPTION IHM

Le développement de notre application s'est effectué en s'appuyant sur des maquettes préalablement dessinées sur l'application Axure RP (cf. [Annexes](#)). Ces maquettes basse fidélité présentent l'architecture de chaque écran (WinForm) ainsi que les différents éléments prévus.

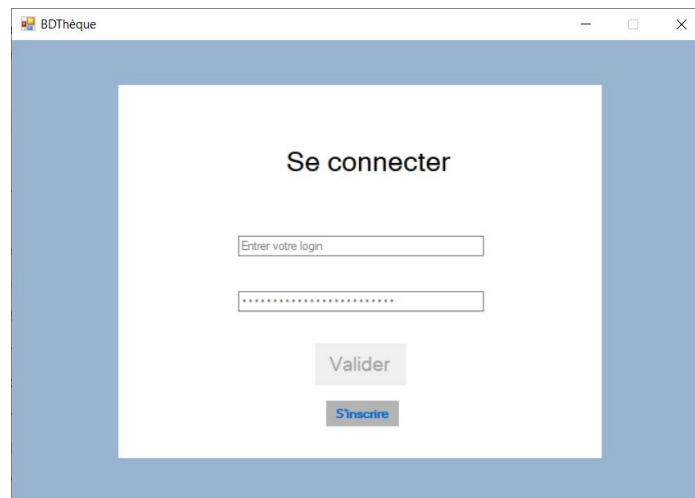
Afin de rendre l'application la plus intuitive, nous avons adapté les différents éléments de celle-ci afin de renvoyer autant de feedback que nécessaire aux utilisateurs. Nous essayons également au maximum de limiter le risque d'erreur.

Ainsi, les formulaires de l'application sont sécurisés par différentes vérifications. Celles-ci permettent, entre autres, d'éviter de laisser des valeurs nulles, d'empêcher la sauvegarde de doublons ou d'éviter la saisie de caractères spéciaux. Dans le premier cas, le bouton permettant d'envoyer le formulaire est désactivé (non cliquable et grisé) tant que tous les champs ne sont pas saisis.



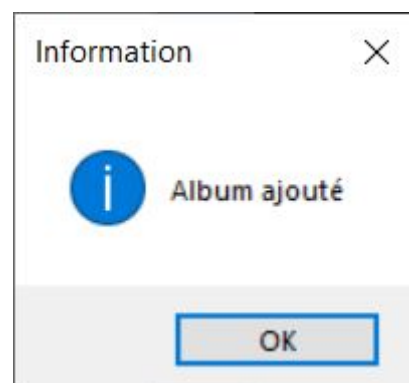
*Exemple de fenêtre affichée lorsqu'on essaye de s'inscrire avec un login déjà existant*

Lorsqu'un utilisateur s'inscrit ou se connecte, son mot de passe est dissimulé par autant de "." qu'il a entré de caractères, ce qui assure que personne ne puisse en prendre connaissance en regardant l'écran. De plus, nous avons imité (à l'aide d'événements à défaut d'avoir une propriété pour le gérer) le comportement d'un placeholder avec du texte inscrit donnant les indications de ce qu'il faut rentrer dans les champs.



*Exemple du placeholder + bouton grisé (ne pouvant pas être cliqué) car aucune information n'a été saisie*

Lorsqu'une action a bien été effectuée (inscription, ajout, suppression), ou au contraire, lorsqu'une erreur a été produite, une pop-up s'ouvre pour le préciser à l'utilisateur. Les données affichées à l'écran se mettent à jour en conséquence lorsque c'est nécessaire (par exemple : mise à jour de la liste des albums qu'un utilisateur souhaite une fois que ce dernier a choisi d'y ajouter un album).



*Exemple de fenêtre s'affichant quand un album a été ajouté*

Afin d'éviter la multiplication de fenêtres, nous avons choisi de développer plusieurs des écrans de l'application sur un seul WinForm. Ainsi, l'écran de connexion, ainsi que les écrans principaux de l'utilisateur et de l'administrateur se concrétisent par différents panel sur le *mainform*.

Les autres écrans sont des forms distincts.



