

Atelier de professionnalisation 3

Application C# - Gestion de la médiathèque

Compte rendu



Carl Fremault - 2ième année BTS SIO - Mars 2022

Sommaire

1. Accès à l'application.....	3
2. Contexte.....	4
3. Présentation des missions.....	4
3.1 Mission 1 (optionnelle) : Gestion des documents.....	4
3.2 Mission 2 : Gestion des commandes.....	5
3.3 Mission 3 (optionnelle) : Gérer le suivi de l'état des documents.....	5
3.4 Mission 4 : Mettre en place des authentifications.....	6
3.5 Mission 5 : Qualité, tests et documentation technique.....	6
4. Ressources logicielles utilisées.....	6
5. Mission 0 : Préparer l'environnement de travail.....	7
6. Mission 1 (optionnelle) : Gestion des documents.....	12
6.1 Ajout des fonctionnalités pour ajouter/modifier/supprimer des Livres, DVD, Revues.....	13
6.2 Mises en place des sécurités pour éviter des erreurs de manipulation.....	15
6.3 Implémentation d'ajout et de modification des livres.....	22
6.4 Suppression de livres.....	33
6.5 Implémentation des fonctionnalités ajout/modification/suppression pour les onglets DVD et Revue.....	37
6.6 Création de trigger pour contrôler les contraintes de partition d'héritage sur Document et LivresDvd.....	38
6.7 Bilan pour la mission 1.....	42
7. Mission 2 : Gérer les commandes.....	43
7.1 Création de table 'suivi' dans la bdd.....	45
7.2 Création d'onglets pour gérer les commandes de livres et de DVD, et abonnements de revues. Implémentation zones recherche.....	47
7.3 Implémentation de zones d'ajout/modification/suppression d'une commande ou d'un abonnement.....	51
7.4 Création de trigger qui ajoute les exemplaires lorsqu'une commande est livrée.....	63
7.5 Création de trigger pour contrôler la contrainte de partition de l'héritage sur commande.....	65
7.6 Création de procédure stockée d'alerte de fin d'abonnements.....	66
7.7 Création de fenêtre d'alerte de fin d'abonnements au démarrage de l'application.....	67
7.8 Bilan pour la mission 2.....	73
8. Mission 4 : Mettre en place des authentifications.....	75
8.1 Création des tables Utilisateur et Service dans la base de données.....	76
8.2 Ajout de fenêtre d'authentification.....	78
8.3 Gérer les accès aux fonctionnalités suivant le service de l'utilisateur.....	84
8.4 Bilan pour la mission 4.....	86
9. Mission 5 : Qualité, tests et documentation technique.....	87
9.1 Contrôle de qualité de code avec SonarLint.....	88
9.2 Création de tests fonctionnels avec SpecFlow.....	92
9.3 Journalisation pour récupérer les erreurs dans les blocs try/catch.....	96
9.4 Génération de la documentation technique.....	100
9.5 Bilan pour la mission 5.....	102
10. Déploiement de l'application.....	102
11. Bilan final.....	106
12. Compétences mobilisées pendant cet Atelier.....	107

1. Accès à l'application

Installation

- Télécharger le fichier 'Mediatek86.zip' depuis mon site portfolio
- Installer l'application avec l'installateur 'Mediatek86 Installer.msi'
- Placer le dossier 'MediatekImages' en racine du c: (c:\MediatekImages)

Comptes Utilisateurs :

Compte administrateur : tous les droits

- Nom d'utilisateur : admin
- Mot de passe : admin

Compte Services Administratifs : tous les droits

- Nom d'utilisateur: serviceadmin
- Mot de passe: serviceadmin

Compte Service Prêts : droit de consultation des catalogues livres, DVD, revues et exemplaires revues

- Utilisateur : pret
- Mot de passe : pret

Compte Service Culture : aucun droit, pas d'accès à l'application

- Utilisateur : culture
- Mot de passe : culture

Accès à la base de données sur Azure (avec par exemple MySQL Workbench, ...)

- Adresse : mediatek86.mysql.database.azure.com
- Nom d'utilisateur administration base de données : mediadmin
- Mot de passe : Gestion86

Le script de la base de données est disponible dans le dépôt Github, dans le dossier 'script bdd'.

2. Contexte

Les médiathèques MediaTek86 disposent d'une application qui permet de gérer les différents documents proposés : livres, DVD, Revues. Pour l'instant l'application permet de consulter le catalogue avec des fonctionnalités tel la recherche (sur le nom ou le numéro d'un document), le filtrage (sur un genre, public ou rayon). Pour chaque document il est possible de visualiser des détails (auteur, code ISBN, synopsis, ...) en fonction du type de document. Il est également possible d'ajouter des nouvelles parutions de revues.

Dans cet atelier je vais m'occuper des différents besoins formulés afin de faire évoluer cette application.

Cet atelier volumineux est conçu pour être réalisé par une équipe de développeurs. Toutefois il est possible de le réaliser seul, option que j'ai retenue. Dans ce cas deux des missions sont optionnelles : une première mission, 'Gestion des documents' que j'ai quand même choisi de faire, puis une deuxième mission où il est demandé de gérer l'état d'usure des documents, que je n'ai pas retenue.

3. Présentation des missions

3.1 Mission 1 (optionnelle) : Gestion des documents

Pour la première mission, il est demandé d'ajouter des boutons dans chaque onglet (Livres, DVD, Revues) qui permettent d'ajouter, modifier ou supprimer un document. Quelques contraintes sont à respecter : il ne doit pas être possible de supprimer un document s'il est lié à un exemplaire ou une commande. On ne doit pas pouvoir modifier l'identifiant d'un document. En plus il est important de mettre en place les sécurisations nécessaires pour éviter des erreurs de manipulation.

Un trigger doit être créé dans la base de donner afin de vérifier le respect de la contrainte de partition de l'héritage pour les documents (soit une revue, soit un 'Livres-DVD') et pour les 'Livres-DVD' (soit un livre, soit un DVD).

Comme j'effectue ce travail de groupe seul, cette mission est marqué comme étant optionnelle. Elle me semble intéressante cependant, et la possibilité d'ajouter, modifier et supprimer des documents sera pratique pour la suite de cet atelier. Ainsi j'ai choisi de prendre en charge cette mission.

3.2 Mission 2 : Gestion des commandes

La deuxième mission consiste à ajouter des fonctionnalités pour pouvoir commander des livres / DVD ou souscrire à des abonnements pour des revues. On demande de faire un onglet pour chaque cas qui permet de chercher le document pour lequel on souhaite passer commande (ou souscrire) et qui affiche les commandes effectuées.

Il doit être possible de créer des nouvelles commandes pour les livres et DVD, et de modifier leur état de suivi. À cette fin il est également demandé de créer une table 'suivi' dans la base de données, qui doit être relié à la table 'commandedocument'.

Le passage d'un état à un autre est sous contrainte : il n'est pas possible de revenir en arrière quand une commande est livrée ou réglée, il n'est pas possible de supprimer une commande qui a été livrée, et il n'est pas possible de régler une commande non livrée.

Un trigger doit être créé dans la base de données qui insère automatiquement les exemplaires d'un document lorsqu'une commande le concernant est livrée.

Pour les abonnements, il n'y a pas de suivi, toutefois il ne doit pas être possible de supprimer un abonnement si un exemplaire est rattaché. Il est demandé de créer une méthode 'ParutionDansAbonnement' qui vérifie cela, en comparant la date de parution avec les dates de début et de fin des abonnements, et aussi de faire un test unitaire sur cette méthode.

Pour tous les onglets, il est demandé de les faire dans le même esprit des onglets existants, et plus précisément de l'onglet de parution des revues. Un tri doit être possible dans les listes, en cliquant sur les colonnes, et toutes les fonctionnalités doivent être sécurisées afin d'éviter des erreurs de manipulation.

Finalement il faudra créer une procédure stockée dans la base de données qui retourne les abonnements qui expirent dans moins de 30 jours. Cette liste doit être affichée dans une fenêtre 'pop-up' dès le démarrage de l'application.

3.3 Mission 3 (optionnelle) : Gérer le suivi de l'état des documents

La troisième mission est à nouveau une mission optionnelle. Comme je travaille seul, que j'ai fait la première mission optionnelle, et par contrainte de temps j'ai décidé de ne pas la faire.

3.4 Mission 4 : Mettre en place des authentifications

Le but de la quatrième mission est d'implémenter un système d'authentification pour l'accès à l'application. Ils existent de différents profils d'employés dans la médiathèque, qui auront accès à certaines fonctionnalités en fonction de leur profil.

3.5 Mission 5 : Qualité, tests et documentation technique

Pour la cinquième mission, il est demandé de faire une contrôle de qualité de code avec SonarLint. Il faut également créer quelques tests fonctionnels (avec SpecFlow) et ajouter la gestion de logs pour les méthodes qui utilisent des blocs try/catch (les méthodes qui communiquent avec la base de données). Finalement il faut vérifier que les commentaires normalisés ont été mises en place partout et générer la documentation technique.

4. Ressources logicielles utilisées

- L'application de départ qui nous a été fourni est écrite en C#.
- L'IDE que j'ai utilisé est Visual Studio 2019 (version 16.11.10), avec installation des plugins SonarLint for Visual Studio 2019, GitHub extension for Visual Studio, Serilog et Serilog.Sinks.File pour la journalisation, et SpecFlow pour les tests fonctionnels.
- Le serveur par lequel on accède à la base de données est WampServer version 3.2.6, avec PhpMyAdmin version 5.1.1.
- La base de données est définie en MySQL. J'ai utilisé la version 5.7.36.
- Pour le versioning j'ai utilisé GitHub : <https://github.com/carlfremault/Mediatek86>
- Pour le suivi du projet j'ai utilisé Trello :
<https://trello.com/invite/b/kDeTnxI7/c958591cb5c23a7725c91ba072fdd8dc/mediatek86>
- Pour la génération de la documentation technique dans un format 'user friendly' j'ai utilisé SandCastle.

- Finalement, pour la gestion de qualité du code ainsi que l'intégration continue j'ai utilisé SonarQube.

5. Mission 0 : Préparer l'environnement de travail

Avant d'entamer les travaux il convient de vérifier que tous les outils nécessaires sont en place. Aussi je me suis familiarisé avec le contexte et les demandes pour ce projet. Voici les travaux réalisés pendant la préparation :

- Vérification de l'installation des outils de travail nécessaires (IDE Visual Studio, WampServer)
- Familiarisation avec le dossier documentaire
- Initialisation d'un nouveau projet dans Visual Studio en clonant le dépôt GitHub MediaTek86 Gestion

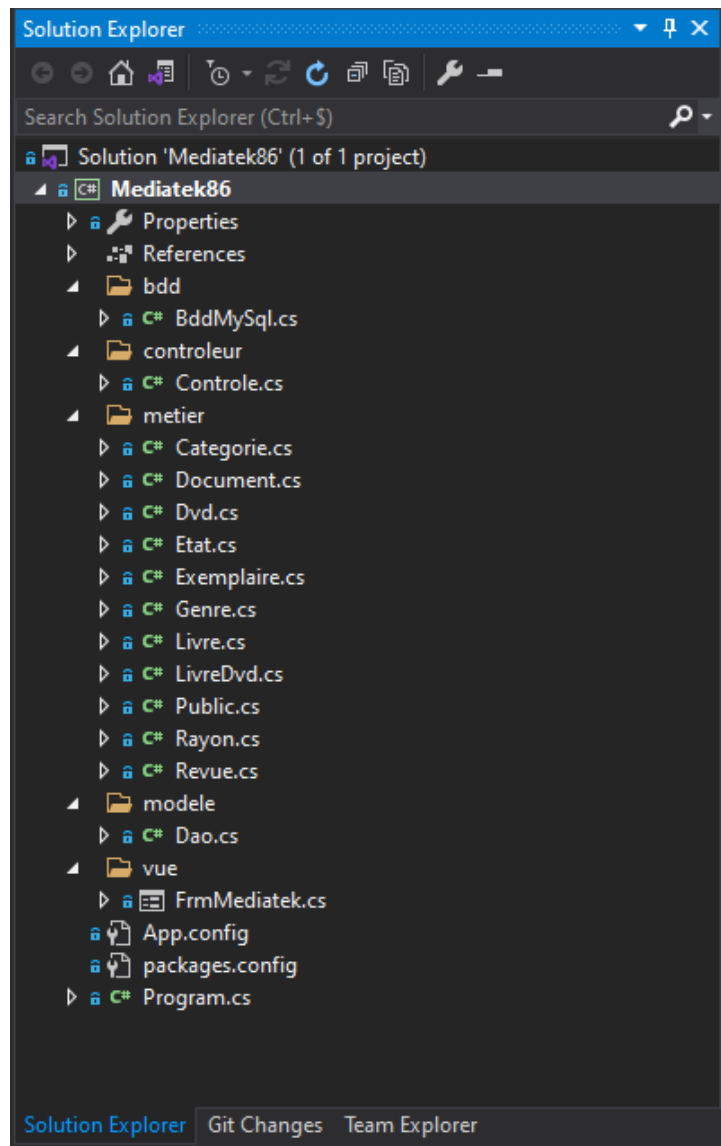


Figure 1: L'arborescence initiale du projet

- Familiarisation avec le code du projet existant
- Récupération du script de création et remplissage de la base de données MySQL 'mediatek86'
- Création de la base de données dans PhpMyAdmin depuis le script SQL fourni

Server: MySQL:3306 » Database: mediatek86

Structure SQL Search Query Export Import Operations Privileges Routines Events Triggers

Filters

Containing the word:

Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> abonnement	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 KiB	-
<input type="checkbox"/> commande	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> commandedocument	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 KiB	-
<input type="checkbox"/> document	★ Browse Structure Search Insert Empty Drop	41	InnoDB	utf8mb4_general_ci	64.0 KiB	-
<input type="checkbox"/> dvd	★ Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> etat	★ Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> exemplaire	★ Browse Structure Search Insert Empty Drop	14	InnoDB	utf8mb4_general_ci	32.0 KiB	-
<input type="checkbox"/> genre	★ Browse Structure Search Insert Empty Drop	19	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> livre	★ Browse Structure Search Insert Empty Drop	26	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> livres_dvd	★ Browse Structure Search Insert Empty Drop	30	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> public	★ Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> rayon	★ Browse Structure Search Insert Empty Drop	15	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> revue	★ Browse Structure Search Insert Empty Drop	11	InnoDB	utf8mb4_general_ci	16.0 KiB	-
13 tables	Sum	168	MyISAM	utf8_unicode_ci	304.0 KiB	0 B

↑ ☐ Check all With selected:

Figure 2: La structure de la base de données

- Test de l'application dans son état initial

Gestion Médiathèque

☐ Livres
 ☐ DVD
 ☐ Revues
 ☐ Parutions des revues

Recherches

Saisir le titre ou la partie d'un titre :
 Ou sélectionner le genre : X

Saisir un numéro de document : Rechercher
 Ou sélectionner le public : X

Ou sélectionner le rayon : X

Id	Titre	Auteur	Collection	Genre	Public	Rayon
00017	Catastrophes au Brésil	Philippe Masson		Policier	Ados	Jeunesse romans
00007	Dans les coulisses du musée	Kate Atkinson		Roman	Tous publics	Littérature étrangère
00003	Et je danse aussi	Anne-Laure Bondoux		Comédie	Tous publics	Littérature française
00019	Guide Vert - Iles Canaries		Guide Vert	Voyages	Tous publics	Voyages
00020	Guide Vert - Irlande		Guide Vert	Voyages	Tous publics	Voyages
00008	Histoire du juif errant	Jean d'Omesson		Roman	Adultes	Littérature française
00025	L'archipel du danger	Ayrolles - Masbou	De cape et de crocs	Bande dessinée	Adultes	BD Adultes
00004	L'armée furieuse	Fred Vargas	Commissaire Adamsberg	Policier	Adultes	Policiers français étrangers

Informations détaillées

Numéro de document : 00019
 Code ISBN : 3,21457E+12

Titre : Guide Vert - Iles Canaries

Auteur(e) :

Collection : Guide Vert

Genre : Voyages

Public : Tous publics

Rayon : Voyages

Chemin de l'image :

Figure 3: L'application dans son état initial

- Création du dépôt GitHub 'Mediatek86', liaison du dépôt avec le projet local
- Création de projet Trello 'Mediatek86', insertion des premières tâches

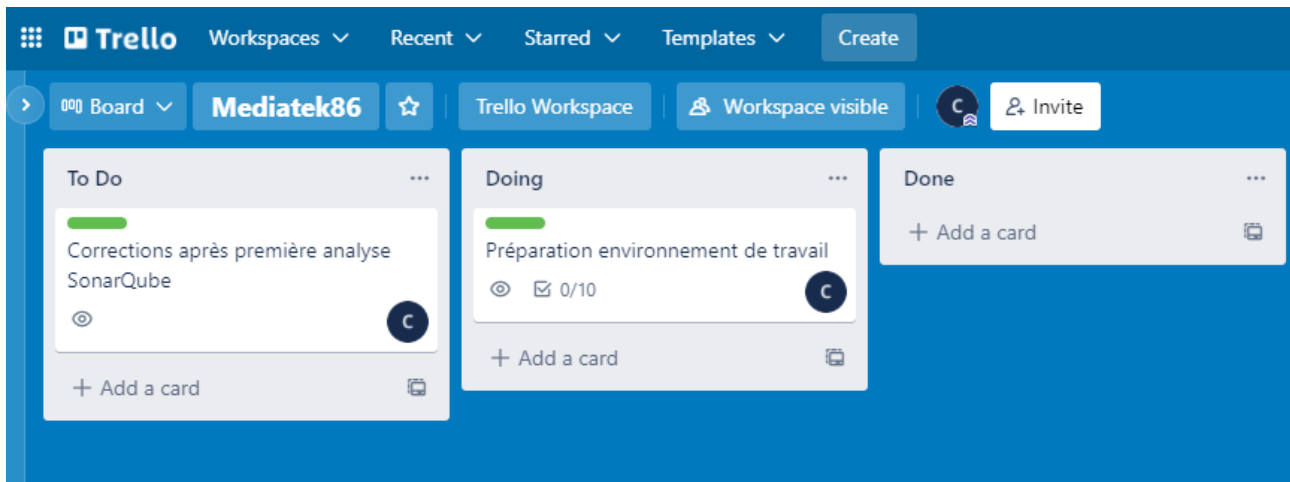


Figure 4: Storyboard Mission 0

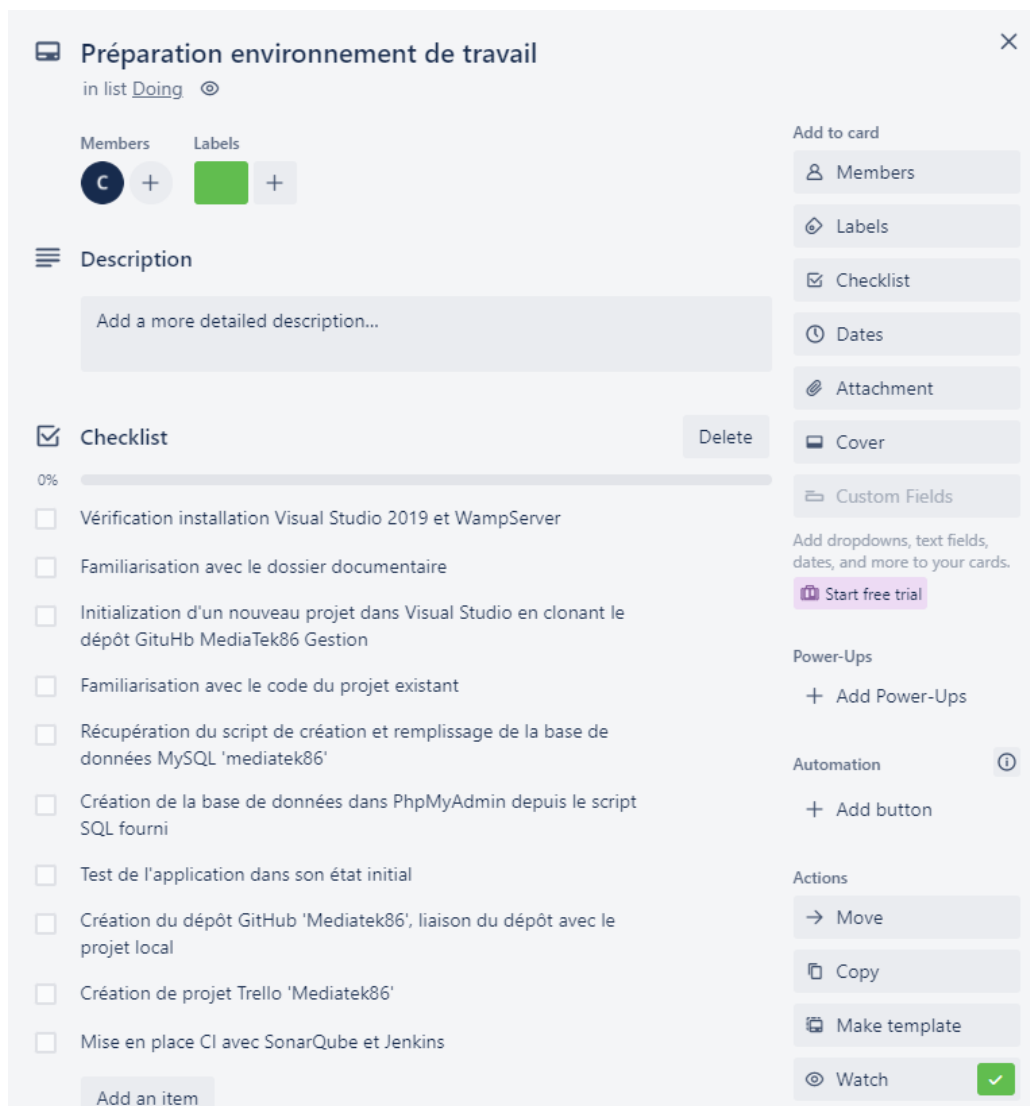


Figure 5: Premier 'Kanban'

