UE : Sciences fondamentales

Module : Génie Logiciel

Projet : BDthèque

Professeurs : Baptiste Pesquet,
Sébastien Bertrand

Étudiantes : Marion Othéguy,
Maika Touzet

Répertoire github : https://github.com/ensc-glog/projet-2020-marion-maika

Date de rendu : 15 janvier 2021, 23h59

Table des matières

I. Liste des exigences métier respectées par l'application

L'objectif de ce projet est de créer une application WinForms de gestion de collections de BDs. Cette application doit répondre aux exigences fonctionnelles ci-dessous.

Code	Description	Implémentation
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.	✔ Ajout un par un
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.	✔ Ajout un par un
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.	✔

Exigences fonctionnelles supplémentaires ajoutées aux consignes :

EF_12	En tant qu'administrateur, je peux supprimer un album de la liste des albums du marché	✔
EF_13	En tant qu'utilisateur, je peux effectuer une recherche dans ma liste de BD possédées. Cette recherche peut être basée sur les critères suivants : nom (ou partie du nom), série, auteur, genre.	✔
EF_14	En tant qu'utilisateur, je peux effectuer une recherche dans ma liste de BD souhaitées. Cette recherche peut être basée sur les critères suivants : nom (ou partie du nom), série, auteur, genre.	✔

II. Démarche de conception de l'IHM

En raison d'un temps limité, nous n'avons pas pu établir une démarche IHM complète (maquettage et tests utilisateurs). Cependant, nous avons veillé à ce que l'interface soit compréhensible et intuitive.

Tout d'abord, nous avons créé une interface proche de ce qui existe déjà et est généralement utilisée. La page de connexion est notamment proche des espaces de connexion usuels. La seule différence ici est la case " Administrateur " qui permet à l'utilisateur d'indiquer s'il a un compte administrateur ou pas. Nous avons hésité à la mettre, de peur de compliquer l'interface. Cependant, elle permet de pouvoir choisir l'interface souhaitée dans le cas où un compte administrateur et un compte utilisateur ont les mêmes identifiants (une personne ayant les deux types de comptes par exemple).

Nous avons aussi utilisé des termes explicites pour chaque bouton (" acheter ", " ajouter à mes souhaits ", …). Ceci limite le risque de mauvaises manipulations (suppression ou achat involontaire, …).

L'affichage des informations est ordonné : la recherche s'effectue en haut, une boîte séparée sur la gauche affiche la liste des albums et les informations concernant un album sont regroupées et rangées.

Enfin, afin de s'assurer que toutes les informations restent affichées et visibles sur une page, nous avons réglé une taille minimale pour la fenêtre. Il n'est pas possible pour l'utilisateur de réduire et donc de perdre des informations.

III. Modélisation des données

A. Modèle relationnel des données

Pour modéliser la base de données, nous avons décidé de créer deux tables principales : `individu` et `album`. Ces tables sont reliées par `bdsouhait` et `bdpossession`, qui prennent chacune un couple (individu, album) qui représente respectivement un album apparaissant sur la liste de souhaits de l'individu en question ou un album lui appartenant. Cette architecture est représentée en **Figure 1**.