# IMPLÉMENTATION

## Architecture technique de l'application

Conformément à l'ET\_03, l'application est structurée selon une architecture en couches (**App/DAL/Domain**). La version du framework .NET utilisé dans notre solution est la 4.6.1.

La couche **App** correspond aux interfaces Hommes-Machine, on peut y retrouver tous les winForms de l'application.

La couche **DAL** (Data Access Layer) contient l'accès aux données. C'est là que se situent les interfaces des repository des classes métiers (ex : **IAlbumRepository**), accompagnées de leurs classes repository définies (ex : **AlbumRepository**). Dans ces classes, il y a les méthodes permettant d'accéder aux données. Nous y avons également placé un dossier **DB** qui contient les scripts SQL correspondant à la création de la base de données associée à l'application.

La couche **Domain** contient tous les processus métiers, c'est-à-dire les classes des différents objets utilisés dans notre application. Elle contient également un dossier "Mapping", dossier contenant des fichiers .hbm.xml, qui décrivent les liens faits entre la base de données et les objets de l'application. Ces liens et ce mapping correspondent à l'utilisation de NHibernate, qui est une solution de mappage relation-objet.

La solution comporte trois projets supplémentaires :

Les projets **DALTests** et **DomainTests** sont deux projets de tests unitaires. Ils recensent et exécutent tous les tests relatifs aux coupes **DAL** et **Domain**. Ces tests sont présentés plus bas.

Le dernier projet est le projet **Exporteur**, qui permet de générer automatiquement toute la structure de la base de données BDtheque en fonction de ce qui a été développé dans le projet **Domain** et spécifié dans le dossier Mapping évoqué plus tôt.



## Tests automatisés

Ci dessous un tableau listant les différents tests effectués sur les couches **Domain** et **DAL**.

Les tests s'effectuent sur un extrait de la base de données. Il n'y a donc que 4 albums sur les 15 préparés dans le fichier '**bdtheque\_database.sql**' qui sont ici exploités. Davantage de données aurait rendu les tests plus longs à écrire comme à effectuer.

N° TEST	NOM DU TEST	CLASSE ASSOCIÉE	MÉTHODE TESTÉE	DESCRIPTION
T_01	Test_GetAll()	AdministrateurRepository	GetAll()	Vérifie que la fonction retourne tous les <b>Administrateurs</b> de la base de données : <ul style="list-style-type: none"> <li>- vérifie que le nombre est celui attendu</li> <li>- Vérifie que les <i>logins</i> sont ceux attendus</li> </ul>
T_02	Test_GetAdminConnexion()	AdministrateurRepository	GetAdminConnexion()	Vérifie que la fonction retourne l' <b>Administrateur</b> associé aux <i>login</i> et au <i>mdp</i> donnés.
T_03	Test_GetAdminConnexion_Erreur()	AdministrateurRepository	GetAdminConnexion()	Vérifie que la fonction retourne <i>null</i> si aucun <b>Administrateur</b> ne correspond aux <i>login</i> et au <i>mdp</i> donnés.
T_04	Test_GetAll	AlbumRepository	GetAll()	Vérifie que la fonction retourne tous les <b>Albums</b> de la base de données : <ul style="list-style-type: none"> <li>- vérifie que le nombre est celui attendu</li> <li>- Vérifie que les <i>nomAlbum</i> sont ceux attendus</li> </ul>
T_05	Test_GetAlbumParId	AlbumRepository	GetAlbumParId()	Vérifie que la fonction retourne le bon <b>Album</b> en comparant le <i>nomAlbum</i> avec