- Configuration SonarQube, et une première vérification et quelques corrections à faire, afin de partir sur de bonnes bases

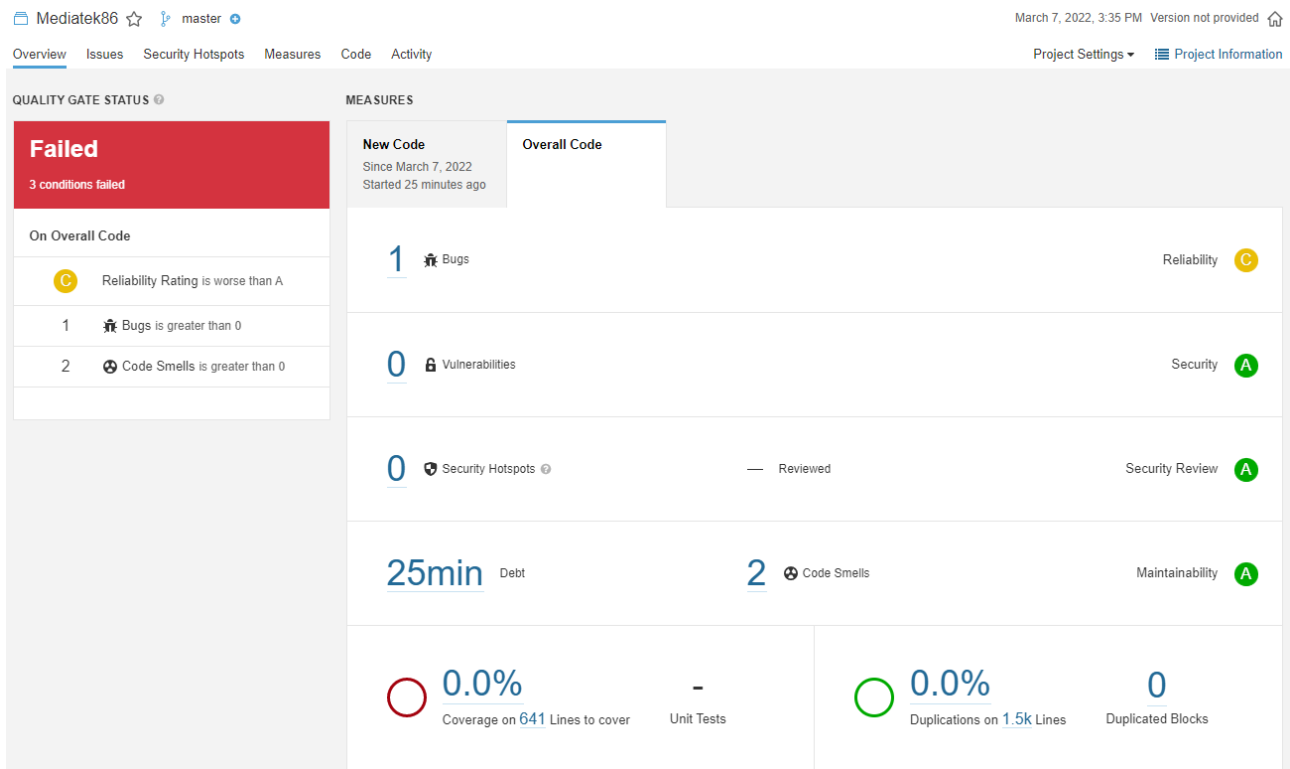


Figure 6: Première analyse SonarQube

- Création d'une branche 'dev' puis nettoyage initial du code suite recommandations SonarQube. (Comme d'habitude, tous les 'problèmes' n'en sont pas forcément ...)

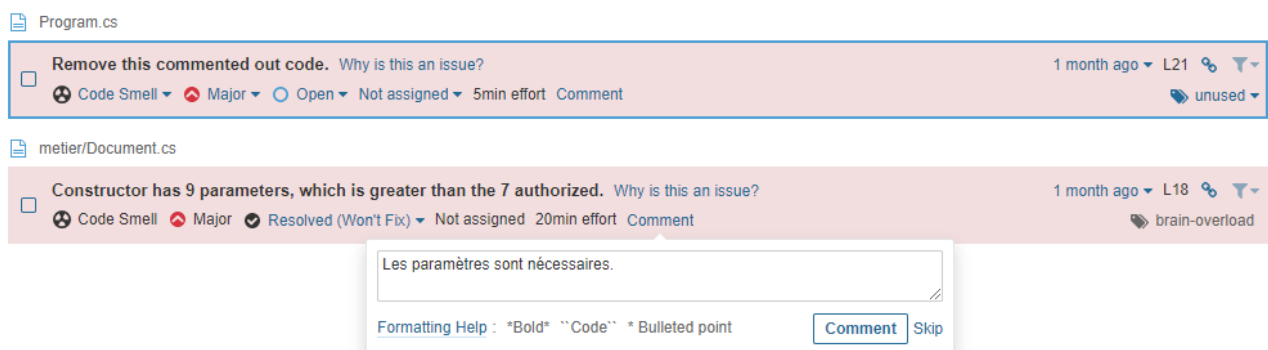


Figure 7: Les deux 'code smells' constatés

- Commit, push et merge avec la branche 'master'



Figure 8: Premier commit

6. Mission 1 (optionelle) : Gestion des documents

Plusieurs tâches sont à réaliser pour cette première mission :

- Dans les onglets actuels (Livres, DVD, Revues), ajouter les fonctionnalités (boutons) qui permettent d'ajouter, de modifier ou de supprimer un document.
- Un document ne peut être supprimé que s'il n'a pas d'exemplaire rattaché, ni de commandes.
- La modification d'un document ne peut pas porter sur son id.
- Toutes les sécurités seront mises en place pour éviter des erreurs de manipulation.
- Créer le trigger qui contrôle la contrainte de partition de l'héritage sur Document, idem pour LivresDvd .

J'ai commencé par mettre à jour le Storyboard dans Trello :

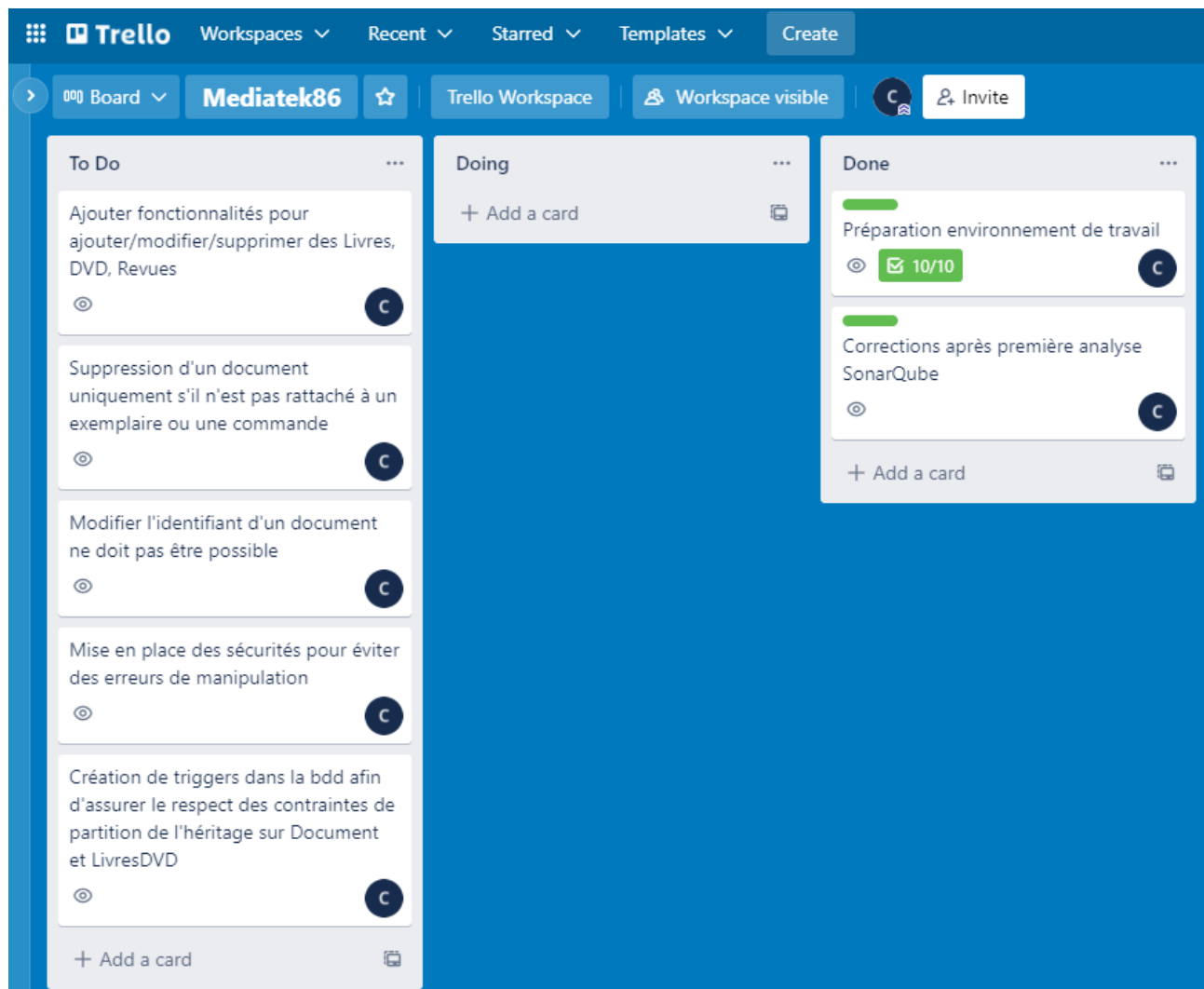


Figure 9: Storyboard Mission 1

6.1 Ajout des fonctionnalités pour ajouter/modifier/supprimer des Livres, DVD, Revues

Pour cette tâche, j'ai d'abord ajouté un 'groupBox' avec trois boutons 'ajouter', 'modifier' et 'supprimer' dans les onglets 'Livres', 'DVD' et 'Revues'.

The screenshot shows a web interface with two main sections. The top section, titled 'Informations détaillées', contains several labeled input fields: 'Numéro de document', 'Code ISBN', 'Titre', 'Auteur(e)', 'Collection', 'Genre', 'Public', 'Rayon', and 'Chemin de l'image'. To the right of these fields is a large, empty rectangular box with a dashed border. The bottom section, titled 'Gestion des livres', contains three buttons: 'Ajouter', 'Modifier', and 'Supprimer'.

Figure 10: Informations détaillées & GroupBox de gestion des livres

Rapidement je me suis réalisé qu'il fallait réfléchir sur l'implémentation de ces fonctionnalités, en faisant de sorte que l'utilisateur ne pourrait pas faire de fausses manipulations. On pourrait par exemple, après un clic sur le bouton 'Ajouter' faire apparaître une nouvelle fenêtre permettant de faire les saisies nécessaires. L'avantage étant qu'ainsi, les fonctionnalités de recherche (qui modifient automatiquement les champs 'Informations détaillées' dès qu'une nouvelle sélection de documents apparaît) seraient inaccessibles.

Toutefois, le dossier documentaire précise : *"Le but est d'exploiter les trois premiers onglets déjà existants en les faisant évoluer pour ajouter ces nouvelles fonctionnalités."*

J'aurais aussi pu désactiver le groupBox 'Rechercher' dès qu'on entame une saisie. Toutefois, il me semble intéressant, d'un point de vue utilisateur, de pouvoir faire une recherche, et consulter la liste obtenue, avant et pendant la saisie. Or la désactivation du groupBox 'Rechercher' désactive aussi la possibilité de faire défiler la liste.

La difficulté qui se présente est que, dès qu'une recherche est effectuée, la liste d'affichage de documents change et l'application remplit les champs infos avec les détails du premier document (automatiquement) sélectionné, écrasant ainsi les informations saisies si une saisie est en cours.

Ainsi, il me semble logique, avant de passer à l'étape d'ajout/modification de tuples dans la base de données quand les saisies sont faites, de s'occuper de la sécurisation des manipulations.

J'ai donc procédé avec le prise en charge immédiate et en parallèle d'un prochain ticket :

6.2 Mises en place des sécurités pour éviter des erreurs de manipulation.

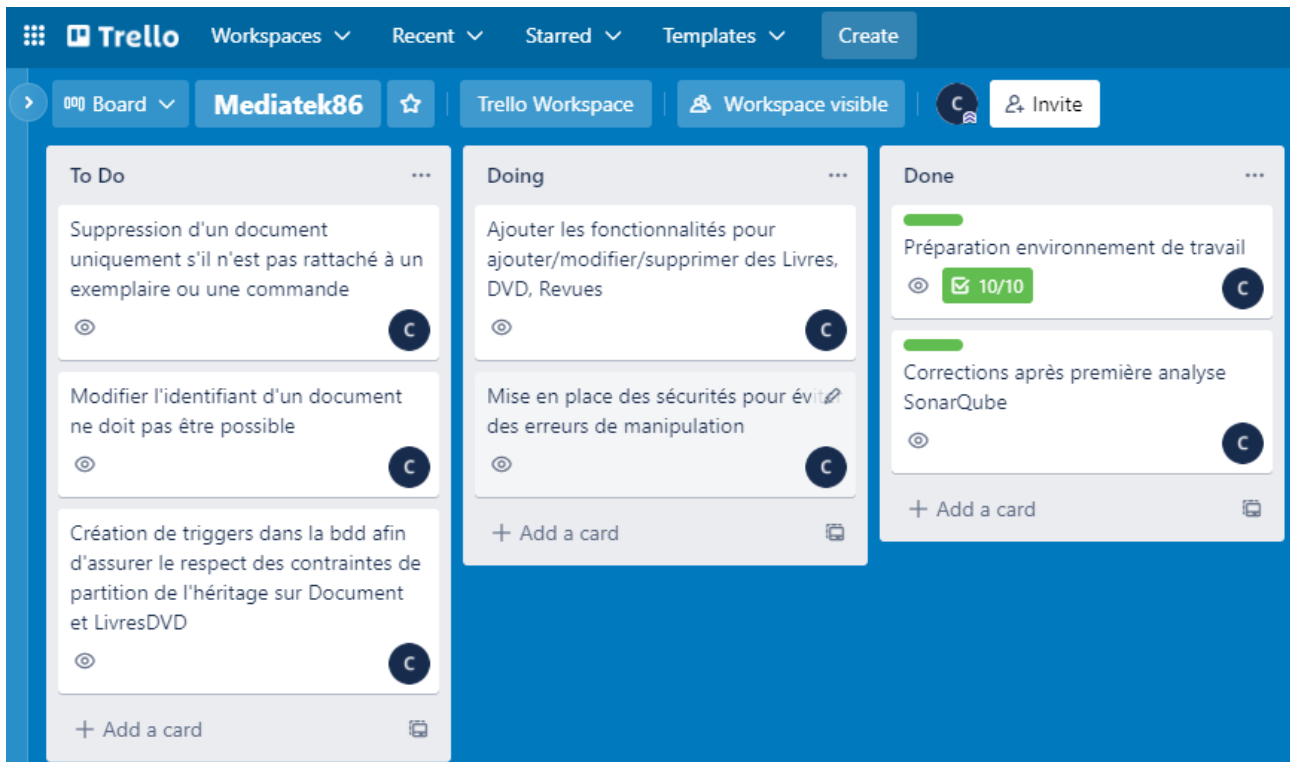


Figure 11: Storyboard : prise en charge de tâches

Afin de gérer la saisie de données lors d'un ajout ou modification, j'ai ajouté des boutons 'Enregistrer' et 'Annuler' dans le groupBox qui représente les Informations détaillées, dans les trois onglets.

Informations détaillées

Numéro de document :	<input type="text" value="00007"/>	Code ISBN :	<input type="text" value="6541236987541"/>
Titre :	<input type="text" value="Dans les coulisses du musée"/>		
Auteur(e) :	<input type="text" value="Kate Atkinson"/>		
Collection :	<input type="text"/>		
Genre :	<input type="text" value="Roman"/>		
Public :	<input type="text" value="Tous publics"/>		
Rayon :	<input type="text" value="Littérature étrangère"/>		
Chemin de l'image :	<input type="text"/>		

Figure 12: Information détaillées avec boutons pour enregistrer ou annuler une saisie

Comme le code sera très similaire, s'il s'agit de Livres, DVD ou Revues, j'ai commencé par faire les implémentations nécessaires pour les Livres. Ensuite je pourra recopier et adapter les fonctionnalités pour les autres onglets.

Dans la vue FrmMediaTek j'ai ajouté des fonctions qui permettent d'activer ou désactiver chaque bouton créé jusqu'ici (Ajouter, Modifier, Supprimer, Enregistrer, Annuler). Ceci permettra de guider et limiter les utilisateurs dans leurs manipulations.

```
/// <summary>
/// (Dés)Activer le bouton qui permet de modifier un livre
/// </summary>
/// <param name="actif"></param>
2 references | 0 changes | 0 authors, 0 changes
private void ActiverBoutonModifLivre(Boolean actif)
{
    btnModifLivre.Enabled = actif;
}
```

Figure 13: Exemple de méthode qui permet d'activer ou désactiver un bouton

J'ai ajouté une propriété booléen qui définit si on est en train de faire une saisie ou non :

```
/// <summary>
/// Booléen, validé comme 'true' si on est en train de faire une saisie pour ajouter/modifier un livre
/// </summary>
private bool saisieLivre;
```

Figure 14: Booléen 'saisieLivre'

Dès que l'onglet 'Livres' est ouvert, le booléen est mis à 'faux' et les boutons 'Enregistrer' et 'Annuler' sont désactivés.

```
private void TabLivres_Enter(object sender, EventArgs e)
{
    lesLivres = controle.GetAllLivres();
    RemplirComboCategorie(controle.GetAllGenres(), bdgGenres, cbxLivresGenres);
    RemplirComboCategorie(controle.GetAllPublics(), bdgPublics, cbxLivresPublics);
    RemplirComboCategorie(controle.GetAllRayons(), bdgRayons, cbxLivresRayons);
    RemplirLivresListeComplete();
    ActiverBoutonEnregLivre(false);
    ActiverBoutonAnnulerSaisieLivre(false);
    saisieLivre = false;
}
```

Figure 15: La méthode événementielle d'entrée dans l'onglet 'Livres'

Puis j'ai ajouté trois fonctions :

- AutoriserModifLivre qui enlève la protection 'Read Only' des champs infos, et active les boutons 'Enregistrer' et 'Annuler'.
- StartSaisieLivre et
- StopSaisieLivre qui actionnent cette fonction ainsi que le booléen 'saisieLivre' :

```
/// <summary>
/// Démarre la saisie d'un livre, déverrouille les champs 'infos'
/// </summary>
1 reference | 0 changes | 0 authors, 0 changes
private void StartSaisieLivre()
{
    : saisieLivre = true;
    : AutoriserModifLivre(true);
}

/// <summary>
/// Arrête la saisie d'un livre, verrouille les champs 'infos'
/// </summary>
7 references | 0 changes | 0 authors, 0 changes
private void StopSaisieLivre()
{
    : saisieLivre = false;
    : AutoriserModifLivre(false);
}

/// <summary>
/// (Dés)activation de La protection des différents champs 'informations détaillées' ainsi que le bouton 'enregistrer'
/// </summary>
/// <param name="actif"></param>
3 references | 0 changes | 0 authors, 0 changes
private void AutoriserModifLivre(Boolean actif)
{
    : txbLivresAuteur.ReadOnly = !actif;
    : txbLivresCollection.ReadOnly = !actif;
    : txbLivresImage.ReadOnly = !actif;
    : txbLivresIsbn.ReadOnly = !actif;
    : txbLivresNumero.ReadOnly = !actif;
    : txbLivresGenre.ReadOnly = !actif;
    : txbLivresPublic.ReadOnly = !actif;
    : txbLivresRayon.ReadOnly = !actif;
    : txbLivresTitre.ReadOnly = !actif;
    : ActiverBoutonEnregLivre(actif);
    : ActiverBoutonAnnulerSaisieLivre(actif);
}
```

Figure 16: Méthodes de sécurisation de manipulations

Pour tester les implémentations suivantes j'ai ajouté la fonction événementielle suite à un clic sur le bouton 'Ajouter' :

```

/// <summary>
/// Événement clic sur le bouton 'Ajouter'
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | 0 changes | 0 authors, 0 changes
private void btnAjoutLivres_Click(object sender, EventArgs e)
{
    VideLivresInfos();
    StartSaisieLivres();
}

```

Figure 17: Cliquer sur 'Ajouter' vide les infos détaillées et démarre la saisie

Ensuite j'ai sécurisé les différentes fonctionnalités de recherche. Il fallait un peu de gymnastique algorithmique, mais au fin de compte le processus est similaire pour les différents cas : J'ai extrait les fonctionnalités existantes des méthodes événementielles des différents champs de recherche dans des nouvelles fonctions afin d'éviter des répétitions de code (Don't Repeat Yourself).