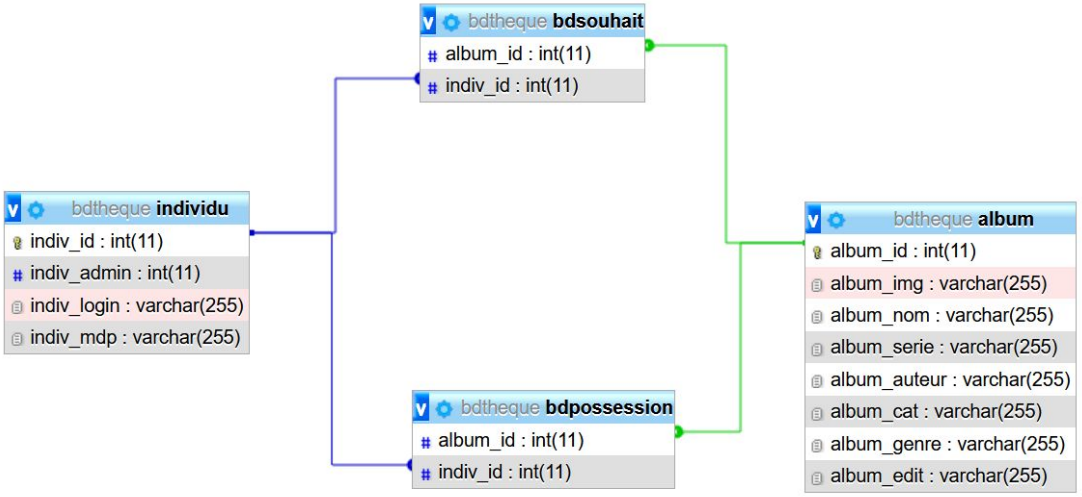

Figure 1 : Architecture de la base de données `bdtheque`

B. Diagramme des classes métier

Pour ce qui est du diagramme des classes, nous avons fait le choix de créer une classe abstraite `Individual` (correspondant à la table `individu` de la base de données), de laquelle héritent deux classes : `Admin` et `User`. Pour un individu, si `indiv_admin=1`, alors on créera un objet `Admin`, si `indiv_admin=0`, il n'est pas administrateur et on créera un objet `User`. Nous avons aussi créé une classe `Album` correspondant à la table homonyme. Les tables `bdsouhait` et `bdpossession` sont représentées par des associations n à n entre `User` et `Album`. En effet, un administrateur ne possède pas d'album ou de liste de souhait, l'association n'a donc pas été faite avec la classe abstraite `Individual`.
La modélisation est visible en **Figure 2**.

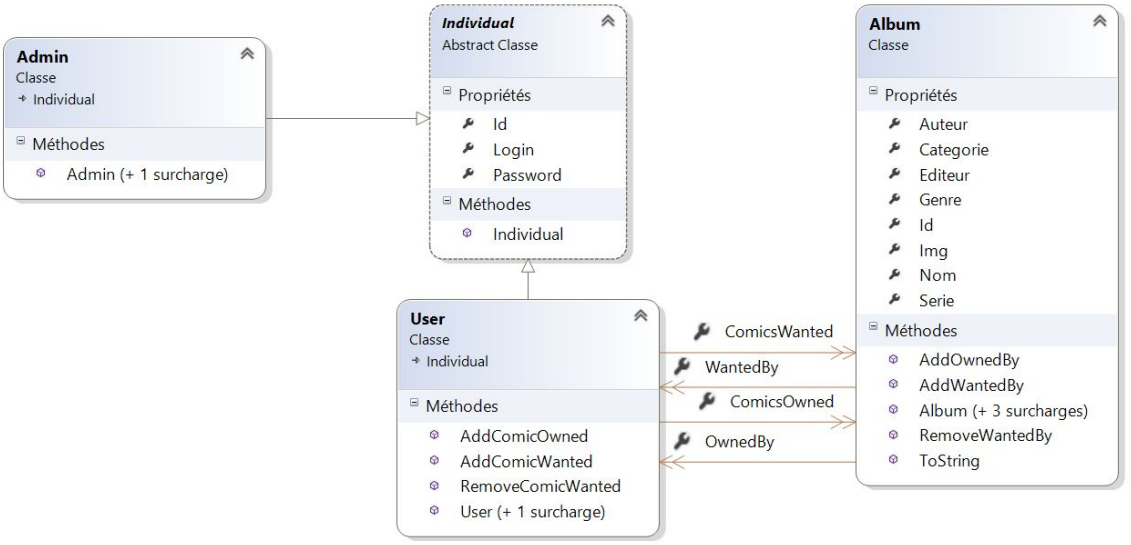

Figure 2 : Diagramme des classes métiers de la solution

Ces classes ont été traduites en deux répertoires : `AlbumRepository` pour `Album` et `IndividualRepository` pour `Individual`. Chacun hérite simultanément d'une classe abstraite principale (`Repository`) et d'une interface (`IAlbumRepository` ou `IIndividualRepository`). Cette architecture est visible en **Figure 3**.