				celui attendu
T_06	Test_GetAlbumParId_Erreur()	AlbumRepository	GetAlbumParId()	Vérifie que la fonction retourne <i>null</i> en cas d' <i>id</i> incorrect.
T_07	Test_GetAlbumParNom()	AlbumRepository	GetAlbumParNom()	Vérifie que la fonction retourne le bon <b>Album</b> en comparant l' <i>id</i> avec celui attendu.
T_08	Test_GetAlbumParNom_Erreur()	AlbumRepository	GetAlbumParNom()	Vérifie que la fonction retourne aucun <b>album</b> en cas de <i>NomAlbum</i> incorrect.
T_09	Test_GetAlbumParSerie()	AlbumRepository	GetAlbumParSerie()	Vérifie que la fonction retourne les bons <b>Albums</b> : <ul style="list-style-type: none"> <li>- vérifie que le nombre est celui attendu</li> <li>- Vérifie que les <i>nomAlbum</i> sont ceux attendus</li> </ul>
T_10	Test_GetAlbumParSerie_Erreur()	AlbumRepository	GetAlbumParSerie()	Vérifie que la fonction retourne aucun <b>album</b> en cas de <i>serie</i> incorrecte.
T_11	Test_GetAlbumParAuteur()	AlbumRepository	GetAlbumParAuteur()	Vérifie que la fonction retourne les bons <b>Albums</b> : <ul style="list-style-type: none"> <li>- vérifie que le nombre est celui attendu</li> <li>- Vérifie que les <i>nomAlbum</i> sont ceux attendus</li> </ul>
T_12	Test_GetAlbumParAuteur_Erreur()	AlbumRepository	GetAlbumParAuteur()	Vérifie que la fonction retourne aucun <b>album</b> en cas d' <i>auteur</i> incorrect.
T_13	Test_GetAlbumParGenre()	AlbumRepository	GetAlbumParGenre()	Vérifie que la fonction retourne les bons <b>Albums</b> : <ul style="list-style-type: none"> <li>- vérifie que le nombre est celui attendu</li> <li>- Vérifie que les <i>nomAlbum</i> sont ceux attendus</li> </ul>

T_14	Test_GetAlbumParGenre_Erreur()	AlbumRepository	GetAlbumParGenre	Vérifie que la fonction retourne aucun <b>album</b> en cas de <i>genre</i> incorrect.
T_15	Test_Save()	AlbumRepository	Save()	Vérifie que l'ajout d'un nouvel <b>album</b> se fait correctement en BD : <ul style="list-style-type: none"> <li>- Vérifie qu'il y a le bon nombre d'<b>albums</b> en BD</li> <li>- vérifie que le nouvel <b>album</b> a les bons <b>genres</b> (table jointe)</li> <li>- vérifie que le nouvel <b>album</b> a les bons <b>auteurs</b> (table jointe)</li> </ul>
T_16	Test_GetAll()	AuteurRepository	GetAll()	Vérifie que la fonction retourne tous les <b>Auteurs</b> de la base de données : <ul style="list-style-type: none"> <li>- vérifie que le nombre est celui attendu</li> <li>- Vérifie que les <i>nom</i> sont ceux attendus</li> </ul>
T_17	Test_GetAuteursParAlbumId()	AuteurRepository	GetAuteursParAlbumId()	Vérifie que pour un <i>id</i> d' <b>album</b> donné, les <b>auteurs</b> retournés sont les bons : <ul style="list-style-type: none"> <li>- vérifie leur nombre</li> <li>- vérifie leur <i>nom</i>.</li> </ul>
T_18	Test_GetAuteursParAlbumId_Erreur()	AuteurRepository	GetAuteursParAlbumId()	Vérifie que pour un <i>id</i> d' <b>album</b> inexistant, le nombre d' <b>auteurs</b> associé est nul.
T_19	Test_Save()	AuteurRepository	Save()	Vérifie qu'un nouvel <b>auteur</b> est bien sauvegardé en BD : <ul style="list-style-type: none"> <li>- vérifie que le nombre d'<b>auteurs</b> total est bon</li> </ul>
T_20	Test_GetAll()	CategorieRepository	GetAll()	Vérifie que la fonction retourne tous les <b>catégories</b> de la base de données : <ul style="list-style-type: none"> <li>- vérifie que le nombre est celui attendu</li> <li>- Vérifie que les <i>nomCategorie</i> sont ceux attendus</li> </ul>