Pour la recherche d'un document par numéro par exemple :

```

/// <summary>
/// Recherche et affichage du livre dont on a saisi le numéro.
/// Si non trouvé, affichage d'un MessageBox.
/// </summary>
2 references | 0 changes | 0 authors, 0 changes
private void LivresNumRecherche()
{
    if (!txbLivresNumRecherche.Text.Equals(""))
    {
        txbLivresTitreRecherche.Text = "";
        cbxLivresGenres.SelectedIndex = -1;
        cbxLivresRayons.SelectedIndex = -1;
        cbxLivresPublics.SelectedIndex = -1;
        Livre livre = lesLivres.Find(x => x.Id.Equals(txbLivresNumRecherche.Text));
        if (livre != null)
        {
            List<Livre> livres = new List<Livre>();
            livres.Add(livre);
            RemplirLivresListe(livres);
        }
        else
        {
            MessageBox.Show("numéro introuvable");
            RemplirLivresListeComplete();
        }
    }
    else
    {
        RemplirLivresListeComplete();
    }
}

```

Figure 18: Exemple : la recherche d'un livre par son numéro

Puis j'ai modifié la méthode événementielle initiale pour faire les vérifications nécessaires et actionner la recherche si besoin :

```
/// <summary>
/// Événement clic sur le bouton 'Rechercher'. Vérifie si on est en train de faire une saisie (ajout ou modif de livre)
/// Si non : effectue la recherche
/// Si oui : demande si l'utilisateur veut abandonner la saisie, si oui : abandon, vide les champs 'infos' et effectue la recherche
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | 0 changes | 0 authors, 0 changes
private void BtnLivresNumRecherche_Click(object sender, EventArgs e)
{
    if (saisieLivre && verifAbandonSaisie()) // On est en train de saisir, et on décide d'abandonner
    {
        StopSaisieLivre();
        LivresNumRecherche();
    }
    else if (saisieLivre) // On est en train de saisir, et on n'abandonne pas
    {
        txbLivresNumRecherche.Text = "";
    }
    else // On n'est pas en train de saisir
    {
        LivresNumRecherche();
    }
}
```

Figure 19: Méthode événementielle : clic sur le bouton 'Rechercher'

Pour cela j'ai créé une méthode 'VerifAbandonSaisie' qui présente un dialogue à l'utilisateur, lui demandant s'il veut bien abandonner sa saisie :

```
/// <summary>
/// Affichage d'un MessageBox pour vérifier si l'utilisateur veut bien abandonner sa saisie
/// </summary>
/// <returns>True si abandon saisie confirmé, sinon false</returns>
9 references | 0 changes | 0 authors, 0 changes
private bool VerifAbandonSaisie()
{
    return (MessageBox.Show("Etes-vous sûr de vouloir abandonner votre saisie ?",
        "Confirmation d'abandon de saisie", MessageBoxButtons.YesNo) == DialogResult.Yes);
}
```

Figure 20: MessageBox pour vérifier si l'utilisateur souhaite abandonner la saisie

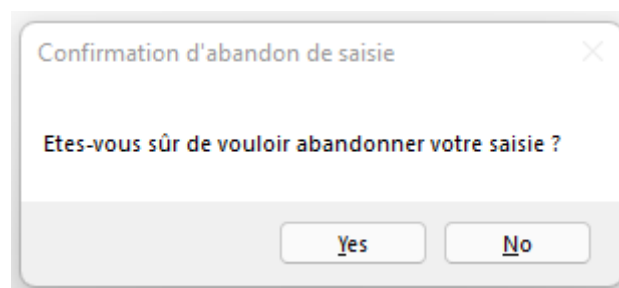


Figure 21: Le MessageBox généré

Cette méthode pourra être utilisée pour les trois cas (Livres, DVD, Revues).

Pour les autres champs j'ai eu besoin d'une vérification en plus : effectivement, la recherche sur un titre ou le filtrage sur genre, public ou rayon, se font dès qu'un changement dans un de ces champs est détecté.

Comme j'ai préféré de vider ces champs, quand un utilisateur décide de ne pas abandonner sa saisie (autrement la liste affichée ne correspondrait plus au contenu des champs), le fait de vider la recherche actionnait à nouveau l'événement, et le dialogue de demande de confirmation d'abandon s'affichait une deuxième fois. Ce n'est clairement pas souhaitable.

```
/// <summary>
/// Cette procédure est exécutée à chaque ajout ou suppression de caractère dans le textBox de recherche de titre.
/// Vérifie si on est en train de faire une saisie (ajout ou modif de livre)
/// Si non : effectue la recherche
/// Si oui : vérification si l'utilisateur veut abandonner la saisie et si le champs n'est pas vidé (pour éviter double
trigger)
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | 0 changes | 0 authors, 0 changes
private void TxbLivresTitreRecherche_TextChanged(object sender, EventArgs e)
{
    if (saisieLivre) // On est en train de saisir
    {
        if (txbLivresTitreRecherche.Text != "" && verifAbandonSaisie()) // Le champ de recherche n'est pas vidé, on vérifie
        si l'utilisateur veut abandonner la saisie
        {
            StopSaisieLivre();
            LivresTitreRecherche();
        }
        else // L'utilisateur ne veut pas abandonner la saisie
        {
            txbLivresTitreRecherche.Text = "";
        }
    }
    else // On n'est pas en train de saisir
    {
        LivresTitreRecherche();
    }
}
```

Figure 22: Empêchement de double affichage du MessageBox

La fonctionnalité est la même pour les comboBox.

Ils restent alors les trois boutons qui annulent les filtre de ces combos, les trois sont traités de façon similaire :

```

/// <summary>
/// Sur le clic du bouton d'annulation, appel de la fonction AnnulFiltreCbo
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | 0 changes | 0 authors, 0 changes
private void BtnLivresAnnulGenres_Click(object sender, EventArgs e)
{
    AnnulFiltreCbo();
}

/// <summary>
/// Sur le clic d'un bouton d'annulation, vérification si on est en train de faire une saisie.
/// Si non affichage de la liste complète des livres.
/// Si oui, vérification si l'utilisateur veut abandonner sa saisie d'abord.
/// </summary>
3 references | 0 changes | 0 authors, 0 changes
private void AnnulFiltreCbo()
{
    if (saisieLivre)
    {
        if (verifAbandonSaisie())
        {
            StopSaisieLivre();
            RemplirLivresListeComplete();
            LivresListeSelection();
        }
    }
    else
    {
        RemplirLivresListeComplete();
    }
}

```

Figure 23: Appel de la méthode qui annule l'application des filtres

La sécurisation des champs étant faite pour l'onglet 'Livres', et comme il y a eu déjà beaucoup de changements, j'ai préféré faire un commit et push avant de continuer les travaux. Bien entendu je n'ai pas encore fait de 'merge' avec la branche 'master'.

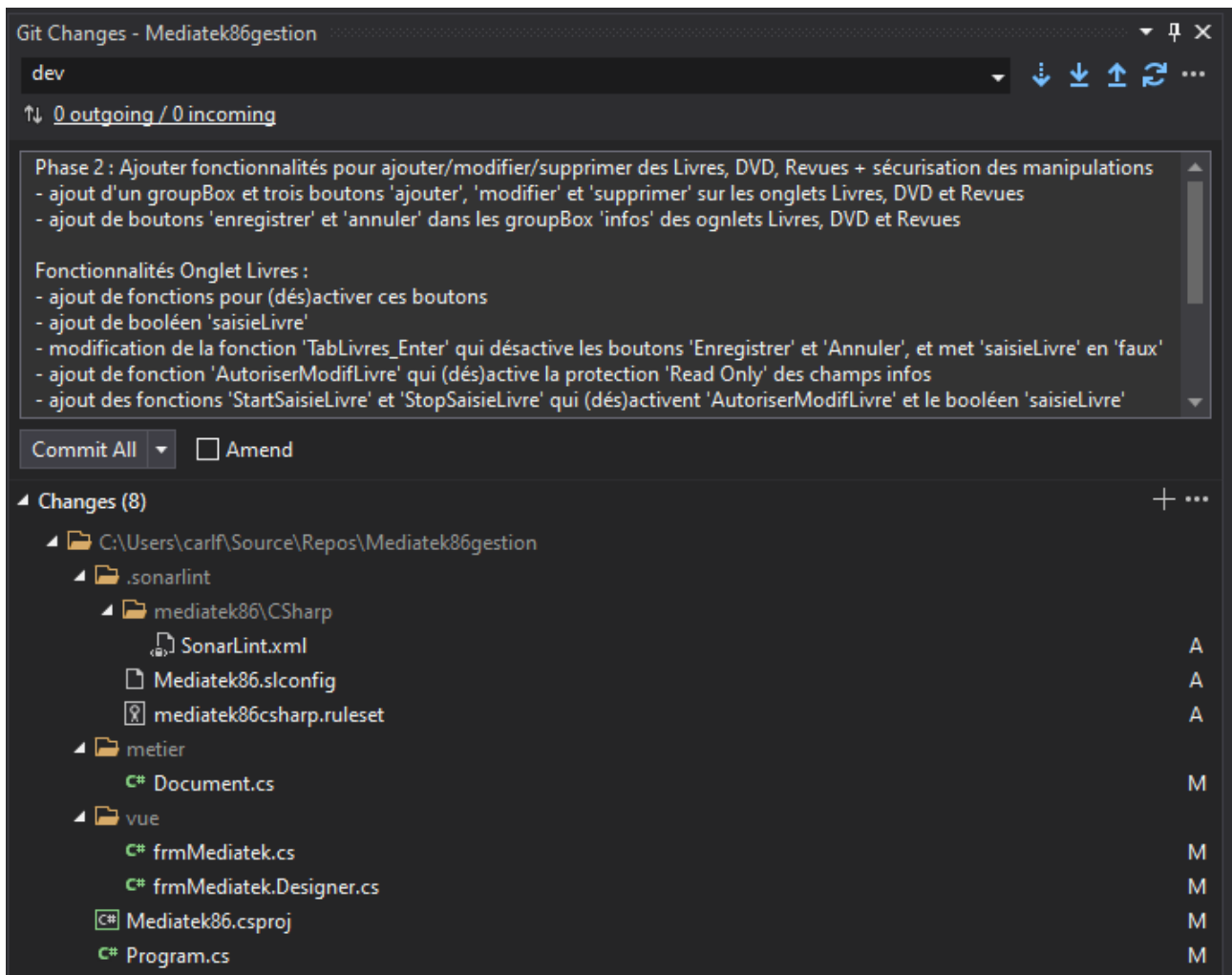


Figure 24: Commit de la phase 2

Avant d'implémenter tous ces changements à l'identique pour les onglets DVD et Revues j'ai préféré de faire le reste des fonctionnalités d'ajout/modification/suppression. On ne sait jamais s'il faudrait modifier quelque chose quelque part.

6.3 Implémentation d'ajout et de modification des livres

Au niveau des champs 'infos', afin de sécuriser les saisies, il convient de transformer les champs texte 'Genre', 'Public' et 'Rayon' en comboBox. Ainsi l'utilisateur ne pourra uniquement saisir des valeurs existantes dans la base de données.

The screenshot shows a form titled 'Informations détaillées'. It contains the following fields and controls:

- Numéro de document :** A text input field.
- Code ISBN :** A text input field.
- Titre :** A text input field.
- Auteur(e) :** A text input field.
- Collection :** A text input field.
- Genre :** A dropdown menu with a downward arrow.
- Public :** A dropdown menu with a downward arrow.
- Rayon :** A dropdown menu with a downward arrow.
- Chemin de l'image :** A text input field.
- Buttons:** Two buttons labeled 'Enregistrer' and 'Annuler' are positioned below the 'Chemin de l'image' field.
- Image Placeholder:** A large rectangular area on the right side of the form, outlined with a dashed border and small square handles at the corners and midpoints, indicating it is a placeholder for an image.

Figure 25: Les nouveaux comboBox sont implémentés

Ensuite j'ai fait les modifications nécessaires dans la vue :

- Ajout de 'BindingSources' pour chaque comboBox.
- Modification de la fonction 'TabLivres_Enter' pour remplir les comboBox depuis ces nouvelles BindingSources.
- Modification des fonctions 'AfficheLivresInfos' et 'VideLivresInfos' pour exploiter les comboBox.
- Modification de la fonction 'AutoriserModifLivre' pour verrouiller ou non les comboBox.

```

private void AfficheLivresInfos(Livre livre)
{
    txblivresAuteur.Text = livre.Auteur;
    txblivresCollection.Text = livre.Collection;
    txblivresImage.Text = livre.Image;
    txblivresIsbn.Text = livre.Isbn;
    txblivresNumero.Text = livre.Id;
    cbxInfosLivresGenres.SelectedIndex = cbxInfosLivresGenres.FindStringExact(livre.Genre);
    cbxInfosLivresPublics.SelectedIndex = cbxInfosLivresPublics.FindStringExact(livre.Public);
    cbxInfosLivresRayons.SelectedIndex = cbxInfosLivresRayons.FindStringExact(livre.Rayon);
    txblivresTitre.Text = livre.Titre;
    string image = livre.Image;
    try
    {
        pcbLivresImage.Image = Image.FromFile(image);
    }
    catch
    {
        pcbLivresImage.Image = null;
    }
}

/// <summary> Vide les zones d'affichage des informations du livre
2 references | 0 changes | 0 authors, 0 changes
private void VideLivresInfos()
{
    txblivresAuteur.Text = "";
    txblivresCollection.Text = "";
    txblivresImage.Text = "";
    txblivresIsbn.Text = "";
    txblivresNumero.Text = "";
    cbxInfosLivresGenres.SelectedIndex = -1;
    cbxInfosLivresPublics.SelectedIndex = -1;
    cbxInfosLivresRayons.SelectedIndex = -1;
    txblivresTitre.Text = "";
    pcbLivresImage.Image = null;
}

```

Figure 26: Modifications pour exploiter les comboBox

Après j'ai géré l'événement d'un clic sur le bouton 'annuler' lors d'une saisie. Après confirmation de l'utilisateur les champs infos sont vidés, la saisie est arrêtée, et si un livre est sélectionné dans la liste ses informations sont affichées.

```

/// <summary>
/// Événement clic sur le bouton annuler lors d'une saisie
/// Vide les champs infos, arrête la saisie et affiche les infos du livre sélectionné dans la liste
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | 0 changes | 0 authors, 0 changes
private void btnAnnulerSaisieLivre_Click(object sender, EventArgs e)
{
    if (VerifAbandonSaisie())
    {
        VideLivresInfos();
        StopSaisieLivre();
        LivresListeSelection();
    }
}

```

Figure 27: Annulation d'une saisie

Une nouvelle question s'est présentée : que faire si l'utilisateur clic sur le bouton 'Enregistrer' ? Il faut bien que l'application sache si elle doit ajouter un livre dans la base de données, ou en modifier un. Le plus simple semblait d'insérer un nouveau booléen 'modifLivre', 'false' par défaut et qui sera validé comme 'true' lors d'une modification.

```
/// <summary>
/// Booléen, validé comme 'true' si on est en train de modifier un livre
/// </summary>
private bool modifLivre = false;
```

Figure 28: Nouveau booléen pour signaler qu'une modification est en cours

J'ai découplé le déverrouillage du champ de numéro d'un document et je l'ai mis dans une fonction séparée. Effectivement, lors d'une modification il ne doit pas être possible de modifier le numéro d'un document.

```
/// <summary>
/// (Dés)activation de la protection des différents champs 'informations détaillées' ainsi que
/// le bouton 'enregistrer'
/// </summary>
/// <param name="actif"></param>
4 references | 0 changes | 0 authors, 0 changes
private void AutoriserModifLivre(Boolean actif)
{
    txbLivresAuteur.ReadOnly = !actif;
    txbLivresCollection.ReadOnly = !actif;
    txbLivresImage.ReadOnly = !actif;
    txbLivresIsbn.ReadOnly = !actif;
    cbxInfosLivresGenres.Enabled = actif;
    cbxInfosLivresPublics.Enabled = actif;
    cbxInfosLivresRayons.Enabled = actif;
    txbLivresTitre.ReadOnly = !actif;
    ActiverBoutonEnregLivre(actif);
    ActiverBoutonAnnulerSaisieLivre(actif);
}

/// <summary>
/// (Dés)activation de la protection du champ 'Numéro du document'
/// </summary>
/// <param name="actif"></param>
2 references | 0 changes | 0 authors, 0 changes
private void AutoriserModifDocId(Boolean actif)
{
    txbLivresNumero.ReadOnly = !actif;
}
```

Figure 29: Sécurisations en fonction de l'ajout ou de la modification d'un livre

Quelques verrouillages et fonctionnalités supplémentaires étaient nécessaires aussi concernant les boutons implémentés en début de cette mission. Il semble logique de désactiver les boutons 'Ajouter', 'Modifier' et 'Supprimer' quand on est en train de faire une saisie.

```
/// <summary>
/// Démarre la saisie d'un livre, déverrouille les champs 'infos'
/// </summary>
2 references | 0 changes | 0 authors, 0 changes
private void StartSaisieLivre()
{
    saisieLivre = true;
    AutoriserModifLivre(true);
    ActiverBoutonAjoutLivre(false);
    ActiverBoutonModifLivre(false);
    ActiverBoutonSupprLivre(false);
}

/// <summary>
/// Arrête la saisie d'un livre, verrouille les champs 'infos'
/// </summary>
10 references | 0 changes | 0 authors, 0 changes
private void StopSaisieLivre()
{
    saisieLivre = false;
    modifLivre = false;
    AutoriserModifLivre(false);
    ActiverBoutonAjoutLivre(true);
    ActiverBoutonModifLivre(true);
    ActiverBoutonSupprLivre(true);
}
```

Figure 30: Début et fin d'une saisie

Puis j'ai modifié la méthode événementielle qui se déclenche lors d'un clic sur le bouton 'Ajouter' et j'ai implémenté celle pour le bouton 'Modifier'. Bien entendu, lors d'un clic sur 'Modifier' les champs ne doivent pas être vidés

```

/// <summary>
/// Événement clic sur le bouton 'Ajouter'. Vide les champs 'infos', déverrouille le champ 'Numéro de
  document', Démarre la saisie d'un livre
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | 0 changes | 0 authors, 0 changes
private void btnAjoutLivre_Click(object sender, EventArgs e)
{
    VideLivresInfos();
    AutoriserModifDocId(true);
    StartSaisieLivre();
}

/// <summary>
/// Événement clic sur le bouton 'Modifier'. Verrouille le champ 'Numéro de document'. Démarre la saisie
  d'un livre
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | 0 changes | 0 authors, 0 changes
private void btnModifLivre_Click(object sender, EventArgs e)
{
    modifLivre = true;
    AutoriserModifDocId(false);
    StartSaisieLivre();
}

```

Figure 31: Différents traitements en fonction de l'action choisie

Ensuite il fallait implémenter l'événement d'un clic sur le bouton 'Enregistrer'. Tout d'abord il fallait vérifier si les informations obligatoires ont été saisies. Mise à part du numéro de document, Genre, Public et Rayon, le contexte ne donne pas de précisions là-dessus, toutefois on peut s'imaginer que le titre soit obligatoire aussi.