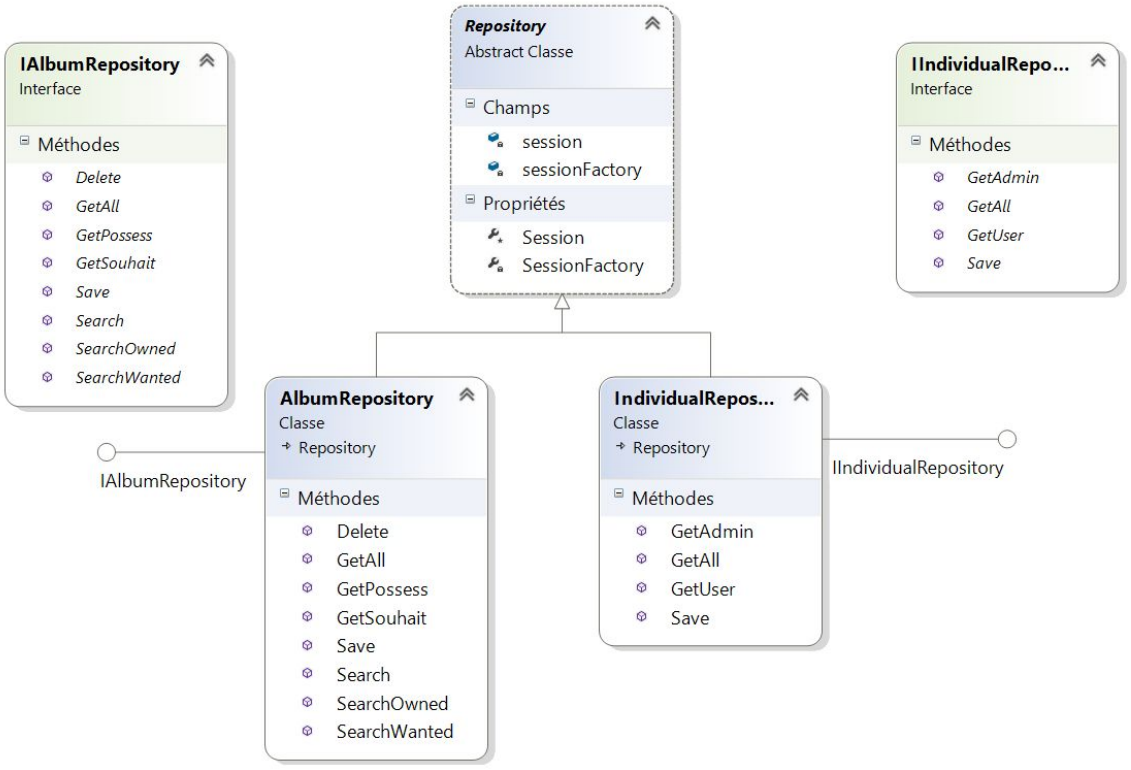

Figure 3 : Diagramme des classes de DAL

IV. Architecture technique de l'application

Pour réaliser cette application, nous avons utilisé le framework NHibernate. Notre BDthèque répond à une architecture en couches (APP/ DAL / Domain) conformément à l'exigence technique ET_03.

A. App

Cette partie contient tout ce qui est relatif à l'interface utilisateur, à savoir :

- ConnectionForm : page de connexion pour un utilisateur ou un administrateur. C'est la page principale de l'application. Elle est cachée lorsque la connexion réussit et qu'une autre page est ouverte. Elle est à nouveau visible lorsque l'individu se déconnecte
- ConsultUserForm : interface de l'utilisateur. Elle permet de consulter le catalogue, ainsi que les albums possédés et souhaités par l'utilisateur qui est connecté.
- ConsultAdminForm : interface de l'administrateur. Elle permet de consulter le catalogue, de supprimer un album ou d'ouvrir AddAlbumForm.
- AddAlbumForm : interface d'ajout d'album. Elle s'ouvre en fenêtre pop-up depuis l'interface administrateur et permet la création d'un album.

B. DAL

Ce répertoire permet de faire la liaison avec la base de données, grâce aux fichiers SQL et aux repositories.

C. Domain

Ce répertoire contient les classes métiers `Album.cs`, `Admin.cs`, `Individual.cs` et `User.cs`, ainsi que les fichiers de mapping.

D. Tests unitaires

Les tests des méthodes des répertoires DAL et Domain sont contenus respectivement dans les répertoires DALTests et DomainTests. Ils sont détaillés dans la partie X *" Liste et description des procédures de test automatisé ".*

E. InitDB

Ce programme permet d'initialiser la base de données. Il est à lancer avant l'utilisation de l'application comme expliqué dans la partie suivante : "*Procédure d'installation détaillée*".

V. Procédure d'installation détaillée

Tout d'abord veuillez cloner le projet présent à l'adresse suivante : https://github.com/ensc-glog/projet-2020-marion-maika .