T_21	Test_Save()	CategorieRepository	Save()	Vérifie qu'une nouvelle <b>catégorie</b> est bien sauvegardé en BD : <ul style="list-style-type: none"> <li>- vérifie que le nombre de <b>catégories</b> total est bon</li> </ul>
T_22	Test_GetAll()	GenreRepository	GetAll()	Vérifie que la fonction retourne tous les <b>genres</b> de la base de données : <ul style="list-style-type: none"> <li>- vérifie que le nombre est celui attendu</li> <li>- Vérifie que les <i>nomGenre</i> sont ceux attendus</li> </ul>
T_23	Test_Save()	GenreRepository	Save()	Vérifie qu'un nouveau <b>genre</b> est bien sauvegardé en BD : <ul style="list-style-type: none"> <li>- vérifie que le nombre de <b>genre</b> total est bon</li> <li>- Vérifie que les <i>nomGenre</i> de chacun d'eux est bon</li> </ul>
T_24	Test_GetParAlbumNom()	GenreRepository	GetParAlbumNom()	Vérifie que la fonction retourne les bons <b>genres</b> pour un <b>album</b> donné. <ul style="list-style-type: none"> <li>- vérifie que le nombre de genre est correct</li> <li>- vérifie les <i>nomGenre</i> de chacun</li> </ul>
T_25	Test_GetParAlbumNom_Erreur()	GenreRepository	GetParAlbumNom()	Vérifie que la fonction ne retourne aucun <b>genre</b> dans le cas d'un <i>nomAlbum</i> erroné. <ul style="list-style-type: none"> <li>- vérifie que le nombre de <b>genre</b> retourné est nul</li> </ul>
T_26	Test_Save()	SerieRepository	Save()	Vérifie qu'une nouvelle <b>serie</b> est bien sauvegardé en BD : <ul style="list-style-type: none"> <li>- vérifie que le nombre de <b>séries</b> total est bon</li> <li>- vérifie que les <i>nomSerie</i> correspondent</li> </ul>

T_27	Test_GetAll()	SerieRepository	GetAll()	Vérifie que la fonction retourne toutes les <b>séries</b> de la base de données : <ul style="list-style-type: none"> <li>- vérifie que le nombre est celui attendu</li> <li>- Vérifie que les <i>nomSerie</i> sont ceux attendus</li> </ul>
T_28	Test_GetAll()	UtilisateurRepository	GetAll()	Vérifie que la fonction retourne tous les <b>utilisateurs</b> de la base de données : <ul style="list-style-type: none"> <li>- vérifie que le nombre est celui attendu</li> <li>- Vérifie que les <i>nom</i> sont ceux attendus</li> </ul>
T_29	Test_GetUtilisateurParId()	UtilisateurRepository	GetUtilisateurParId()	Vérifie que la fonction retourne le bon <b>utilisateur</b> pour un <i>id</i> donné. (même <i>nom</i> )
T_30	Test_GetUtilisateurParId_Erreur()	UtilisateurRepository	GetUtilisateurParId()	Vérifie que la fonction ne retourne aucun <b>utilisateur</b> dans le cas d'un <i>id</i> erroné.
T_31	Test_GetUtilisateurParLoginMdp()	UtilisateurRepository	GetUtilisateurParLoginMdp()	Vérifie que la fonction retourne le bon <b>utilisateur</b> pour un <i>login</i> et un <i>mdp</i> donné. (même <i>id</i> )
T_32	Test_GetUtilisateurParLoginMdp_Erreur()	UtilisateurRepository	GetUtilisateurParLoginMdp()	Vérifie que la fonction ne retourne aucun <b>utilisateur</b> dans le cas d'une association <i>login/mdp</i> erroné.
T_33	Test_Save()	UtilisateurRepository	Save()	Vérifie qu'un nouvel <b>utilisateur</b> est bien inscrit en BD : <ul style="list-style-type: none"> <li>- vérifie le nombre total d'<b>utilisateurs</b></li> <li>- vérifie que les <i>nom</i> sont les bons.</li> </ul>
T_34	Test_Save_Erreur()	UtilisateurRepository	Save()	Vérifie qu'un nouvel <b>utilisateur</b> ayant un <i>login</i> déjà utilisé ne