Si toutes les informations obligatoires sont saisies, un nouvel objet 'Livre' est créé avec les éléments renseignés dans les différents champs. Dans un premier temps il n'y a pas de différence à faire entre l'ajout et la modification d'un livre car en cas de modification le numéro de document a été récupéré et maintenu depuis le document initial.

```

/// <summary>
/// Événement clic sur le bouton 'Enregistrer'
/// Vérifie si les champs requis (numéro, genre, public, rayon) sont saisis.
/// Si oui procède à l'ajout ou modification du livre
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | 0 changes | 0 authors, 0 changes
private void btnEnregistrerLivre_Click(object sender, EventArgs e)
{
    if(cbxInfosLivresGenres.SelectedIndex == -1 || cbxInfosLivresPublics.SelectedIndex == -1 ||
        cbxInfosLivresRayons.SelectedIndex == -1 || txtLivresNumero.Text == "" || txtLivresTitre.Text == "")
    {
        MessageBox.Show("Les champs marqués d'un astérisque (*) sont obligatoires.", "Information");
        return;
    }

    Genre leGenre = (Genre)bdgInfosGenres.List[bdgInfosGenres.Position];
    String idGenre = leGenre.Id;
    String genre = leGenre.ToString();
    Public lePublic = (Public)bdgInfosPublics.List[bdgInfosPublics.Position];
    String idPublic = lePublic.Id;
    String unPublic = lePublic.ToString();
    Rayon leRayon = (Rayon)bdgInfosRayons.List[bdgInfosRayons.Position];
    String idRayon = leRayon.Id;
    String rayon = leRayon.ToString();
    String id = txtLivresNumero.Text;
    String titre = txtLivresTitre.Text;
    String image = txtLivresImage.Text;
    String isbn = txtLivresIsbn.Text;
    String auteur = txtLivresAuteur.Text;
    String collection = txtLivresCollection.Text;

    Livre leLivre = new Livre(id, titre, image, isbn, auteur, collection, idGenre, genre, idPublic, unPublic, idRayon,
        rayon);
}

```

Figure 32: Début du processus d'enregistrement d'un livre

Ensuite, cette même fonction vérifie l'état du booléen 'modifLivre' et appelle la fonction correspondante du contrôleur, en lui envoyant l'objet à ajouter/modifier en paramètre :

```

if (modifLivre)
{
    if (!controle.ModifLivre(leLivre))
    {
        MessageBox.Show("Une erreur est survenue.", "Erreur");
        return;
    }
}
else
{
    if(!controle.CreerLivre(leLivre))
    {
        MessageBox.Show("numéro de publication déjà existant", "Erreur");
        txbLivresNumero.Text = "";
        txbLivresNumero.Focus();
        return;
    }
}
VideLivresInfos();
StopSaisieLivre();
controle.RefreshAllLivres();
lesLivres = controle.GetAllLivres();
RemplirLivresListeComplete();

```

Figure 33: Suite et fin de la procédure d'enregistrement d'un livre

Les fonctions 'ModifLivre' et 'CreerLivre' du contrôleur renvoient un booléen, qui est 'true' si l'insertion/update a réussi, 'false' en cas d'échec. En cas d'échec, un message avertit l'utilisateur. Les erreurs concernant la liaison à la base de données étant gérées dans la classe qui s'occupe de cette liaison, les autres cas d'erreurs sont limités. En cas d'ajout d'un nouveau livre il se peut que l'utilisateur saisisse un numéro existant. L'enregistrement s'arrête alors, l'utilisateur est averti et peut réessayer.

En cas de succès il convient de vider les champs de saisie, d'arrêter la saisie (ce qui permet de remettre les verrouillages des champs et boutons, ainsi que l'état des booléens `saisieLivre` et `modifLivre` en état), et de rafraîchir la liste des livres pour que les modifications soient prises en compte.

Or il se trouve que initialement, lors de l'ouverture de l'onglet, la vue récupère la liste des livres depuis l'objet 'lesLivres' du contrôleur. Cet objet est une propriété 'readonly' qui est remplie quand le contrôleur est instancié. Ce fonctionnement n'est pas optimal. Effectivement il doit être possible pour le contrôleur de rafraîchir cet objet. Du coup j'ai fait quelques modifications dans le contrôleur :

- Suppression de la 'keyword' `readonly` pour cette propriété.
- Ajout d'une fonction 'RefreshAllLivres'

```

/// <summary>
/// Recupère la liste des livres depuis la bdd
/// </summary>
2 references | 0 changes | 0 authors, 0 changes
public void RefreshAllLivres()
{
    ...
    lesLivres = Dao.GetAllLivres();
}

```

Figure 34: Fonction pour rafraîchir la collection 'lesLivres'

Ceci permet de limiter les appels à la base de données. Plutôt d'incorporer la rafraîchissement dans la fonction 'GetAllLivres' on garde la main et on ne fait cet appel que si nécessaire.

Maintenant il fallait s'occuper des fonctions CreerLivre et ModifLivre. Le contrôleur passe les objets envoyés en paramètre au 'Dao' (Data Access Object) :

```

/// <summary>
/// Crée un livre dans la bdd
/// </summary>
/// <param name="livre">L'objet Livre concerné</param>
/// <returns>True si la création a pu se faire</returns>
1 reference | 0 changes | 0 authors, 0 changes
public bool CreerLivre(Livre livre)
{
    ...
    return Dao.CreerLivre(livre);
}

/// <summary>
/// Modifie un livre dans la bdd
/// </summary>
/// <param name="livre">L'objet Livre concerné</param>
/// <returns>True si la modification a pu se faire</returns>
1 reference | 0 changes | 0 authors, 0 changes
public bool ModifLivre(Livre livre)
{
    ...
    return Dao.ModifLivre(livre);
}

```

Figure 35: Les méthodes du contrôleur

Le Dao prépare la requête et, en cas de requête 'non query' l'envoie à la fonction 'reqUpdate' de la classe BddMySQL qui interface avec la base de données.

Une spécificité de l'architecture de la base de données cependant est que un livre ne correspond pas à un seul objet: un livre (ou un DVD) héritent d'une entité mère 'Livres_DVD' qui à son tour hérite de l'entité mère 'Document'. Ainsi, pour ajouter un livre on doit faire des insertions dans les trois tables. Pour modifier, on doit mettre à jour les

tables 'Document' et 'Livre' (comme la table 'Livres_DVD' ne contient uniquement le numéro de document qui n'est pas modifiable).

Afin de préserver l'intégrité de la base de données, mieux vaut ne pas envoyer des différentes requêtes qui concernent le même livre séparément. Du coup j'ai modifié la fonction 'ReqUpdate' pour qu'elle accepte une Liste de requêtes plutôt qu'une seule, et pour qu'elle regroupe l'ensemble des requêtes dans une transaction.

En cas d'échec d'une des requêtes, la transaction entière est annulé avec un 'rollback'. En cas de réussite, l'ensemble est validé avec un 'commit'.

```
/// <summary>
/// Exécution de requêtes autre que "select" dans une seule transaction
/// </summary>
/// <param name="queries">Liste des requêtes à faire</param>
/// <param name="parameters"></param>
3 references | 0 changes | 0 authors, 0 changes
public void ReqUpdate(List<string> queries, Dictionary<string, object> parameters)
{
    MySqlCommand command;
    MySqlConnection transaction = connection.BeginTransaction();
    try
    {
        foreach (string stringQuery in queries)
        {
            command = new MySqlCommand(stringQuery, connection, transaction);
            if (!(parameters is null))
            {
                foreach (KeyValuePair<string, object> parameter in parameters)
                {
                    command.Parameters.Add(new MySqlParameter(parameter.Key, parameter.Value));
                }
            }
            command.Prepare();
            command.ExecuteNonQuery();
        }
        transaction.Commit();
    }
    catch (MySqlException e)
    {
        transaction.Rollback();
        Console.WriteLine(e.Message);
        throw;
    }
    catch (InvalidOperationException e)
    {
        ErreurGraveBddNonAccessible(e);
    }
}
```

Figure 36: Adaptation de la fonction 'ReqUpdate'

Maintenant il était possible de définir les fonctions 'CreerLivre' et 'ModifLivre' dans le Dao.

```
/// <summary>
/// écriture d'un livre en base de données
/// </summary>
/// <param name="livre"></param>
/// <returns>true si l'insertion a pu se faire</returns>
1 reference | 0 changes | 0 authors, 0 changes
public static bool CreerLivre(Livre livre)
{
    try
    {
        List<string> requetes = new List<string>();
        requetes.Add("insert into document values (@id, @titre, @image, @idRayon, @idPublic, @idGenre)");
        requetes.Add("insert into livres_dvd values (@id)");
        requetes.Add("insert into livre values (@id, @isbn, @auteur, @collection)");
        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@id", livre.Id },
            { "@titre", livre.Titre },
            { "@image", livre.Image },
            { "@idRayon", livre.IdRayon },
            { "@idPublic", livre.IdPublic },
            { "@idGenre", livre.IdGenre },
            { "@isbn", livre.Isbn },
            { "@auteur", livre.Auteur },
            { "@collection", livre.Collection },
        };
        BddMySQL curs = BddMySQL.GetInstance(connectionString);
        curs.ReqUpdate(requetes, parameters);
        curs.Close();
        return true;
    }
    catch
    {
        return false;
    }
}
```

Figure 37: Création d'un nouveau livre

Beaucoup de changements ont été faits, il convenait de faire un autre commit & push. Par ailleurs, en traitant les fonctionnalités d'ajout/modification j'ai géré la tâche concernant la protection de modification d'identifiant d'un document aussi, du coup j'ai mis à jour le storyboard.

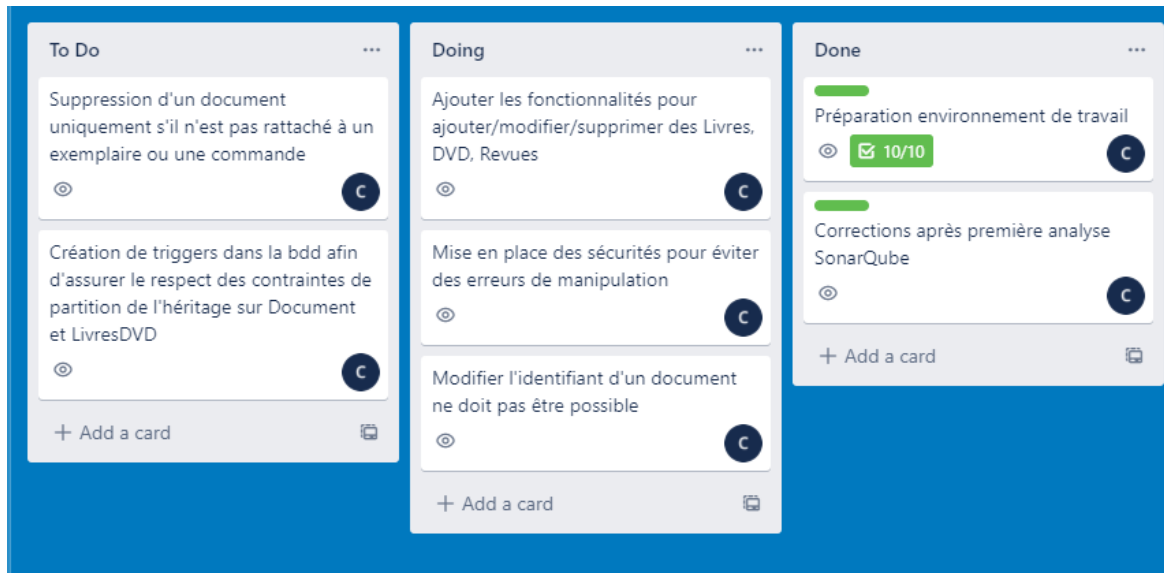


Figure 38: Mise à jour du Storyboard

6.4 Suppression de livres

Au niveau de l'onglet des livres il ne restait uniquement la suppression à gérer. La suppression ne doit uniquement être possible si le document en question n'est pas rattaché à un exemplaire ou une commande. Deux approches étaient possibles :

Je pourrais faire cette vérification dans l'application même. Dans ce cas là il y aurait deux appels à la base de données : un pour vérifier l'existence d'exemplaire ou commande concernant le même identifiant puis un deuxième pour (éventuellement) faire la suppression.

Une autre possibilité était de faire un seul appel à la base de données, avec une demande de suppression, puis de gérer un trigger dans la base de données pour faire la vérification. Cette solution me semblait plus optimale.

Run SQL query/queries on database mediatek86:

```
1 DROP TRIGGER IF EXISTS delDocument;
2 DELIMITER //
3 CREATE TRIGGER delDocument
4     BEFORE DELETE ON document
5     FOR EACH ROW
6 BEGIN
7     DECLARE countExemplaire INTEGER;
8     DECLARE countAbonnement INTEGER;
9     DECLARE countCommandeDoc INTEGER;
10    SELECT COUNT(*) INTO countExemplaire FROM exemplaire WHERE id = old.id;
11    SELECT COUNT(*) INTO countAbonnement FROM abonnement WHERE idRevue = old.id;
12    SELECT COUNT(*) INTO countCommandeDoc FROM commandedocument WHERE idLivreDvd = old.id;
13    IF countExemplaire = 1 OR countAbonnement = 1 OR countCommandeDoc = 1 THEN
14        SIGNAL SQLSTATE "45000";
15    END IF;
16 END;
17 // DELIMITER ;
```

Figure 39: Trigger pour la suppression d'un document

Ensuite il fallait gérer le clic sur le bouton 'Supprimer' avec une demande de confirmation de l'utilisateur avant de procéder. Du coup j'ai créé une fonction 'ValidationSuppression' qui pourra être utilisé pour les trois onglets.

```
/// <summary>
/// Affichage d'un MessageBox pour demander validation de suppression d'un document
/// </summary>
/// <param name="titre">Le titre du document concerné</param>
/// <returns>True si suppression confirmée, sinon false</returns>
1 reference | 0 changes | 0 authors, 0 changes
private bool ValidationSuppression(string titre)
{
    return (MessageBox.Show("Etes-vous sûr de vouloir supprimer '" + titre + "' ?", "Confirmation
    de suppression", MessageBoxButtons.YesNo) == DialogResult.Yes);
}
```

Figure 40: Demande de validation de suppression

Par la suite il fallait créer les fonctions 'SupprLivre' dans le contrôleur et le Dao :

```
/// <summary>
/// Supprime un livre dans la bdd
/// </summary>
/// <param name="id">L'id du livre à supprimer</param>
/// <returns>True si la suppression a pu se faire</returns>
1 reference | 0 changes | 0 authors, 0 changes
public bool SupprLivre(string id)
{
    return Dao.SupprLivre(id);
}
```

Figure 41: Méthode de suppression dans le contrôleur

```
/// <summary>
/// Suppression d'un livre de la base de données
/// </summary>
/// <param name="id">identifiant du livre à supprimer</param>
/// <returns>true si la modification a pu se faire</returns>
1 reference | 0 changes | 0 authors, 0 changes
public static bool SupprLivre(string id)
{
    try
    {
        List<string> requetes = new List<string>();
        requetes.Add("delete from livre where id=@id");
        requetes.Add("delete from livres_dvd where id=@id");
        requetes.Add("delete from document where id=@id");
        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@id", id },
        };
        BddMySQL curs = BddMySQL.GetInstance(connectionString);
        curs.ReqUpdate(requetes, parameters);
        curs.Close();
        return true;
    }
    catch
    {
        return false;
    }
}
```

Figure 42: Méthode de suppression dans le Dao

... et implémenter la méthode événementielle suite à un clic sur le bouton 'Supprimer' :

```

/// <summary>
/// Événement clic sur le bouton 'Supprimer'. Vérifie validation de l'utilisateur avant de procéder.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | 0 changes | 0 authors, 0 changes
private void btnSupprLivre_Click(object sender, EventArgs e)
{
    if(ValidationSuppression(txbLivresTitre.Text))
    {
        if (controle.SupprLivre(txbLivresNumero.Text))
        {
            controle.RefreshAllLivres();
            lesLivres = controle.GetAllLivres();
            RemplirLivresListeComplete();
        }
        else
        {
            MessageBox.Show("Il n'est pas possible de supprimer ce document car ils existent un ou plusieurs exemplaires ou commandes le concernant.", "Erreur");
        }
    }
}

```

Figure 43: Méthode événementielle de suppression, dans la vue

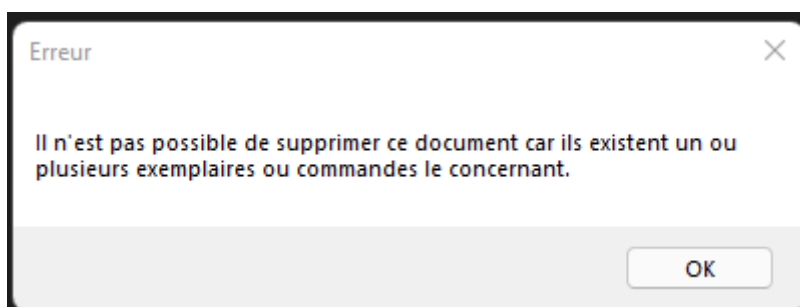


Figure 44: Boîte de dialogue en cas d'impossibilité de suppression

Après quelques tests pour vérifier si tout fonctionnait à souhait j'ai fait un nouveau commit & push. Effectivement, les différentes fonctionnalités étaient maintenant en place pour les livres, mais il fallait les implémenter pour les deux autres onglets aussi. La logique étant similaire, il suffit d'adapter quelques intitulés, champs, ... mais il y aurait beaucoup de code à insérer. Autant partir sur de bonnes bases.

```

Phase 4: Ajouter fonctionnalités pour ajouter/modifier/supprimer des...
- Livres, DVD, Revues + sécurisation des manipulations

- création d'un trigger dans la bdd : 'delDocument' (vérifie la présence de l'id du document à supprimer dans les tables exemplaire, abonnement et commandedocument)
- ajout de fonction 'ValidationSuppression' dans la vue
- ajout de fonctions 'SupprLivre' dans le contrôleur et le Dao
- ajout de méthode événementielle suite clic sur le bouton 'supprimer'

carlfremault committed 28 seconds ago

```

Figure 45: Commit pour la phase 4

6.5 Implémentation des fonctionnalités ajout/modification/suppression pour les onglets DVD et Revue

Pour la plupart des fonctionnalités il suffisait de copier et modifier les différentes méthodes mises en place pour l'onglet 'Livres'. Quelques spécificités étaient à gérer cependant :

Tout d'abord, le contexte ne donne pas de précisions par rapport à l'obligation de saisie des champs. La configuration de la base de données n'oblige uniquement le numéro de document, le genre, le public et le rayon. Il semblait logique toutefois d'obliger la saisie du titre, réalisateur et de la durée pour les DVD, puis le titre, périodicité et délai de mise à disposition pour les revues.

Les champs 'durée' des DVD et 'délai de mise à disposition' des Revues étaient des champs texte, cependant l'enregistrement dans la base de données est sous forme d'entier.

J'ai utilisé la méthode 'TryParse' pour transtyper la valeur 'texte' du champ en entier. Contrairement à la méthode habituelle 'Parse', TryParse retourne un booléen et ne génère pas d'exception si le transtypage ne serait pas possible. Cela permet de gérer la validation du champ de façon plus élégante :

Voici un extrait de la méthode événementielle 'btnEnregistrerRevue_Click', au moment de la valorisation des paramètres nécessaires à la création d'un nouvel objet 'Revue' :

```
int delaiMiseADispo = 0;
if (txbRevuesDateMiseADispo.Text != "")
{
    bool success = int.TryParse(txbRevuesDateMiseADispo.Text, out delaiMiseADispo);
    if (!success)
    {
        MessageBox.Show("La valeur saisie pour le délai de mise à dispo doit être un entier.", "Erreur");
        txbRevuesDateMiseADispo.Text = "";
        txbRevuesDateMiseADispo.Focus();
        return;
    }
}
```

Figure 46: Utilisation de 'TryParse'

J'ai utilisé le même procédé pour gérer le champ 'durée' de l'onglet DVD.