Une fois cela fait, ouvrez le logiciel xampp et activez les deux entités suivantes Apache et mySQL.
Appuyez sur Admin de mySQL pour vous rendre sur phpMyAdmin, nous allons maintenant créer la base de données. Tout d'abord commençons par la *database* elle-même, pour cela deux fois s'offre à vous :

- importez le fichier `Database.sql` présent dans `.../projet-2020-marion-maika/BDtheque/DAL/DB/` en faisant ….
- ou bien copier le contenu de ce fichier dans la console de requête SQL et exécutez là.

```
create database if not exists bdtheque character set utf8 collate
utf8_unicode_ci;
use bdtheque;

grant all privileges on bdtheque.* to 'bdtheque_user'@'localhost'
identified by 'secret';
```

Figure 4 : contenu de `Database.sql`

Une fois la base de données instaurée, initiez le contenu de celle-ci. Pour cela, deux choix se présentent : une manœuvre manuelle ou une manœuvre semi-automatisée, nous vous préconisons l'installation semi-automatisée.

- Installation manuelle

Importer les fichiers `Structure.sql` et `Content.sql` l'un après l'autre en cliquant sur l'option *Importer* de phpMyAdmin.

- Installation semi-automatisée

Ouvrez la solution `BDtheque.sln` présente dans `.../projet-2020-marion-maika/BDtheque` et lancez le projet `InitDB`.

La base de données et son contenu sont maintenant instaurés.

Afin de pouvoir charger les images dans le catalogue veuillez déplacer le dossier `img` présent dans `.../projet-2020-marion-maika/BDtheque` dans le dossier `Debug` du projet `App`, c'est à dire dans `.../projet-2020-marion-maika/BDtheque/App/bin/Debug`

Puis pour lancer la BDthèque, lancez le projet `App` depuis la solution `BDtheque.sln` .

Afin de se connecter il existe 4 comptes individus “sécurisés” par un mot de passe chacun :

Login	Mot de passe	Utilisateur	Administrateur
bpesquet	incognito		X
motheguy	1234		X
sbertrand	abcd	X	
mtouzet	mdp	X	

VI. Répartition des tâches dans le groupe et planning

Les grandes tâches définies sont les suivantes :

- Initialisation de la solution adoptant l’architecture MVP
- Création de la base de donnée
- Création des classes métiers
- Création et gestion des Repository
- Implémentation des interfaces (Connexion, interface utilisateur, interface administrateur)
- Tests de automatisés d’accès aux données et de classes métier (DAL)

La répartition entre les membres de l’équipe et dans le temps est visible sur le diagramme de Gantt en **Figure 5**.