				soit pas enregistré en BD. <ul style="list-style-type: none"> <li>- vérifie le nombre total d'<b>utilisateurs</b></li> <li>- vérifie que les <i>nom</i> sont les bons.</li> </ul>
T_35	Test_Save_ErreurAdmin()	UtilisateurRepository	Save()	Vérifie qu'un nouvel <b>utilisateur</b> ayant un <i>login</i> déjà utilisé d'administrateur ne soit pas enregistré en BD. <ul style="list-style-type: none"> <li>- vérifie le nombre total d'<b>utilisateurs</b></li> <li>- vérifie que les <i>nom</i> sont les bons.</li> </ul>
T_36	Test_GetCollectionUtilisateur()	UtilisateurRepository	GetCollectionUtilisateur	Vérifie que la fonction retourne les <b>albums</b> de la collection de l' <b>utilisateur</b> . <ul style="list-style-type: none"> <li>- vérifie que le nombre est le bon</li> <li>- vérifie que les <i>nomAlbum</i> sont les bons.</li> </ul>
T_37	Test_GetSouhaitsUtilisateur()	UtilisateurRepository	GetSouhaitsUtilisateur()	Vérifie que la fonction retourne les <b>albums</b> souhaités par l' <b>utilisateur</b> . <ul style="list-style-type: none"> <li>- vérifie que le nombre est le bon</li> <li>- vérifie que les <i>nomAlbum</i> sont les bons.</li> </ul>
T_38	Test_AjouterAlbumSouhait()	UtilisateurRepository	AjouterAlbumSouhait()	Vérifie que la fonction enregistre bien en BD l' <b>album</b> que souhaite l' <b>utilisateur</b> : <ul style="list-style-type: none"> <li>- vérifie que le nombre est le bon</li> <li>- vérifie que les <i>nomAlbum</i> sont les bons.</li> </ul>
T_39	Test_AjouterAlbumSouhait_Erreur()	UtilisateurRepository	AjouterAlbumSouhait()	Vérifie que si l' <b>album</b> est déjà dans la liste de souhaits de l' <b>utilisateur</b> , celui-ci n'est pas ajouté.

				<ul style="list-style-type: none"> <li>- vérifie que le nombre est le bon</li> <li>- vérifie que les <i>nomAlbum</i> sont les bons.</li> </ul>
T_40	Test_AjouterAlbumSouhaiteCollec()	UtilisateurRepository	AjouterAlbumCollec()	<p>Vérifie que la fonction enregistre bien en BD l'<b>album</b> qu'a acquis l'<b>utilisateur</b> :</p> <ul style="list-style-type: none"> <li>- vérifie que le nombre est le bon</li> <li>- vérifie que les <i>nomAlbum</i> sont les bons.</li> </ul> <p>Vérifie que la liste de souhait été modifiée en conséquence (suppression du souhait)</p> <ul style="list-style-type: none"> <li>- vérifie que le nombre est le bon</li> <li>- vérifie que les <i>nomAlbum</i> sont les bons.</li> </ul>
T_41	Test_AjouterAlbumSouhaiteCollec_Erreur()	UtilisateurRepository	AjouterAlbumCollec()	<p>Vérifie que si l'<b>album</b> est déjà dans la liste de collection de l'<b>utilisateur</b>, celui-ci n'est pas ajouté.</p> <ul style="list-style-type: none"> <li>- vérifie que le nombre est le bon</li> <li>- vérifie que les <i>nomAlbum</i> sont les bons.</li> </ul>
T_42	Test_AjouterAlbumCollec()	UtilisateurRepository	Test_AjouterAlbumCollec()	<p>Vérifie que la fonction enregistre bien en BD l'<b>album</b> qu'a acquis l'<b>utilisateur</b> :</p> <ul style="list-style-type: none"> <li>- vérifie que le nombre est le bon</li> <li>- vérifie que les <i>nomAlbum</i> sont les bons.</li> </ul>
T_43	Test_SupprimerAlbumUtilisateur()	UtilisateurRepository	SupprimerAlbumUtilisateur()	<p>Vérifie que l'<b>album</b> passé en paramètre est bien supprimé de la collection de l'<b>utilisateur</b></p>