En faisant mes tests je me suis rendu compte que je n'avait pas sécurisé le changement des onglets. Effectivement l'utilisateur qui est en train de faire une saisie doit être averti et il doit pouvoir choisir d'abandonner sa saisie ou non.

Ainsi j'ai ajouté une méthode événementielle qui se déclenche lors d'un clic sur un des onglets. La méthode vérifie si une saisie est en cours. Si c'est le cas une confirmation est demandée avant d'abandonner la saisie et changer d'onglet :

```

/// <summary>
/// Événement de changement d'onglet. Vérifie si une saisie est en cours
/// Si oui, demande validation de l'utilisateur avant d'abandonner saisie et changer d'onglet
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | 0 changes | 0 authors, 0 changes
private void tabOngletsApplication_Deselecting(object sender, TabControlCancelEventArgs e)
{
    if ((saisieLivre || saisieDvd || saisieRevue) && !VerifAbandonSaisie())
    {
        e.Cancel = true;
    }
    else
    {
        CancelAllSaisies();
    }
}

```

Figure 47: Événement de changement d'onglet

Pour faciliter la procédure j'ai également ajouté une fonction 'CancelAllSaisies' qui désactive tous les booléens concernant les saisies et modifications. Il était temps de faire un autre commit & push ...

6.6 Création de trigger pour contrôler les contraintes de partition d'héritage sur Document et LivresDvd

Certaines des protections étaient en place avec la configuration de la base de données (il n'est pas possible de créer un document dont l'id existe déjà, par exemple), toutefois certaines 'mauvaises manipulations' pouvaient passer. En effet, si un Livre avec un certain id existe, l'id existe dans les tables document, livres_dvd et livre, mais il est tout à fait possible d'ajouter un tuple dans la table dvd tant que son id existe bien dans livres_dvd (car c'est une clé étrangère).

Même si, de par la nature de l'implémentation des insertions dans la base de données (dans une transaction), l'application ne permettait pas cette manipulation (par exemple : une création de livre insère d'un coup dans les trois tables document, livres_dvd et livre), il est toujours prudent d'ajouter cette protection supplémentaire.

Le principe est le même pour l'héritage revue - livres_dvd sur document.

J'ai donc commencé par créer ces trigger dans la base de données.

Comme il n'est pas possible de modifier l'identifiant d'un document j'avais juste à m'occuper de l'insertion de nouveaux tuples. Effectivement, si on modifie un document, cela implique qu'il a été créé auparavant ...

Voici par exemple le trigger sur insertion de revue (les autres étant similaires) :

```
Run SQL query/queries on table mediatek86.revue: ⓘ

1 DROP TRIGGER IF EXISTS insRevue;
2 DELIMITER //
3 CREATE TRIGGER insRevue
4     BEFORE INSERT ON revue
5     FOR EACH ROW
6 BEGIN
7     DECLARE countId INTEGER;
8     SELECT COUNT(*) INTO countId FROM livres_dvd WHERE id = new.id;
9     IF countId > 0 THEN
10         SIGNAL SQLSTATE "45000"
11         SET MESSAGE_TEXT = "Un livre ou DVD avec cet identifiant existe déjà.";
12     END IF;
13 END;
14 // DELIMITER ;
```

Figure 48: Trigger sur l'insertion d'une revue

Puis j'ai fait quelques tests directement dans le SGBDR pour m'assurer du bon fonctionnement des trigger :



Figure 49: Le trigger fonctionne

Maintenant il fallait gérer l'arrivée de ces messages d'erreur dans l'application (même si, comme expliqué ci-dessus, à priori ces manipulations ne devraient pas être possible d'emblée).

Pour l'instant les différentes méthodes de création de document retournent un booléen. Il faudra adapter pour qu'ils puissent retourner le message d'erreur.

```
BddMySQL curs = BddMySQL.GetInstance(connectionString);  
curs.ReqUpdate(requetes, parameters);  
curs.Close();  
return "Ajout de livre réussi!";  
}  
catch (Exception e)  
{  
    return e.Message;  
}
```

Figure 50: Modification des méthodes pour retourner une chaîne de caractères



Figure 51: Boîte de dialogue résultante

Il fallait gérer aussi le retour de message d'erreur envoyé par SGBDR lors d'une tentative d'insertion avec un identifiant existant car il était peu élégant :

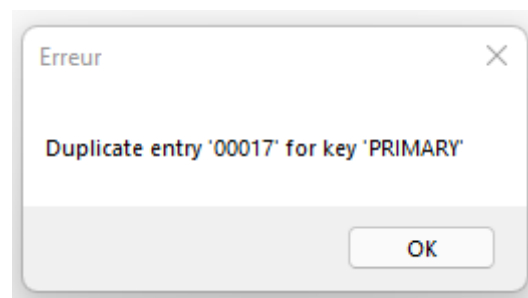


Figure 52: Ancien message d'erreur suite duplicata

Du coup les fonctionnalités demandées étaient tous implémentées. Un commit & push s'imposait. Avant de faire le merge avec la branche master j'ai fait passer SonarLint et SonarQube sur le code. Tout passait sauf que SonarQube m'a fait une remarque sur la répétition de code, au niveau de la vue et du Dao.

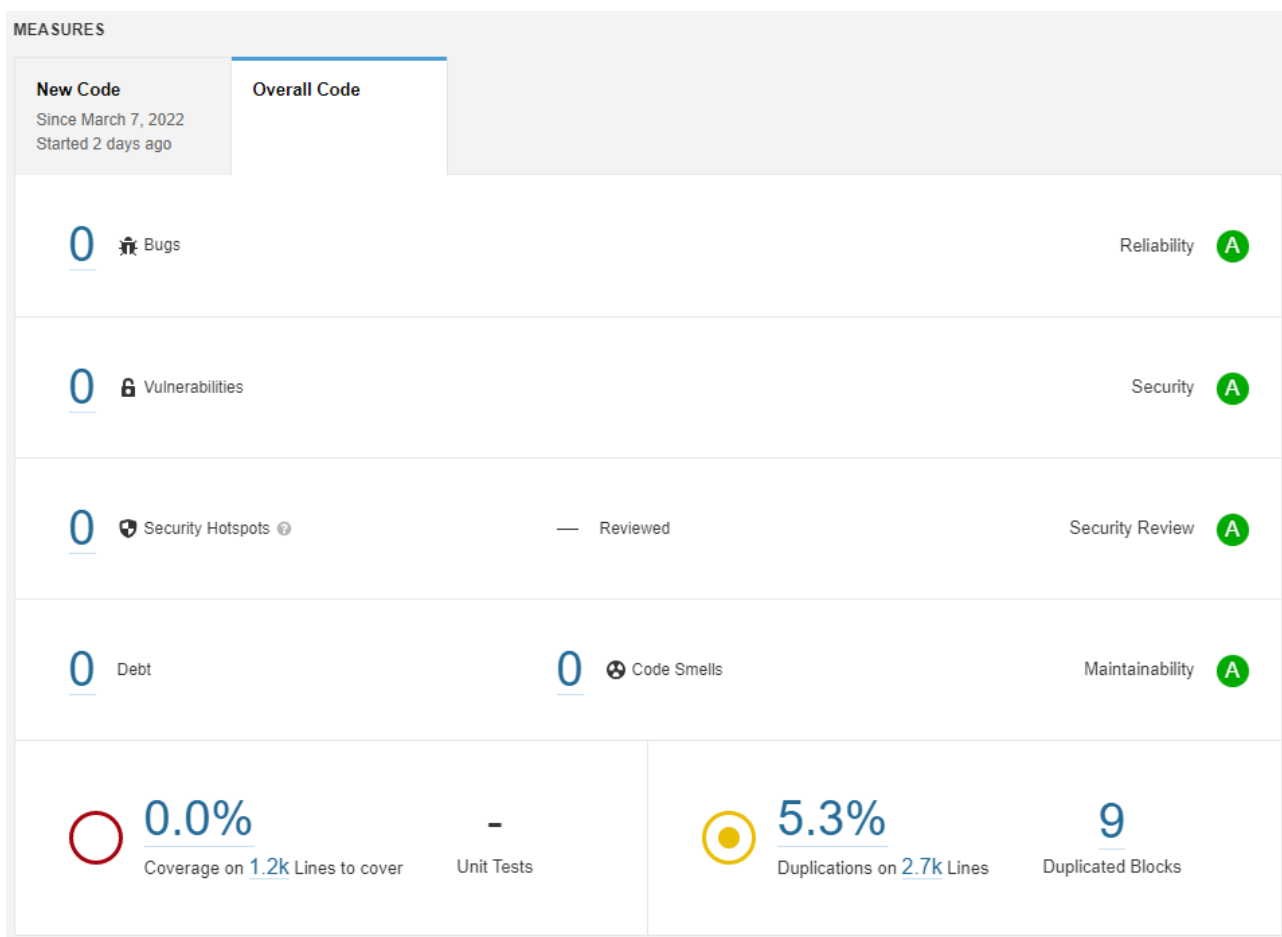


Figure 53: Résultat analys SonarQube

Dans la vue il y avait effectivement quelques répétitions qui n'étaient toutefois pas évitables. Avant d'enregistrer un document il faut récupérer les différents éléments, dont quelques qui sont les mêmes pour les trois cas de figure (genre, public, rayon). J'estime que ce ne serait pas optimisé de créer une fonction séparée juste pour ça.

```

{
    MessageBox.Show("Les champs marqués d'un astérisque (*) sont obligatoires.", "Information");
    return;
}

Genre leGenre = (Genre)bdgInfosGenres.List[bdgInfosGenres.Position];
String idGenre = leGenre.Id;
String genre = leGenre.ToString();
Public lePublic = (Public)bdgInfosPublics.List[bdgInfosPublics.Position];
String idPublic = lePublic.Id;
String unPublic = lePublic.ToString();
Rayon leRayon = (Rayon)bdgInfosRayons.List[bdgInfosRayons.Position];
String idRayon = leRayon.Id;
String rayon = leRayon.ToString();
String id = txtRevuesNumero.Text;

```

Figure 54: Les répétitions de code, signalés par un trait gris foncé (le trait rouge concerne la couverture par des tests unitaires)

Pour ce qui est les lignes dupliquées dans le Dao, j'ai choisi de ne pas y remédier non plus pour l'instant. Certes, l'instanciation du curseur est le même dans chaque fonction, et le Dictionnaire des paramètres est identique à chaque fois pour les manipulations d'insertion et de 'update'. À nouveau, je ne suis pas convaincu si c'est mieux d'extraire ces lignes dans une fonction séparée. Il y aurait énormément de paramètres à gérer et la lisibilité et la modularité du code souffrirait. Pour l'instant j'ai donc décidé d'ignorer ces remarques SonarQube et je procède avec le merge.

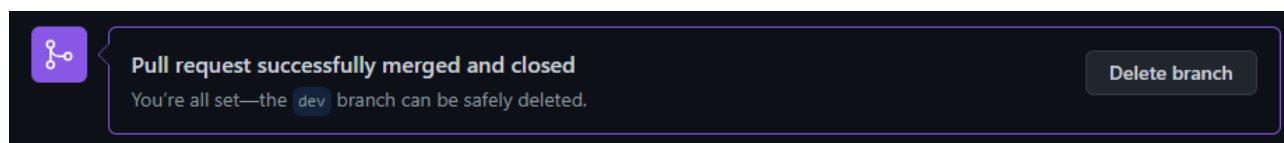


Figure 55: Merge de la branche 'dev' avec la branche principale

6.7 Bilan pour la mission 1

Cette mission était volumineuse, dû aux changements répétés pour chaque onglet. Dans l'ensemble je suis satisfait du résultat. Les fonctionnalités demandées ont été implémentées, et les différentes mesures pour sécuriser l'application et protéger l'intégrité de la base de données sont en place.

Je ne suis pas entièrement sûr toutefois si j'ai choisi l'implémentation la plus 'simple'. Le nombre de lignes de code ajouté dans la vue est impressionnant. Ceci dit, la plupart des cas reposent sur la même logique (vérification si une saisie est en cours, demande de confirmation d'annulation) et sont implémentés de façon identique.

Il aurait été possible de le faire de façon plus simple, mais cela aurait sacrifié certains confort pour les utilisateurs. J'aurais pu tout simplement ouvrir une nouvelle fenêtre pendant une saisie mais je peux m'imaginer des cas de figure où un employé fait une recherche et aimerait consulter la liste de résultats pendant une saisie. Il en est de même si j'aurais juste désactivé la zone de recherche puisque la liste ne peut plus défiler dans ce cas là.

Alors bloquer toutes les zones sauf la liste déroulante, en désactivant la fonctionnalité qui affiche les détails d'un document quand il est sélectionné dans la liste ? Dans ce cas là on aura un même élément avec un comportement qui est différent en fonction de la manipulation en cours (recherche ou saisie) ... Ce ne serait pas optimal d'un point de vue utilisateur non plus.

Pour ce qui est l'enregistrement de changements dans la base de données je suis satisfait du procédé mis en place. Comme tout 'reqUpdate' se passe dorénavant dans une transaction, cela apporte une sécurité de plus pour l'intégrité de la base de données.

7. Mission 2 : Gérer les commandes

Voici les tâches qui m'ont été confiées pour la deuxième mission :

- Création d'une table 'suivi' dans la bdd, contenant les différentes étapes de suivi d'une commande, reliée à CommandeDocument
 - En cours
 - Relancée
 - Livrée
 - Réglée
- Création d'un onglet pour gérer les commandes de livres
 - Sélection de livre par son numéro, affichage des détails du livre et des commandes le concernant, triés par date (ordre inverse de la chronologie)
 - Date de commande
 - Montant
 - Nombre d'exemplaires
 - Étape de suivi de la commande
 - Groupbox pour saisir et enregistrer une nouvelle commande
 - État 'en cours' lors de l'enregistrement d'une nouvelle commande
 - Modification de l'état d'une commande en respectant des contraintes
 - 'Livree' ou 'réglée' ne peut pas revenir en arrière
 - 'Réglée' n'est pas possible si la commande n'est pas 'Livree'
 - Suppression d'une commande uniquement si elle n'est pas Livree
 - Création de trigger : si une commande est Livree, génération automatique d'exemplaires
 - Date d'achat = date de commande
 - État de l'exemplaire = 'neuf'
 - Numéro d'exemplaire séquentiel par rapport au livre concerné
- Création d'onglet pour gérer les commandes de DVD avec les mêmes fonctionnalités

- Création d'onglet pour gérer les commandes (nouvel abonnement ou renouvellement) de Revues
 - Dans le cas d'un nouvel abonnement la revue doit être créée au préalable dans l'onglet Revues
 - Sélection de revue par son numéro, affichage des détails de la revue et des commandes la concernant, triés par date (ordre inverse de la chronologie)
 - Date de commande
 - Montant
 - Date de fin d'abonnement
 - Groupbox pour saisir et enregistrer une nouvelle commande (même principe pour un nouvel abonnement ou un renouvellement)
 - Suppression de commande uniquement si aucun exemplaire n'est rattaché (un exemplaire est 'rattaché' si la date de l'achat de l'exemplaire est comprise entre la date de commande et la date de fin d'abonnement)
 - Création de méthode 'ParutionDansAbonnement' pour vérifier cela
 - Créer un test unitaire pour cette méthode
- Présentation des onglets de commande à l'identique de l'onglet 'Parutions des revues'
 - Tri sur les colonnes dans les listes
- Mise en place des sécurités pour éviter les erreurs de manipulation
- Création de trigger pour contrôler la contrainte de partition de l'héritage sur commande
- Création de procédure stockée dans la bdd pour obtenir la liste des revues dont l'abonnement se termine dans moins de 30 jours
- Apparition d'une fenêtre d'alerte dès l'ouverture de l'application rappelant cette liste, trié sur la date dans l'ordre chronologique

J'ai mis à jour le 'Storyboard' et j'en ai profité pour ajouter des codes couleurs :

- vert pour toute tâche qui concerne le code C#
- orange pour toute tâche qui concerne des manipulations dans la bdd (triggers, procédures)
- jaune pour des tâches plus générales de type 'configuration' (par exemple la mise en place de l'environnement de travail en début de l'atelier)

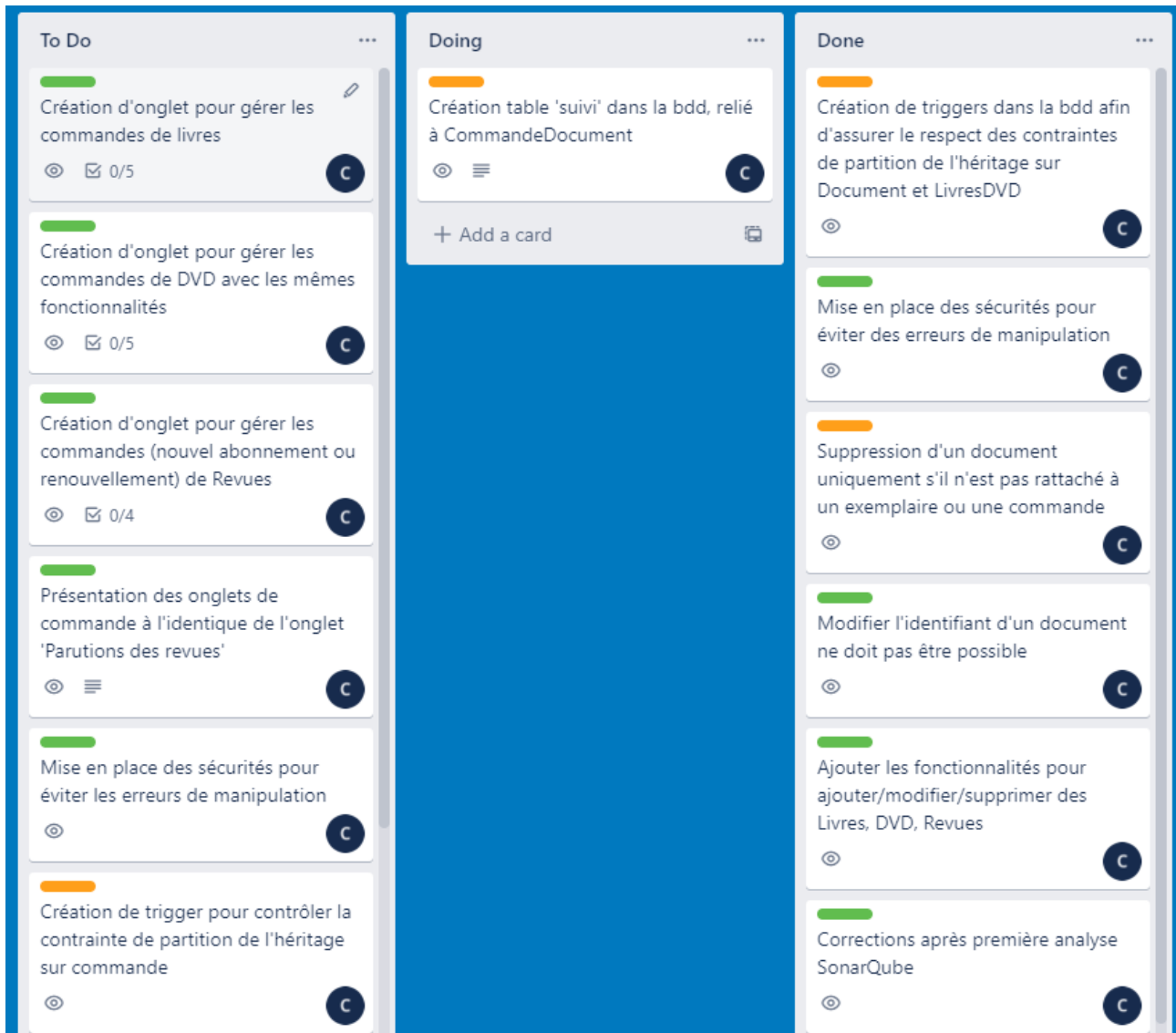


Figure 56: Storyboard pour la mission 2

7.1 Création de table 'suivi' dans la bdd

J'ai commencé par créer une table 'suivi' avec deux champs: id et libellé.

Si j'aurais pu insérer cette table dès la conception de l'application, et de la base de données, il aurait été possible d'ajouter un champ 'idsuivi' en tant que clé étrangère, en référant 'id' de 'suivi', dans la table 'commandedocument'.