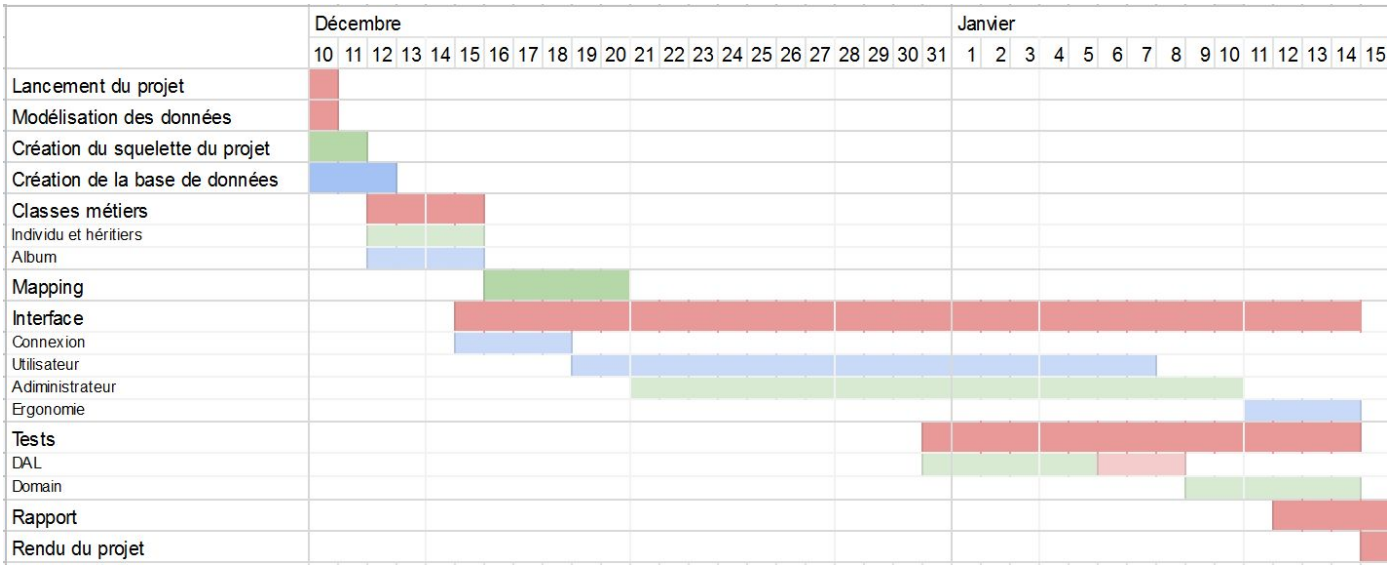

Figure 5 : Planning réel pour la réalisation du projet BDthèque
(légende de couleur - en vert : tâches effectuées par Marion, en bleu : Maika, en rouge : les deux)

VII. Liste et description des procédures de test automatisé

A. Test du projet DAL

- `AlbumRepositoryTests.cs`

Outil testée pour l'exigence	Méthode de test	Description du test
EF_03	GetAllTest()	Vérifie que l'on récupère tous les albums du répertoire : c'est le même le même nombre d'albums que l'on retrouve dans le répertoire et grâce à la méthode GetAll().
EF_02	GetPossessTest()	Vérifie que l'on récupère tous les albums possédé par un utilisateur donné : c'est le même le même nombre d'albums ajoutés que l'on retrouve dans la liste possédé et grâce à la méthode GetPossess().
EF_07	GetSouhaitTest()	Vérifie que l'on récupère tous les albums souhaités par un utilisateur donné : c'est le même le même nombre d'albums ajoutés que l'on retrouve dans la liste possédé et grâce à la méthode GetSouhait().
EF_04	SearchTest()	Vérifie que l'on arrive à retrouver un album à partir d'un mot clé dans le catalogue général grâce à la méthode Search().
EF_13	SearchOwnedTest()	Vérifie que l'on arrive à retrouver un album à partir d'un mot clé dans le catalogue des albums possédés par l'utilisateur connecté grâce à la méthode SearchOwned().
EF_14	SearchWantedTest()	Vérifie que l'on arrive à retrouver un album à partir d'un mot clé dans le catalogue des albums possédés par l'utilisateur connecté grâce à la méthode SearchWanted().
EF_11	SaveTest()	Vérifie que la fonction Save() ajoute bien un album dans AlbumRepository
EF_12	DeleteTest()	Vérifie que la fonction Delete() supprime bien un album de AlbumRepository

- `IndividualRepositoryTests.cs`

Outil testée pour l'exigence	Méthode de test	Description du test
	SaveTest()	Vérifie que la fonction Save() ajoute bien un individu dans IndividuRepository
	GetAllTest()	Vérifie que l'on récupère tous les individus du répertoire : c'est le même le même nombre d'individus que l'on retrouve dans le répertoire et grâce à la méthode GetAll().
EF_01	GetUserTest()	Vérifie que l'on récupère tous les utilisateurs du répertoire : c'est le même le même nombre d'utilisateurs que l'on retrouve dans le répertoire et grâce à la méthode GetUserAll().
EF_10	GetAdminTest()	Vérifie que l'on récupère tous les administrateurs du répertoire : c'est le même le même nombre d'administrateurs que l'on retrouve dans le répertoire et grâce à la méthode GetAdminAll().

B. Test du projet Domain

- `AdminTests.cs`

Pas de méthodes à tester.

- `AlbumTests.cs`

Outil testée pour l'exigence	Méthode de test	Description du test
EF_06	AddWantedBy_Test1()	Vérifie l'ajout d'un utilisateur dans la liste d'utilisateur possédant la BD depuis la classe Album : c'est le même objet qui est inséré et celui que l'on retrouve dans la liste.
EF_06	AddWantedBy_Test2()	Vérifie l'ajout d'un utilisateur dans la liste d'utilisateur possédant la BD depuis la classe Album : le nombre de BD dans la liste est le même nombre que de BD insérés
EF_08	RemoveWantedBy_Test1()	Vérifie la suppression d'un utilisateur dans la liste d'utilisateur possédant la

		BD depuis la classe Album : le premier objet inséré une fois supprimé est remplacé par le second objet en première position de la liste de BD possédées
EF_08	RemoveWantedBy_Test2()	Vérifie la suppression d'un utilisateur dans la liste d'utilisateur possédant la BD depuis la classe Album : il n'y a plus d'objet dans la liste de BD possédé une fois ceux ci supprimés
EF_05	AddOwnedBy_Test1()	Vérifie l'ajout d'un utilisateur dans la liste d'utilisateur souhaitant la BD depuis la classe Album : c'est le même objet qui est inséré et celui que l'on retrouve dans la liste.
EF_05	AddOwnedBy_Test2()	Vérifie l'ajout d'un utilisateur dans la liste d'utilisateur souhaitant la BD depuis la classe Album : le nombre de BD dans la liste est le même nombre que de BD insérés

- `IndividualTests.cs`

Pas de méthodes à tester.