				<ul style="list-style-type: none"> <li>- vérifie que le nombre est le bon</li> </ul>
T_44	Test_SupprimerAlbumUtilisateur_Erreur	UtilisateurRepository	SupprimerAlbumUtilisateur()	Vérifie que l' <b>album</b> passé en paramètre n'étant pas dans sa collection n'est pas supprimé <ul style="list-style-type: none"> <li>- vérifie que le nombre est le bon</li> </ul>
T_45	Test_SupprimerAlbumSouhaitUtilisateur	UtilisateurRepository	SupprimerAlbumSouhaitUtilisateur()	Vérifie que l' <b>album</b> passé en paramètre est bien supprimé de la collection de l' <b>utilisateur</b> <ul style="list-style-type: none"> <li>- vérifie que le nombre est le bon</li> <li>- vérifie que les <i>nomAlbum</i> sont les bons.</li> </ul>
T_46	Test_SupprimerAlbumSouhaitUtilisateur_Erreur()	UtilisateurRepository	SupprimerAlbumSouhaitUtilisateur()	Vérifie que l' <b>album</b> passé en paramètre n'étant pas dans sa liste de souhait n'est pas supprimé <ul style="list-style-type: none"> <li>- vérifie que le nombre est le bon</li> </ul>
T_47	ToStringTest	Album	ToString()	S'assure que la fonction renvoie la bonne chaîne de caractère.
T_48	ToStringTest	Auteur	ToString()	S'assure que la fonction renvoie la bonne chaîne de caractère.
T_49	ToStringTest	Categorie	ToString()	S'assure que la fonction renvoie la bonne chaîne de caractère.
T_50	ToStringTest	Genre	ToString()	S'assure que la fonction renvoie la bonne chaîne de caractère.
T_51	ToStringTest	Serie	ToString()	S'assure que la fonction renvoie la bonne chaîne de caractère.
T_52	ToStringTest	Utilisateur	ToString()	S'assure que la fonction renvoie la bonne chaîne de caractère.





# CONCLUSION

## Difficultés rencontrées

Au cours de ce projet, nous avons rencontré plusieurs difficultés. Tout d'abord, il nous a fallu un certain temps avant d'appréhender la bonne méthode de réflexion quant à l'accès aux données, particulièrement pour les tables de jointure. En effet, nous étions focalisés sur un schéma de pensée axé base de données, comme nous en avons l'habitude. Cependant, avec Nhibernate, il est essentiel d'appréhender l'accès aux données en réfléchissant "objet". Lorsque ce déclic s'est opéré, nous avons enfin pu réaliser les méthodes suivant la logique de `GetCollectionUtilisateur()` par exemple (dans `UtilisateurRepository`). Effectivement, il nous a fallu du temps avant de trouver le passage par notre attribut `ListeUtilAchetes` avec la méthode `Any()`, pour pouvoir accéder à l'attribut `id` de l'objet Utilisateur contenu à l'intérieur de la liste.

À cette difficulté s'étend un autre, celle de la suppression d'un objet.

Vraisemblablement, nous voulions utiliser la fonction `Delete()` disponible avec LINQ, cependant cela ne fonctionnait pas, nous avons donc dû nous adapter en utilisant `SaveOrUpdate()` sur la liste des objets existants avec celui que nous voulions supprimer en moins.

## Perspectives & améliorations

Après avoir fait le constat des exigences fonctionnelles et non fonctionnelles remplies par notre application, nous avons relevé des points d'amélioration à apporter.

Tout d'abord, nous sommes assez frustrés de ne pas avoir réussi à obtenir un comportement couvrant toutes les possibilités concernant notre recherche d'album avec des critères. En effet, avec le critère "Auteur", dans le cas où l'utilisateur écrirait le nom ET le prénom dans le champ de recherche, aucun album n'est trouvé. Cela ne fonctionne que si seul le nom OU le prénom est entré, entièrement ou partiellement. En effet, après maints essais sur une concaténation de chaînes, ou division de sous-chaînes, nous n'avons pas réussi à ce que cela fonctionne parfaitement (nous étions arrivés laborieusement à un résultat intermédiaire qui permettait d'entrer le nom et le prénom et de trouver un album, mais cela était sensible à la casse, ce qui n'est pas satisfaisant).