Toutefois, la base de données existait déjà, il y avait des tuples d'enregistrés et une application qui l'exploite. Du coup il est dangereux de modifier l'existant, car cela pourrait générer des erreurs et possiblement une perte de données. J'ai donc créé une deuxième table 'suivicommandedoc' avec une clé primaire composée de 'idsuivi' (en référence à id de suivi) et 'idcommande' (en référence à id de commandedocument).

```

1 CREATE TABLE suivi (
2     id integer (3) NOT NULL,
3     libelle varchar (20) NOT NULL
4 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
5
6 CREATE TABLE suivicommandedoc (
7     idsuivi integer (3) NOT NULL,
8     idcommande varchar (5) NOT NULL
9 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
10
11 INSERT INTO suivi (id, libelle) VALUES
12     ('001', 'En cours'),
13     ('002', 'Relancée'),
14     ('003', 'Livrée'),
15     ('004', 'Réglée');
16
17 ALTER TABLE suivi
18     ADD PRIMARY KEY (id);
19
20
21 ALTER TABLE suivicommandedoc
22     ADD PRIMARY KEY (idsuivi, idcommande);
23
24 ALTER TABLE suivicommandedoc
25     ADD CONSTRAINT suivicommandedoc_fk1 FOREIGN KEY (idsuivi) REFERENCES suivi (id),
26     ADD CONSTRAINT suivicommandedoc_fk2 FOREIGN KEY (idcommande) REFERENCES commandedocument (id);
27

```

Figure 57: Insertion des nouvelles tables dans la base de données

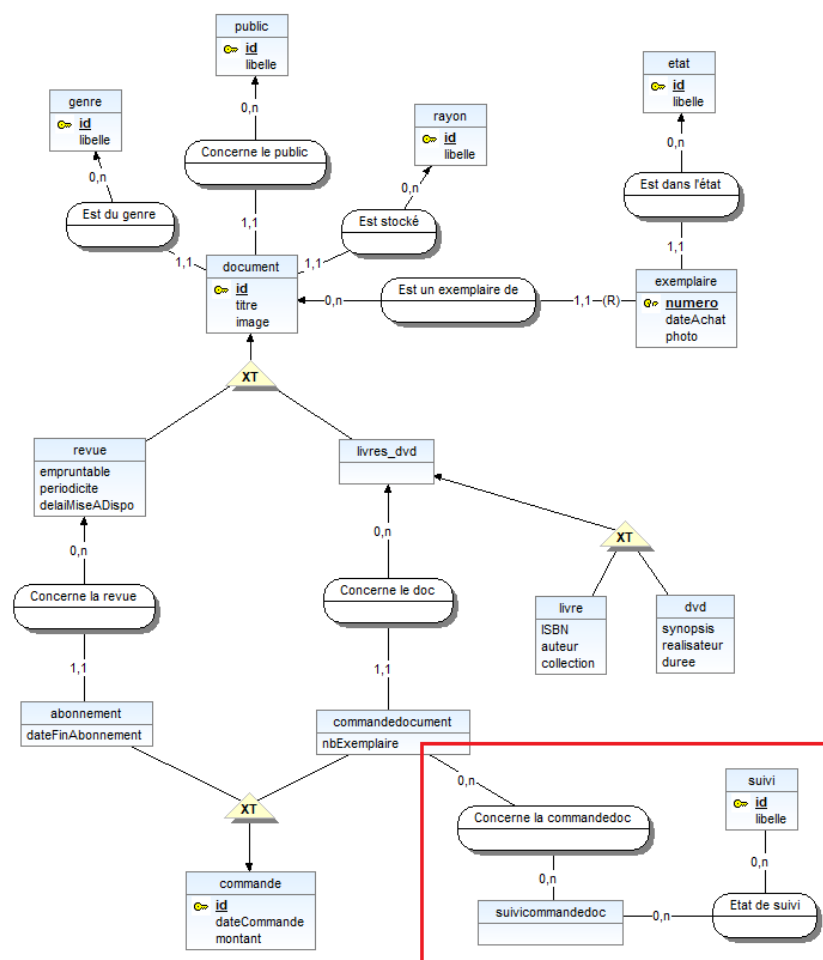


Figure 58: Nouveau schéma de la base de données (ajouts encadrés en rouge)

Ensuite j'ai crée trois nouvelles classes métier dans l'application. Deux dont les tables existaient déjà dans la base de données : Commande et CommandeDocument, et Suivi.

7.2 Création d'onglets pour gérer les commandes de livres et de DVD, et abonnements de revues. Implémentation zones recherche

Plutôt de faire un onglet entièrement (affichage, recherche, commande) avant de passer au suivant, personnellement j'ai préféré regrouper les tâches qui se ressemblent. Ainsi j'ai décidé de prendre en charge les trois kanban en même temps. Comme j'ai crée des checklist, je pourrai suivre l'avancement de chaque kanban en détail.

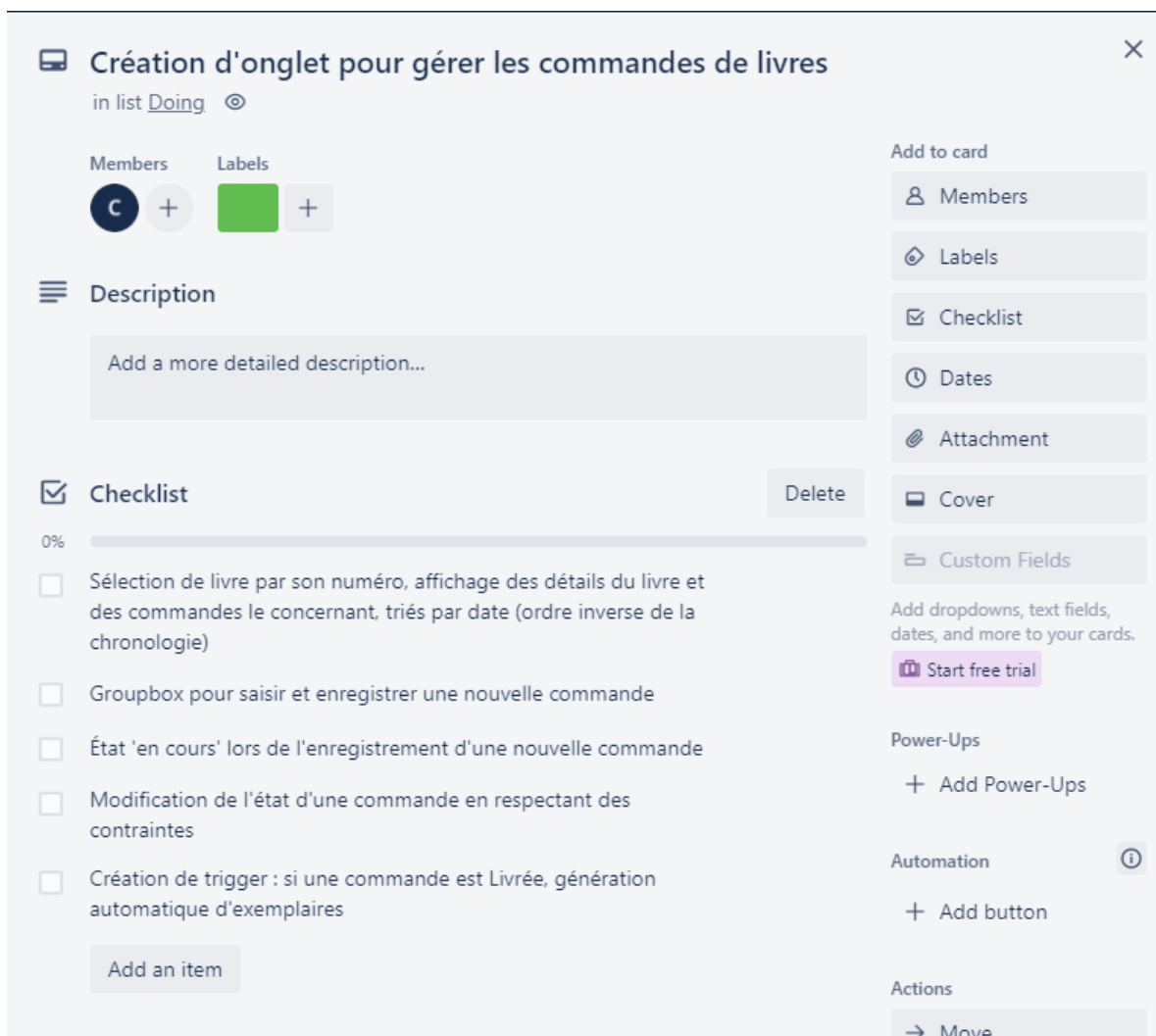


Figure 59: Exemple de Kanban avec checklist

Pour commencer j'ai créé un nouvel onglet 'Commande de livres' puis j'ai inséré le groupBox de recherche de livre, à l'identique de l'onglet de Parutions de revues (comme demandé dans la mission).

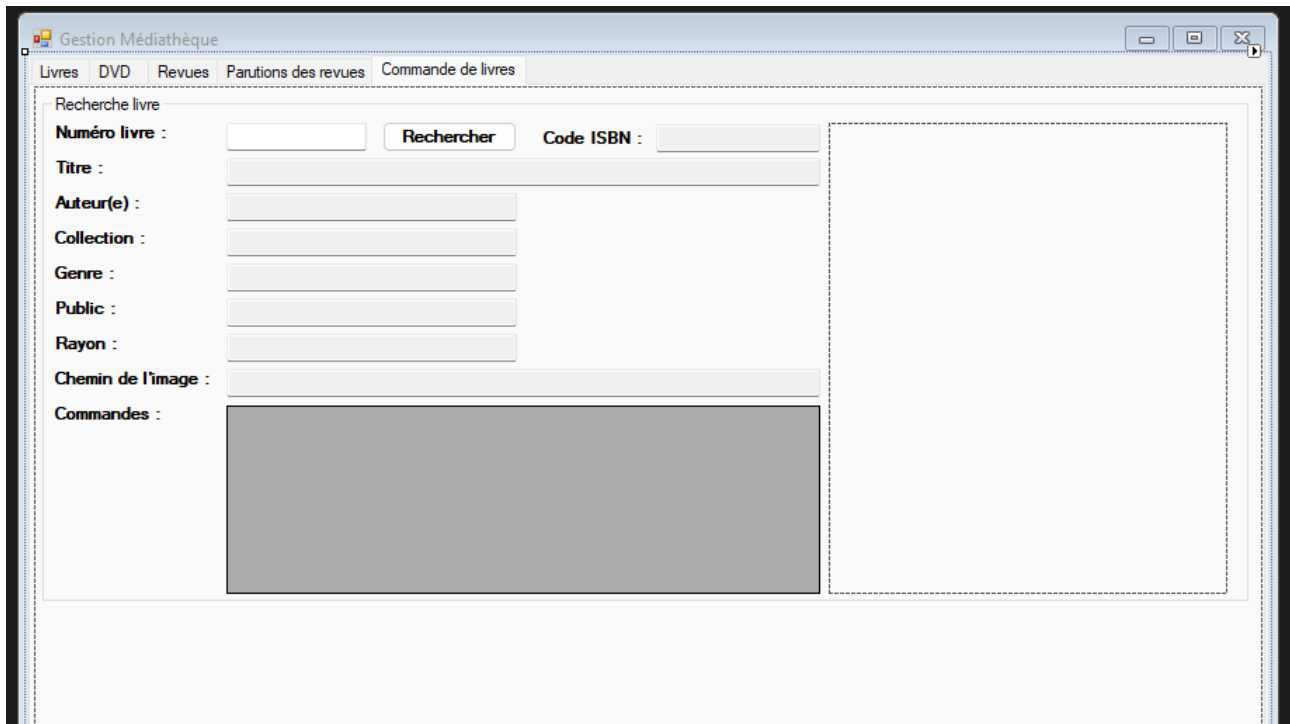


Figure 60: Début de la nouvelle vue

Ensuite il fallait incorporer les différentes méthodes pour gérer la recherche d'un livre. Comme la fonctionnalité était similaire à celle utilisée dans l'onglet des Parutions des revues, j'ai copié ce code pour l'adapter ensuite (noms des variables, des champs, ...). Les méthodes sont similaires à celles utilisées dans les onglets de gestion de Livres, DVD et Revues (remplissage du DataGridView, remplissage des champs, ...).

Il fallait également créer une méthode `GetCommandeDocument` dans le contrôleur et le Dao afin de pouvoir récupérer les commandes de la base de données. Cette méthode pourra être utilisée aussi bien pour les livres que les DVD :


```

/// <summary>
/// Retourne les commandes d'un livre ou d'un DVD
/// </summary>
/// <param name="idDoc">Identifiant du livre ou DVD</param>
/// <returns>Liste d'objets CommandeDocument</returns>
1 reference | Carl Fremault, 22 hours ago | 1 author, 1 change
public static List<CommandeDocument> GetCommandeDocument(string idDoc)
{
    List<CommandeDocument> lesCommandes = new List<CommandeDocument>();
    string req = "Select c.id, c.dateCommande, c.montant, cd.nbExemplaire, cd.idLivreDvd,
        scd.idSuivi, s.libelle ";
    req += "from commande c join commandedocument cd on c.id=cd.id ";
    req += "join suivicommandedoc scd on c.id=scd.idcommande ";
    req += "join suivi s on scd.idsuivi = s.id ";
    req += "where cd.idLivreDvd = @id ";
    req += "order by c.dateCommande DESC";
    Dictionary<string, object> parameters = new Dictionary<string, object>
    {
        { "@id", idDoc}
    };

    BddMySQL curs = BddMySQL.GetInstance(connectionString);
    curs.ReqSelect(req, parameters);

    while (curs.Read())
    {
        string id = (string)curs.Field("id");
        DateTime dateCommande = (DateTime)curs.Field("datecommande");
        double montant = (double)curs.Field("montant");
        int nbExemplaire = (int)curs.Field("nbExemplaire");
        string idLivreDvd = (string)curs.Field("idLivreDvd");
        int idSuivi = (int)curs.Field("idSuivi");
        string libelleSuivi = (string)curs.Field("libelle");
        CommandeDocument commandeDocument = new CommandeDocument(id, dateCommande, montant,
            nbExemplaire, idLivreDvd, idSuivi, libelleSuivi);
        lesCommandes.Add(commandeDocument);
    }
    curs.Close();

    return lesCommandes;
}

```

Figure 61: Récupération des 'CommandeDocument' depuis la base de données

J'ai manuellement inséré une première commande dans la base de données afin de faire un premier test. Le résultat était satisfaisant :

Recherche livre

Numéro livre :

00017

Rechercher

Titre :

Catastrophes au Brésil

Auteur(e) :

Philippe Masson

Code ISBN :

3,21457E+12

Collection :

Genre :

Policier

Public :

Ados

Rayon :

Jeunesse romans

Chemin de l'image :

C:\img\Mediatek\51Cc0m4ZNOL.jpg

Commandes :

Date	Montant	Exemplaires	Etat
11/03/2022	80.6	2	En cours
10/03/2022	100	5	En cours

Figure 62: Affichage des commandes d'un livre

Ensuite j'ai procédé à faire les mêmes implémentations pour les deux autres onglets. L'onglet 'Commande de DVD' était très similaire, à l'exception de quelques champs. L'onglet 'Abonnement Revues' était en peu différent cependant comme il n'y a pas de suivi à gérer. Il fallait également ajouter une classe métier 'Abonnement'.

J'ai aussi rajouté une petite fonctionnalité pour améliorer le confort utilisateur : taper "entrée" dans le champ de recherche d'un numéro (dans les onglets de commande de livre, de dvd, et d'abonnement à une revue) déclenche dorénavant la recherche.

```

/// <summary>
/// Entrée dans champ recherche déclenche la recherche aussi
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | 0 changes | 0 authors, 0 changes
private void txbRevuesNumRecherche_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        e.SuppressKeyPress = true;
        btnRevuesNumRecherche_Click(sender, e);
    }
}

```

Figure 63: Déclencher une recherche en tapant 'entrée' dans le textBox du numéro d'une revue

Puis tant que j'étais sur l'amélioration UX j'ai également mis les tabIndex des différents champs à l'équerre. Pas mal de changements ont été faits, temps de faire un commit & push. De plus, comme j'avais terminé la veille sans terminer une tâche/mission j'avais fait un push sur une branche 'temp' (pour ne pas avoir le travail uniquement en local).

J'ai donc fait un commit et push sur cette branche puis un merge avec la branche 'dev'.

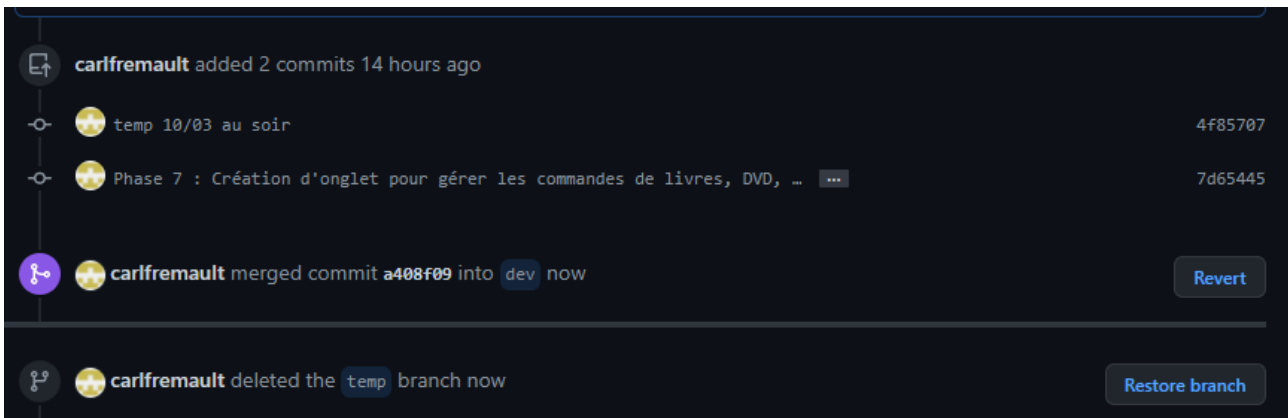


Figure 64: Merge et suppression de la branche temporaire

7.3 Implémentation de zones d'ajout/modification/suppression d'une commande ou d'un abonnement

Tout d'abord, depuis quelque temps je trouvais l'avancement bien moins efficace en raison du nombre important de lignes de code dans la vue (+3000). Une recherche sur internet m'a appris l'existence des 'classes partielles' ce qui répondait exactement au besoin : une seule classe peut être divisé en plusieurs fichiers 'physiques'. Du coup j'ai créé un fichier pour chaque onglet de la vue et transféré le code concerné dans chaque classe partielle.

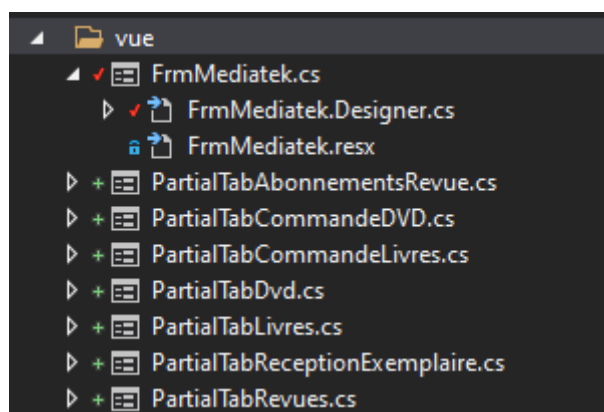


Figure 65: Division de la vue FrmMediatek en classes partielles

Ensuite, j'ai commencé par dessiner la zone de commandes dans l'onglet 'Commande de livres' pour la recopier dans les autres onglets concernés par la suite.