- `UserTests.cs`

Outil testée pour l'exigence	Méthode de test	Description du test
EF_05	AddComicOwned_Test1()	Vérifie l'ajout d'une BD dans la liste de BD possédées par l'utilisateur depuis la classe User : c'est le même objet qui est inséré et celui que l'on retrouve dans la liste.
EF_05	AddComicOwned_Test2()	Vérifie l'ajout d'une BD dans la liste de BD possédées par l'utilisateur depuis la classe User : l'objet inséré possède l'utilisateur dans sa liste d'utilisateurs le possédant
EF_05	AddComicOwned_Test3()	Vérifie l'ajout d'une BD dans la liste de BD possédées par l'utilisateur depuis la classe User : la BD est ajouté en un unique exemplaire
EF_05	AddComicOwned_Test4()	Vérifie l'ajout d'une BD dans la liste

		de BD possédées par l'utilisateur depuis la classe User : l'utilisateur ajouté dans la liste des utilisateurs possédant la BD y est présent en un unique exemplaire
EF_06	AddComicWanted_Test1()	Vérifie l'ajout d'une BD dans la liste de BD voulues par l'utilisateur depuis la classe User : c'est le même objet qui est inséré et celui que l'on retrouve dans la liste
EF_06	AddComicWanted_Test2()	Vérifie l'ajout d'une BD dans la liste de BD voulues par l'utilisateur depuis la classe User : l'objet inséré possède l'utilisateur dans sa liste d'utilisateurs le souhaitant
EF_06	AddComicWanted_Test3()	Vérifie l'ajout d'une BD dans la liste de BD voulues par l'utilisateur depuis la classe User : la BD est ajouté en un unique exemplaire
EF_06	AddComicWanted_Test4()	Vérifie l'ajout d'une BD dans la liste de BD voulues par l'utilisateur depuis la classe User : l'utilisateur ajouté dans la liste des utilisateurs souhaitant la BD y est présent en un unique exemplaire
EF_08	RemoveComicWanted_Test1()	Vérifie la suppression d'une BD dans la liste de BD voulues par l'utilisateur depuis la classe User : l'album est supprimé de la liste
EF_08	RemoveComicWanted_Test2()	Vérifie la suppression d'une BD dans la liste de BD voulues par l'utilisateur depuis la classe User : la suppression affecte qu'un album de la liste de souhait de l'utilisateur

VIII. Bilan et perspectives sur le projet

Le projet ayant été conçu avec une relation n:n entre les utilisateurs et les albums, nous avions envisagé pouvoir retrouver les personnes possédant ou souhaitant les albums à partir de ceux-ci. Cela nous aurait permis d'avoir l'information des personnes possédant la BD présente dans le catalogue, bien que ces informations soient trouvables par une requête HQL. Cela nous permettrait dans le cas futur d'une amélioration de créer une fonctionnalité de gestion des prêts de BD entre utilisateurs.

Les principales difficultés rencontrées sont liées à l'appréhension de la technologie NHibernate, cependant ce projet nous a permis de mieux comprendre celle-ci. Par exemple, en créant le projet InitDB, il nous a été difficile de concevoir que l'on ne pouvait pas insérer notre fichier SQL Content.sql tel quel. En effet, il faut que la requête SQL via Hibernate se rapproche du mapping. Nous avons utilisé l'implémentation des repository pour permettre l'installation de la base de données via InitDB.

De plus, il nous a fallu du temps de débogage avant de comprendre que la liaison entre les classes User et Album n'est possible que si les listes d'albums ou d'individus sont de type IList. En effet, le IList permet de manipuler les objets Nhibernate.

Avant de conclure avec ce projet, il nous faut signaler que la fonctionnalité d'ajout d'un album au catalogue depuis l'interface administrateur présente une erreur lors de l'import d'une photo. Cet import n'est pas nécessaire à l'ajout d'une nouvelle BD. Nous n'avons pas pu identifier la source de l'erreur, le programme prenant certaines photos et pas d'autres.