Exemples pour illustrer avec l'auteur portant le nom Tardi et le prénom Jacques :

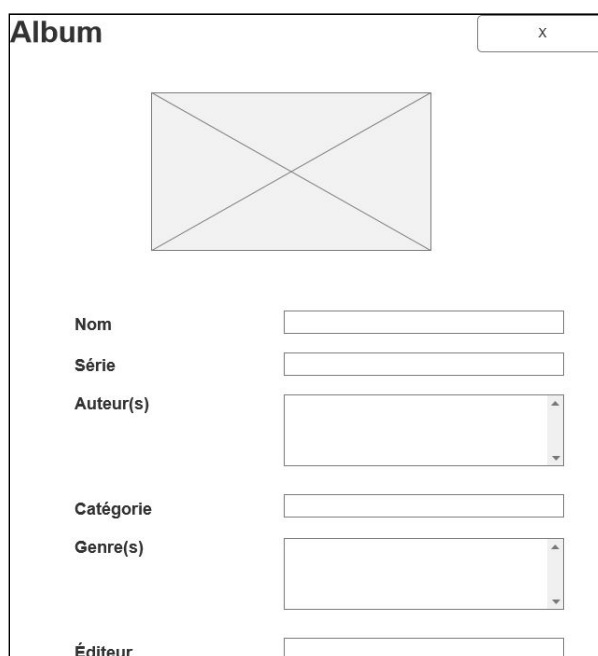
Il est possible d'entrer les lettres composant soit le nom soit le prénom de Jacques (par exemple on peut entrer juste "jac" et il trouve les bons albums) tant qu'elles sont dans le bon ordre (il ne sera pas possible d'écrire "jques"). Et comme nous l'avons expliqué dans le paragraphe précédent, il ne sera pas possible d'écrire "Jacques Tardi" ou une variante équivalente.

D'autres améliorations d'un point de vue fonctionnalité seraient de permettre de trier les objets (albums, auteurs, etc) par ordre alphabétique, ou par d'autres critères supplémentaires. Du côté de l'administrateur, il aurait été intéressant qu'il puisse, en plus d'ajouter des objets (genre, auteur, album, ...) en supprimer ou en modifier.

Une autre perspective d'avancée dans ce projet serait les feedbacks utilisateurs. En effet, nous avons essayé d'en ajouter autant que possible (surtout lorsqu'on ajoute un objet en base de données pour vérifier que l'action s'est bien passée), mais nous avons été contraints de passer par des pop-up, ce qui n'est pas forcément le plus agréable dynamique dans l'expérience utilisateur.

## ANNEXES

### Annexe 1 : Maquette visualisation Album



Album [X]

[Image placeholder: A rectangle with a diagonal cross, indicating a missing image.]

Nom

Série

Auteur(s)

Catégorie

Genre(s)

Éditeur

### Annexe 2 : Maquette Formulaire d'inscription



BDThèque [X]

**Formulaire d'inscription**

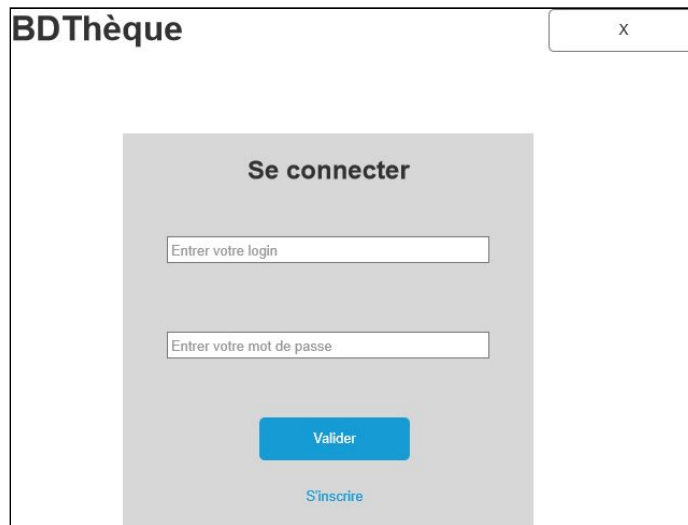
Login

Mot de passe

Nom

Prénom

## Annexe 3 : Maquette Connexion



BDThèque

**Se connecter**

Entrer votre login

Entrer votre mot de passe

Valider

S'inscrire

## Annexe 4 : Maquette Profil Utilisateur



Profil

**Profil Utilisateur**

Nom

Prénom

OK

## Annexe 5 : Maquette BDTheque : écran principal