Figure 66: GroupBox pour la création d'une commande

Puis j'ai implémenté le code nécessaire au fonctionnement. C'est assez similaire que les saisies d'un nouveau livre, DVD, Il y avait toutefois moins de sécurisations à gérer concernant l'abandon éventuel de saisie (comme il y a moins d'interactions possibles dans la zone de recherche), et comme les possibilités de modification de commande étaient limitées et gérées avec les boutons.

```

/// <summary> Début de saisie de commande de livre
1 reference | Carl Fremault, 21 hours ago | 1 author, 3 changes
private void DebutSaisieCommandeLivres()
{
    AccesSaisieCommandeLivre(true);
}

/// <summary> Fin de saisie de commande de livre Affiche les informations de la ...
4 references | Carl Fremault, 4 days ago | 1 author, 2 changes
private void FinSaisieCommandeLivres()
{
    AccesSaisieCommandeLivre(false);
    CommandeLivresListeSelection();
}

/// <summary>
/// Actionne le booléen saisieCommandeLivres
/// Vide les champs de détails d'une commande
/// (Dés)active la protection readonly des champs de détails de commande
/// (Dés)active les boutons concernant l'ajout, validation et annulation de saisie de commande
/// </summary>
/// <param name="acces">'True' active les boutons 'Valider' et 'Annuler', désactive le bouton 'Ajouter', déverrouille
/// les champs des détails de commande</param>
2 references | Carl Fremault, 21 hours ago | 1 author, 2 changes
private void AccesSaisieCommandeLivre(bool acces)
{
    saisieCommandeLivres = acces;
    VideDetailsCommandeLivres();
    btnCommandeLivresValider.Enabled = acces;
    btnCommandeLivresAnnuler.Enabled = acces;
    btnCommandeLivresAjouter.Enabled = !acces;
    txtCommandeLivresNumeroCommande.Enabled = acces;
    dtpCommandeLivresDateCommande.Enabled = acces;
    nudCommandeLivresExemplaires.Enabled = acces;
    txtCommandeLivresMontant.Enabled = acces;
    grpCommandeLivres.Enabled = acces;
}

```

Figure 67: Sécurisation de la saisie de commandes

En revanche il fallait gérer les contraintes au niveau du suivi d'une commande, comme spécifié dans les besoins de mission :

```
/// <summary>
/// Activation des boutons de gestion de commande en fonction de l'état de suivi
/// </summary>
/// <param name="commandeDocument"></param>
1 reference | 0 changes | 0 authors, 0 changes
private void ActivationModificationCommandeLivres(CommandeDocument commandeDocument)
{
    string etatSuivi = commandeDocument.LibelleSuivi;
    switch (etatSuivi)
    {
        case "En cours":
        case "Relancée":
            btnCommandeLivresRelancer.Enabled = true;
            btnCommandeLivresConfirmerLivraison.Enabled = true;
            btnCommandeLivresRegler.Enabled = false;
            btnCommandeLivresSupprimer.Enabled = true;
            break;
        case "Livrée":
            btnCommandeLivresRelancer.Enabled = false;
            btnCommandeLivresConfirmerLivraison.Enabled = false;
            btnCommandeLivresRegler.Enabled = true;
            btnCommandeLivresSupprimer.Enabled = false;
            break;
        case "Réglée":
            DesActivationModificationCommandeLivres();
            break;
    }
}
```

Figure 68: Activation des boutons suivant l'état de suivi actuel

J'ai aussi fait une petite modification au niveau du DataGridView, afin d'assurer un affichage correct pour le montant (à savoir : à deux chiffres après la virgule et en ajoutant le signe '€') :

```

/// <summary>
/// Remplit le datagrid avec la liste reçue en paramètre
/// </summary>
3 references | 0 changes | 0 authors, 0 changes
private void RemplirCommandeLivresListe(List<CommandeDocument> lesCommandeDocument)
{
    bdgCommandesLivresListe.DataSource = lesCommandeDocument;
    dgvCommandeLivresListe.DataSource = bdgCommandesLivresListe;
    dgvCommandeLivresListe.Columns["id"].Visible = false;
    dgvCommandeLivresListe.Columns["idSuivi"].Visible = false;
    dgvCommandeLivresListe.Columns["idLivreDvd"].Visible = false;
    dgvCommandeLivresListe.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
    dgvCommandeLivresListe.Columns[6].DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleRight;
    dgvCommandeLivresListe.Columns[6].DefaultCellStyle.Format = "c2";
    dgvCommandeLivresListe.Columns[6].DefaultCellStyle.FormatProvider = CultureInfo.GetCultureInfo("fr-FR");
    dgvCommandeLivresListe.Columns["dateCommande"].DisplayIndex = 0;
    dgvCommandeLivresListe.Columns["montant"].DisplayIndex = 1;
    dgvCommandeLivresListe.Columns[4].HeaderCell.Value = "Date";
    dgvCommandeLivresListe.Columns[0].HeaderCell.Value = "Exemplaires";
    dgvCommandeLivresListe.Columns[2].HeaderCell.Value = "Etat";
}

```

Figure 69: Remplissage de la liste de commandes

Ensuite il fallait implémenter les interactions avec la base de données. D'abord je me suis occupé de l'ajout de commande.

Afin de pouvoir attribuer un état de suivi à une commande, le contrôleur devait disposer d'une Liste avec les différents Suivis. Dans ce but j'ai ajouté des méthodes GetAllSuivis dans le contrôleur et le Dao (dans le même esprit que GetAllPublics, GetAllGenres et GetAllRayons), ainsi que des propriétés 'lesSuivis' (de type Liste de Suivi) dans le contrôleur et dans la vue.

Après j'ai pu créer la méthode événementielle qui est déclenché lors d'un clic sur le bouton de validation de commande, et les méthodes correspondantes d'ajout de commandeDocument dans le contrôleur et le Dao (qui eux sont similaires aux autres méthodes d'ajout dans la base de données).

```

private void btnCommandeLivresValider_Click(object sender, EventArgs e)
{
    if (txbCommandeLivresNumeroCommande.Text == "" || txbCommandeLivresMontant.Text == "")
    {
        MessageBox.Show("Tous les champs sont obligatoires.", "Information");
        return;
    }

    String id = txbCommandeLivresNumeroCommande.Text;
    DateTime dateCommande = dtpCommandeLivresDateCommande.Value;
    int nbExemplaires = (int)nudCommandeLivresExemplaires.Value;
    string idLivreDvd = txbCommandeLivresNumeroLivre.Text.Trim();
    int idSuivi = lesSuivis[0].Id;
    string libelleSuivi = lesSuivis[0].Libelle;

    String montantSaisie = txbCommandeLivresMontant.Text.Replace(',', '.');
    Double montant;
    bool success = Double.TryParse(montantSaisie, out montant);
    if (!success)
    {
        MessageBox.Show("La valeur saisie pour le montant doit être numérique.", "Erreur");
        txbCommandeLivresMontant.Text = "";
        txbCommandeLivresMontant.Focus();
        return;
    }

    CommandeDocument laCommandeDocument = new CommandeDocument(id, dateCommande, montant, nbExemplaires,
        idLivreDvd, idSuivi, libelleSuivi);

    String message = controle.CreerCommandeDocument(laCommandeDocument);
    if (message.Substring(0, 7) == "Validé!")
    {
        MessageBox.Show(message, "Information");
    }
    else if (message.Substring(0, 9) == "Duplicate")
    {
        MessageBox.Show("Ce numéro de commande existe déjà.", "Erreur");
        txbCommandeLivresNumeroCommande.Text = "";
        txbCommandeLivresNumeroCommande.Focus();
        return;
    }
    else
    {
        MessageBox.Show(message, "Erreur");
        return;
    }
    FinSaisieCommandeLivres();
    CommandeLivresRechercher();
}

```

Figure 70: Validation d'une commande

Ensuite j'ai créé la méthode événementielle (et les méthodes correspondantes du contrôleur et du Dao) qui gère la suppression d'une CommandeDocument (avec un fonctionnement à l'identique des autres méthodes de suppression déjà implémentées) ainsi que les trois méthodes événementielles correspondantes aux boutons qui changent l'état de suivi d'une commande.

Pour ces trois boutons j'ai pu utiliser les mêmes méthodes dans le contrôleur et le Dao.

```

public static bool ModifSuiviCommandeDocument(string idCommandeDocument, int idSuivi)
{
    try
    {
        List<string> requetes = new List<string>();
        requetes.Add("update suivicommandedoc set idsuivi=@idsuivi where idcommande=@idcommande");
        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            {"@idsuivi", idSuivi },
            {"@idcommande", idCommandeDocument },
        };
        BddMySQL curs = BddMySQL.GetInstance(connectionString);
        curs.ReqUpdate(requetes, parameters);
        curs.Close();
        return true;
    }
    catch
    {
        return false;
    }
}

```

Figure 71: Dao : Modification d'un état de suivi

J'ai aussi créé une méthode de demande de validation de modification de l'état de suivi, ce qui me semblait utile d'un point de vue utilisateur, comme les retours en arrière ne sont pas possible.

```

/// <summary>
/// Affichage d'un MessageBox pour demander validation de changement d'état de suivi
/// </summary>
/// <param name="libelleSuivi"></param>
/// <returns></returns>
1reference | 0 changes | 0 authors, 0 changes
private bool ValidationModifEtatSuivi(string libelleSuivi)
{
    return (MessageBox.Show("Confirmez-vous le passage de cette commande en l'état '" +
        libelleSuivi + "' ?", "Confirmation", MessageBoxButtons.YesNo) == DialogResult.Yes);
}

```

Figure 72: Demande de validation par l'utilisateur


```

/// <summary>
/// Modification d'état de suivi de la CommandeDocument : étape 3 "réglée"
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | Carl Fremault, 12 hours ago | 1 author, 1 change
private void btnCommandeLivresRegler_Click(object sender, EventArgs e)
{
    CommandeDocument commandeDocument = (CommandeDocument)bdgCommandesLivresListe.List
        [bdgCommandesLivresListe.Position];
    Suivi nouveauSuivi = lesSuivis.Find(suivi => suivi.Libelle == "Réglée");
    ModifEtatSuiviCommandeDocument(commandeDocument.Id, nouveauSuivi);
}

/// <summary>
/// Demande de modification de l'état de suivi au contrôleur après validation utilisateur
/// </summary>
/// <param name="idCommandeDocument"></param>
/// <param name="nouveauSuivi"></param>
3 references | 0 changes | 0 authors, 0 changes
private void ModifEtatSuiviCommandeDocument(string idCommandeDocument, Suivi nouveauSuivi)
{
    if (ValidationModifEtatSuivi(nouveauSuivi.Libelle))
    {
        if (controle.ModifSuiviCommandeDocument(idCommandeDocument, nouveauSuivi.Id))
        {
            AfficheCommandeDocumentLivres();
        }
        else
        {
            MessageBox.Show("Une erreur s'est produite.", "Erreur");
        }
    }
}

```

Figure 73: Vue : Modification d'un état de suivi

The screenshot displays a web application interface for managing orders. At the top, a table titled 'Commandes :' lists four orders with columns for DateCommande, Montant, Exemplaires, and Etat. The third row is highlighted in blue. Below the table, a 'Détails commande' section shows 'Numéro commande : 45' and 'Date de commande : 11/03/2022'. A 'Confirmation' dialog box is open in the center, asking 'Confirmez-vous le passage de cette commande en l'état 'Livrée' ?' with 'Yes' and 'No' buttons. At the bottom, a 'Gestion des commandes' section contains five buttons: 'Ajouter', 'Relancer', 'Confirmer Livraison' (highlighted), 'Régler', and 'Supprimer'. A small illustration of a person on a bridge is visible on the right side of the interface.

DateCommande	Montant	Exemplaires	Etat
11/03/2022	200,00 €	1	Réglée
11/03/2022	80,60 €	2	Réglée
11/03/2022	45,00 €	1	Relancée
11/03/2022	200,00 €	1	Réglée

Confirmation

Confirmez-vous le passage de cette commande en l'état 'Livrée' ?

Yes No

Gestion des commandes

Ajouter Relancer Confirmer Livraison Régler Supprimer

Figure 74: Demande de confirmation utilisateur

Avant d'implémenter toutes ces modifications dans les onglets des commandes DVD et abonnements Revues j'ai fait un commit & push.

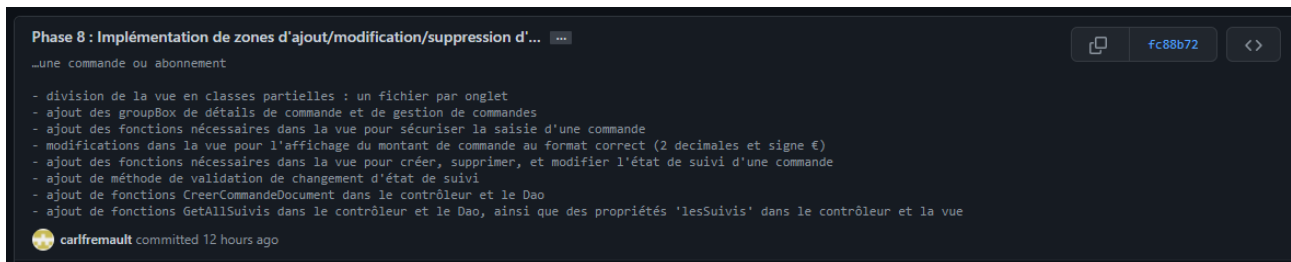


Figure 75: Commit pour la phase 8

En copiant et adaptant les différentes fonctions il était possible d'optimiser le code. Par exemple, la fonction qui trie les colonnes après un clic sur un 'column header' est identique pour les deux onglets. Il est donc possible de 'simplifier' la méthode événementielle et d'extraire les fonctionnalités dans une fonction commune.

```
/// <summary>
/// Tri sur une colonne
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | Carl Fremault, 13 hours ago | 1 author, 1 change
private void dgvCommandeDvdListe_ColumnHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)
{
    string titreColonne = dgvCommandeDvdListe.Columns[e.ColumnIndex].HeaderText;
    List<CommandeDocument> sortedList = SortCommandeDocumentList(titreColonne);
    RemplirCommandeDvdListe(sortedList);
}

/// <summary>
/// Tri sur une colonne pour les listes CommandeDocument
/// </summary>
/// <param name="titreColonne"></param>
/// <returns></returns>
2 references | 0 changes | 0 authors, 0 changes
private List<CommandeDocument> SortCommandeDocumentList(string titreColonne)
{
    List<CommandeDocument> sortedList = new List<CommandeDocument>();
    switch (titreColonne)
    {
        case "Date":
            sortedList = lesCommandeDocument.OrderBy(o => o.DateCommande).Reverse().ToList();
            break;
        case "Montant":
            sortedList = lesCommandeDocument.OrderBy(o => o.Montant).Reverse().ToList();
            break;
        case "Exemplaires":
            sortedList = lesCommandeDocument.OrderBy(o => o.NbExemplaires).Reverse().ToList();
            break;
        case "Etat":
            sortedList = lesCommandeDocument.OrderBy(o => o.IdSuivi).ToList();
            break;
    }
    return sortedList;
}
```

Figure 76: Optimisation de tri sur colonne avec une méthode commune

Ensuite, avant d'attaquer les travaux de l'onglet des abonnements, comme l'implémentation était un peu différente j'ai préféré faire un autre commit & push.

J'ai aussi décidé de modifier mon approche en ce qui est les messages des commit. Plutôt que d'incrémenter à chaque fois le numéro de phase, ce serait bien plus logique de garder le même numéro tant que je suis sur le(s) même(s) kanban. Il y aura donc une certaine inconsistance de numérotation entre le début et le reste du projet.

Bien entendu, à chaque fois que j'ai fait un push j'ai mis à jour le storyboard.

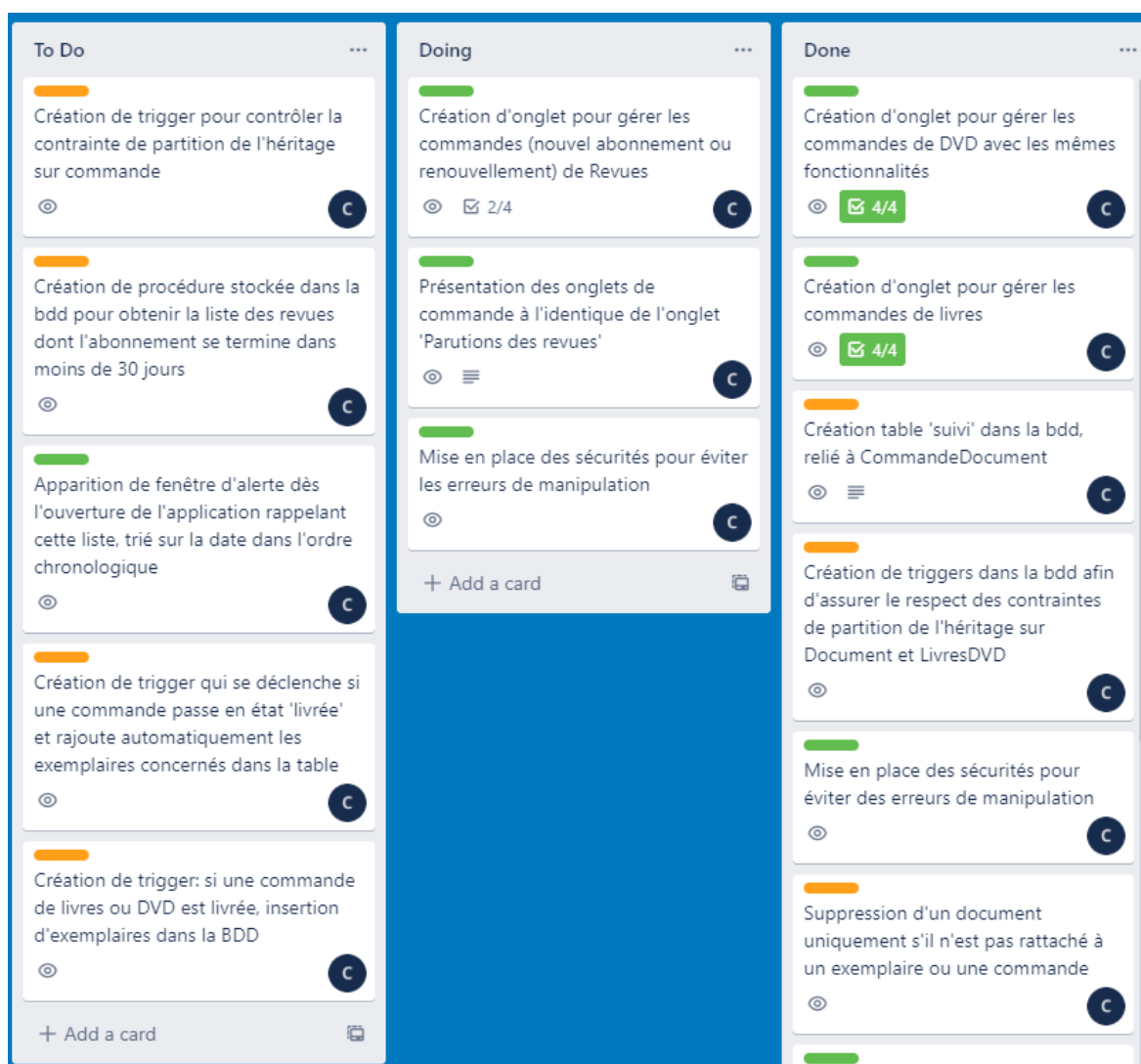


Figure 77: Mise à jour du Storyboard

Pour l'implémentation de souscription aux abonnements de revues l'interface est légèrement différent :

The image shows two parts of a Windows application interface. The top part, titled 'Détails abonnement', contains four text boxes: 'Numéro abonnement :', 'Montant :', 'Date de commande :', and 'Fin abonnement :'. The 'Date de commande' and 'Fin abonnement' boxes have a calendar icon and are currently set to '12/03/2022'. Below these are two buttons: 'Valider' and 'Annuler'. The bottom part, titled 'Gestion des abonnements', contains two buttons: 'Ajouter' and 'Supprimer'.

Figure 78: GroupBox de création d'abonnement

Aussi, il n'y avait pas d'état de suivi à gérer.

Quelques fonctions ont été implémentées différemment. Il semblait logique par exemple d'initialiser la date de fin d'abonnement à un an de la date actuelle :

```
private void VideDetailsAbonnementRevue()  
{  
    txbAbonnementRevueNumeroAbonnement.Text = "";  
    dtpAbonnementRevueDateCommande.Value = DateTime.Now;  
    dtpAbonnementRevueFinAbonnement.Value = DateTime.Now.AddYears(1);  
    txbAbonnementRevueMontant.Text = "";  
}
```

Figure 79: Mise à zéro des champs

Évidemment, lors d'un enregistrement d'un nouvel abonnement, il fallait vérifier si la date de fin d'abonnement n'est pas antérieure à la date d'abonnement !

```
if (DateTime.Compare(dtpAbonnementRevueDateCommande.Value,  
    dtpAbonnementRevueFinAbonnement.Value) >= 0)  
{  
    MessageBox.Show("La date de fin d'abonnement ne peut être antérieure ou égal  
        à la date de souscription à l'abonnement.", "Information");  
    return;  
}
```

Figure 80: Vérification des dates

Concernant la suppression d'un abonnement, le dossier documentaire précise qu'un abonnement ne peut être supprimé s'il y a des exemplaires rattachés (c.à.d : si la date de parution est comprise entre la date de souscription et la date de fin d'abonnement).

Pour ce faire il était demandé de créer une fonction 'ParutionDansAbonnement' qui prend trois paramètres: la date de commande, la date de fin d'abonnement, et la date de parution.

Pour que cette méthode puisse fonctionner cependant il faut également une autre méthode qui récupère ces informations respectivement de la vue (l'abonnement en question) et de la base de données (les exemplaires d'une revue dont on obtient l'identifiant depuis les propriétés de l'abonnement). Ces méthodes ont été insérés dans le contrôleur :

```
/// <summary>
/// Récupère les exemplaires rattachés à la revue concerné par un abonnement
/// puis demande vérification s'ils font partie de l'abonnement
/// </summary>
/// <param name="abonnement"></param>
/// <returns>True si un exemplaire est rattaché à l'abonnement</returns>
1 reference | 0 changes | 0 authors, 0 changes
public bool VerifSuppressionAbonnement(Abonnement abonnement)
{
    List<Exemplaire> lesExemplaires = GetExemplairesRevue(abonnement.IdRevue);
    bool parution = false;
    foreach(Exemplaire exemplaire in lesExemplaires)
    {
        if(ParutionDansAbonnement(abonnement.DateCommande,
            abonnement.DateFinAbonnement, exemplaire.DateAchat))
        {
            parution = true;
        }
    }
    return parution;
}

/// <summary>
/// Vérifie si la dateParution est comprise entre dateCommande et dateFinAbonnement
/// </summary>
/// <param name="dateCommande"></param>
/// <param name="dateFinAbonnement"></param>
/// <param name="dateParution"></param>
/// <returns>True si la date est comprise</returns>
1 reference | 0 changes | 0 authors, 0 changes
public bool ParutionDansAbonnement(DateTime dateCommande, DateTime dateFinAbonnement,
    DateTime dateParution)
{
    return (DateTime.Compare(dateCommande, dateParution) < 0 && DateTime.Compare
        (dateParution, dateFinAbonnement) < 0);
}
```

Figure 81: Vérification de parution dans un abonnement

Le dossier documentaire demandait également de créer un test unitaire pour cette dernière fonction 'ParutionDansAbonnement'. Il semble effectivement prudent de faire cela, avant de faire des tests sur la base de données.

```
[TestClass()]
References
public class ControleTests
{
    private readonly DateTime earlyDate = new DateTime(2022, 1, 1);
    private readonly DateTime middleDate = new DateTime(2022, 2, 1);
    private readonly DateTime lateDate = new DateTime(2022, 3, 1);

    private readonly Controle controleur = new Controle();

    [TestMethod()]
    References
    public void ParutionDansAbonnementTest()
    {
        // Date parution égale à date commande
        bool result1 = controleur.ParutionDansAbonnement(earlyDate, lateDate, earlyDate);
        Assert.AreEqual(false, result1, "Devrait réussir, dateparution égale à date commande donne false");
        // Date parution égale à date fin abonnement
        bool result2 = controleur.ParutionDansAbonnement(earlyDate, lateDate, lateDate);
        Assert.AreEqual(false, result2, "Devrait réussir, dateparution égale à date fin d'abonnement donne false");
        // Date parution avant date commande
        bool result3 = controleur.ParutionDansAbonnement(middleDate, lateDate, earlyDate);
        Assert.AreEqual(false, result3, "Devrait réussir, dateparution avant date commande donne false");
        // Date parution après date fin abonnement
        bool result4 = controleur.ParutionDansAbonnement(earlyDate, middleDate, lateDate);
        Assert.AreEqual(false, result4, "Devrait réussir, dateparution après date fin abonnement donne false");
        // Date parution comprise entre date abonnement et date fin abonnement
        bool result5 = controleur.ParutionDansAbonnement(earlyDate, lateDate, middleDate);
        Assert.AreEqual(true, result5, "Devrait réussir, dateparution comprise entre dates commande et fin abonnement donne true");
    }
}
```

Figure 82: Test unitaire sur la méthode 'ParutionDansAbonnement'

Test	Duration	Traits	Error Message
▲ Mediatek86Tests (1)	3 sec		
▲ Mediatek86.controleur.Tests (1)	3 sec		
▲ ControleTests (1)	3 sec		
ParutionDansAbonnementTest	3 sec		

Figure 83: Résultat du test

Les tests passent. Du coup j'ai pu essayer dans l'application :

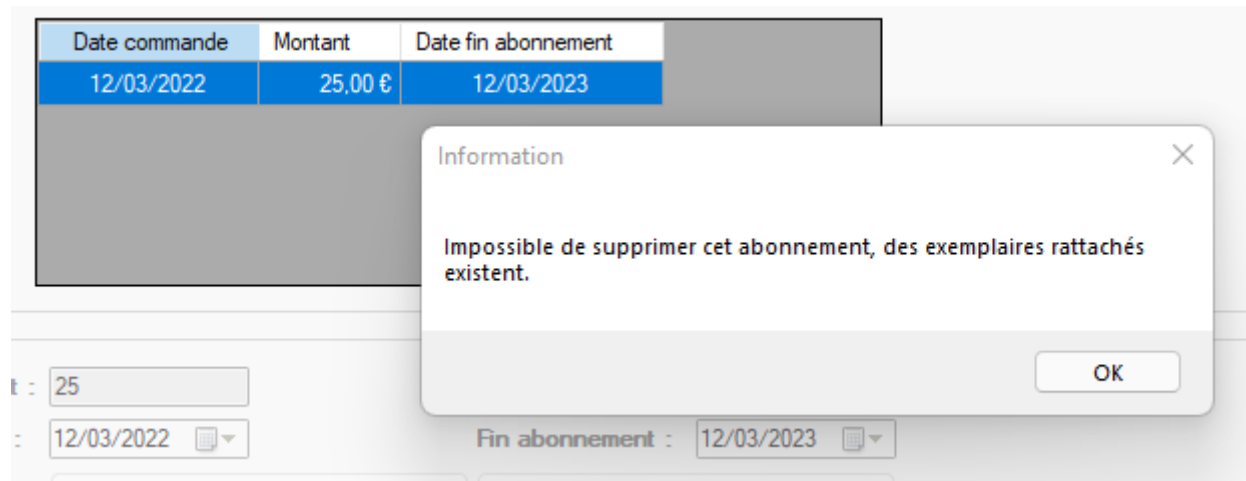


Figure 84: Interdiction de suppression d'un abonnement lorsque des exemplaires sont rattachés

En faisant des tests j'ai également trouvé quelques fonctionnements insatisfaisants, j'en ai profité pour les corriger avant de faire un commit et push.

7.4 Création de trigger qui ajoute les exemplaires lorsqu'une commande est livrée

Pour cette mission ils restaient quelques manipulations à faire au niveau de la base de données. Tout d'abord, quand une commande passe à l'étape "livrée", un trigger doit automatiquement ajouter les exemplaires concernées par cette commande. La 'dateAchat' de l'exemplaire doit être valorisé avec la date de commande, l'identifiant de l'état à '0001' (neuf), et le numéro d'exemplaire doit être séquentiel par rapport au livre concerné.

Le trigger inséré :

```

DROP TRIGGER IF EXISTS updateSuiviCommandeDoc;
DELIMITER //
CREATE TRIGGER updateSuiviCommandeDoc
AFTER UPDATE ON suivicommandedoc
FOR EACH ROW
BEGIN
    DECLARE idDoc VARCHAR(10);
    DECLARE nombre INTEGER;
    DECLARE date DATE;
    DECLARE numeroExemplaire INTEGER;
    /* trigger uniquement en cas de passage à l'étape 3 (livrée) */
    IF new.idSuivi = 3 THEN

        SELECT idLivreDvd, nbExemplaire, dateCommande INTO idDoc, nombre, date
        FROM suivicommandedoc scd
        JOIN commandedocument cd ON scd.idcommande = cd.id
        JOIN commande c ON cd.id = c.id
        WHERE scd.idcommande = old.idcommande;

        /* récupération du numéro d'exemplaire le plus élevé du document concerné */
        SELECT MAX(numero) INTO numeroExemplaire FROM exemplaire e WHERE e.id = idDoc;

        IF numeroExemplaire IS NULL THEN
            SET numeroExemplaire = 1;
        ELSE
            SET numeroExemplaire = numeroExemplaire + 1;
        END IF;

        /* ajout des exemplaires */
        REPEAT
            INSERT INTO exemplaire (id, numero, dateAchat, photo, idEtat) VALUES(idDoc, numeroExemplaire, date, "", "00001");
            SET numeroExemplaire = numeroExemplaire + 1;
            SET nombre = nombre - 1;
        UNTIL nombre = 0 END REPEAT;
    END IF;
END;
// DELIMITER ;

```

Figure 85: Trigger pour l'update de SuiviCommandeDoc

Il semblait bien aussi que l'utilisateur reçoit une confirmation de la création d'exemplaires. À cette fin j'ai légèrement modifié les méthodes concernées dans la vue. 'ModifEtatSuiviCommandeDocumentLivre' (et ...Dvd) retourne un booléen. La seule méthode qui l'exploite est la méthode événementielle lors d'un clic sur le bouton 'Confirmer livraison' qui affiche un message quand elle reçoit 'true' (le retour de 'false' étant géré dans la méthode 'ModifEtatSuiviCommandeDocument...').

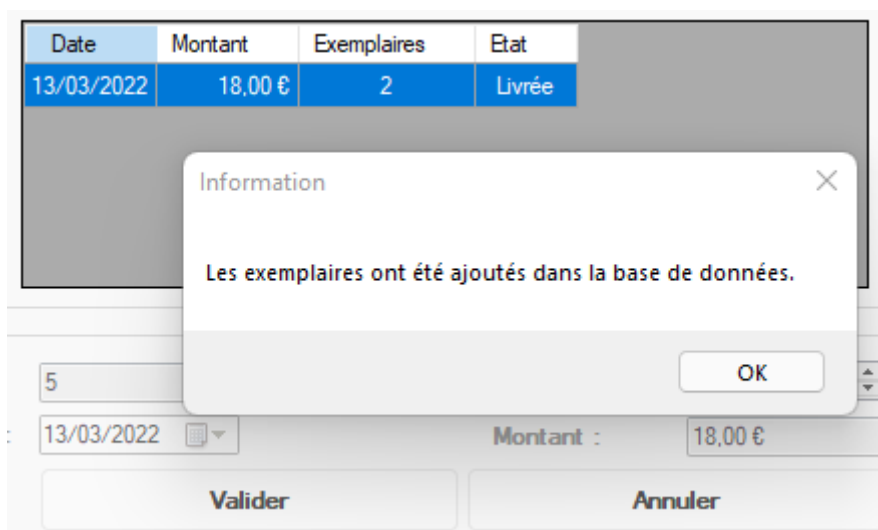


Figure 86: Confirmation d'ajout

7.5 Création de trigger pour contrôler la contrainte de partition de l'héritage sur commande

Ensuite il était demandé de créer un trigger pour éviter l'insertion d'une 'CommandeDocument' si un Abonnement existe avec le même numéro. Comme pour les trigger de contrôle de contrainte de partition d'héritage des tables 'document' et 'livres_dvd' il n'y a aucun risque de violation de la part de l'application. Effectivement, quand un Abonnement ou CommandeDocument est crée la base de données vérifie automatiquement l'unicité de la clé primaire. Toutefois, l'application pourrait évoluer, ou même être remplacée indépendamment de la base de données. Il convient de sécuriser au maximum la base de données contre toute mauvaise manipulation.

Voici les deux trigger insérés :

```
DROP TRIGGER IF EXISTS insAbonnement;
DELIMITER //
CREATE TRIGGER insAbonnement
  BEFORE INSERT ON abonnement
  FOR EACH ROW
BEGIN
  DECLARE countId INTEGER;
  SELECT COUNT(*) INTO countId FROM commandedocument WHERE id = new.id;
  IF countId > 0 THEN
    SIGNAL SQLSTATE "45000"
    SET MESSAGE_TEXT = "Une commande avec cet identifiant existe déjà.";
  END IF;
END;
// DELIMITER ;

DROP TRIGGER IF EXISTS insCommande;
DELIMITER //
CREATE TRIGGER insCommande
  BEFORE INSERT ON commandedocument
  FOR EACH ROW
BEGIN
  DECLARE countId INTEGER;
  SELECT COUNT(*) INTO countId FROM abonnement WHERE id = new.id;
  IF countId > 0 THEN
    SIGNAL SQLSTATE "45000"
    SET MESSAGE_TEXT = "Un abonnement avec cet identifiant existe déjà.";
  END IF;
END;
// DELIMITER ;
```

Figure 87: Triggers d'insertion d'abonnement et de commande

Il était impossible de tester ces triggers depuis l'application en raison de la sécurisation. Du coup j'ai fait des demandes d'insertion directement dans le SGBDR, avec un résultat satisfaisant :



Figure 88: Les triggers fonctionnent

7.6 Création de procédure stockée d'alerte de fin d'abonnements

Pour la prochaine tâche, il était demandé de créer une procédure stockée dans la base de données qui permet de récupérer les abonnement dont la date de fin est dans moins de 30 jours.

Voici la procédure créée :

```
DROP PROCEDURE IF EXISTS abonnementsFin30;
DELIMITER //
CREATE PROCEDURE abonnementsFin30()
BEGIN
SELECT a.dateFinAbonnement, a.idRevue, d.titre
    FROM abonnement a
    JOIN revue r ON a.idRevue = r.id
    JOIN document d ON r.id = d.id
    WHERE DATEDIFF(a.dateFinAbonnement, CURDATE()) < 30
    ORDER BY a.dateFinAbonnement ASC;
END;
// DELIMITER ;
```

Figure 89: Procédure pour récupérer les abonnements qui expirent dans moins de 30 jours

7.7 Création de fenêtre d'alerte de fin d'abonnements au démarrage de l'application

Maintenant que la procédure a été créée il ne restait plus qu'à générer une fenêtre d'alerte qui s'ouvre automatiquement dès l'ouverture de l'application, et qui affiche les abonnements concernés.

D'abord j'ai créé une nouvelle vue 'AlerteFinAbonnements' :

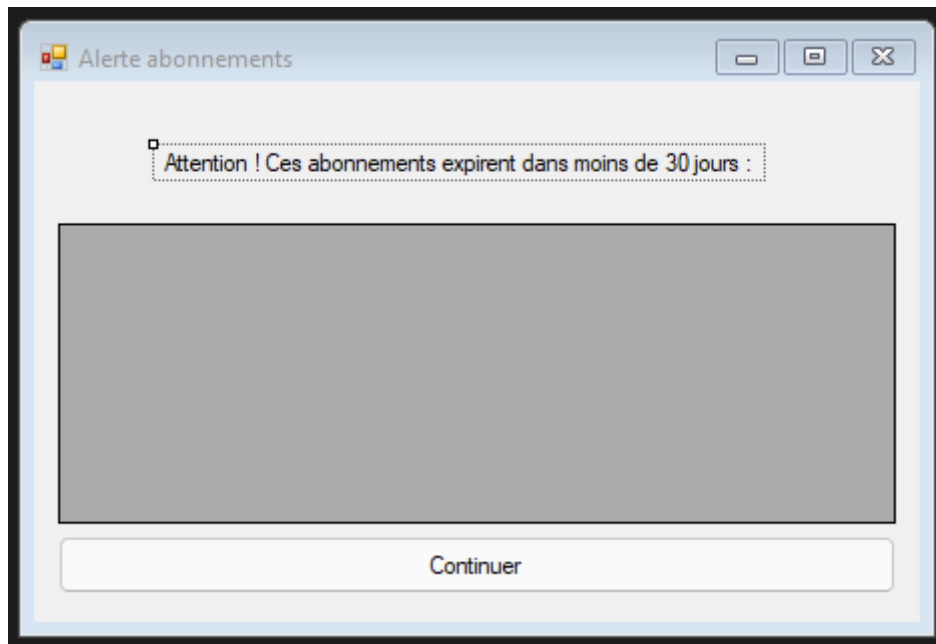


Figure 90: Fenêtre d'alerte de fin d'abonnements

Cette fenêtre devait s'ouvrir dès l'ouverture de l'application. À cet effet j'ai ajouté la méthode événementielle suivante dans la vue principale :

```
/// <summary>
/// Dès l'ouverture de l'application la vue d'alerte de fin d'abonnements s'ouvre
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1reference | 0 changes | 0 authors, 0 changes
private void FrmMediatek_Shown(object sender, EventArgs e)
{
    AlerteFinAbonnements alerteFinAbonnements = new AlerteFinAbonnements(controle);
    alerteFinAbonnements.StartPosition = FormStartPosition.CenterParent;
    alerteFinAbonnements.ShowDialog();
}
```

Figure 91: Méthode événementielle qui se déclenche lors de l'ouverture de la vue principale

Afin de pouvoir facilement remplir un nouveau 'DataGridView' j'ai créé une nouvelle classe métier 'FinAbonnement' qui valorise les valeurs retournés par la procédure créée auparavant : date de fin d'abonnement, identifiant et titre de la revue concernée.

```
namespace Mediatek86.metier
{
    1 reference | 0 changes | 0 authors, 0 changes
    class FinAbonnement
    {
        private readonly DateTime dateFinAbonnement;
        private readonly string idRevue;
        private readonly string titreRevue;

        0 references | 0 changes | 0 authors, 0 changes
        public FinAbonnement(DateTime dateFinAbonnement, string idRevue, string titreRevue)
        {
            this.dateFinAbonnement = dateFinAbonnement;
            this.idRevue = idRevue;
            this.titreRevue = titreRevue;
        }

        0 references | 0 changes | 0 authors, 0 changes
        public DateTime DateFinAbonnement => dateFinAbonnement;
        0 references | 0 changes | 0 authors, 0 changes
        public string IdRevue => idRevue;
        0 references | 0 changes | 0 authors, 0 changes
        public string TitreRevue => titreRevue;
    }
}
```

Figure 92: La classe métier 'FinAbonnement'

Ensuite il fallait gérer la récupération de la liste des abonnements concernés depuis la base de données. J'ai donc ajouté une méthode 'GetFinAbonnement' dans le contrôleur et le Dao :

```
1 reference | 0 changes | 0 authors, 0 changes
public static List<FinAbonnement> GetFinAbonnement()
{
    List<FinAbonnement> lesFinAbonnement = new List<FinAbonnement>();
    string req = "call abonnementsFin30()";

    BddMySQL curs = BddMySQL.GetInstance(connectionString);
    curs.Select(req, null);

    while (curs.Read())
    {
        DateTime dateFinAbonnement = (DateTime)curs.Field("dateFinAbonnement");
        string idRevue = (string)curs.Field("idRevue");
        string titreRevue = (string)curs.Field("titre");

        FinAbonnement finAbonnement = new FinAbonnement(dateFinAbonnement, idRevue,
            titreRevue);
        lesFinAbonnement.Add(finAbonnement);
    }
    curs.Close();

    return lesFinAbonnement;
}
```

Figure 93: Requête qui appelle la procédure créée auparavant

Puis il ne restait plus qu'à gérer l'affichage de la liste obtenue dans le DataGridView de la nouvelle fenêtre d'alerte :

```
4 references | Carl Fremault, 1 hour ago | 1 author, 2 changes
public partial class AlerteFinAbonnements : Form
{
    private readonly BindingSource bdgAlerteAbonnements = new BindingSource();
    private List<FinAbonnement> lesFinAbonnement;

    /// <summary>
    /// Constructeur. Remplit le tableau des abonnements
    /// </summary>
    /// <param name="controle"></param>
    1 reference | Carl Fremault, 2 hours ago | 1 author, 1 change
    internal AlerteFinAbonnements(Controle controle)
    {
        InitializeComponent();
        lesFinAbonnement = controle.GetFinAbonnement();
        bdgAlerteAbonnements.DataSource = lesFinAbonnement;
        dgvAlerteFinAbonnements.DataSource = bdgAlerteAbonnements;
        dgvAlerteFinAbonnements.AutoSizeColumnsMode =
            DataGridViewAutoSizeColumnsMode.AllCells;
        dgvAlerteFinAbonnements.Columns["idRevue"].DisplayIndex = 2;
        dgvAlerteFinAbonnements.Columns[0].HeaderCell.Value = "Date fin d'abonnement";
        dgvAlerteFinAbonnements.Columns[1].HeaderCell.Value = "Identifiant Revue";
        dgvAlerteFinAbonnements.Columns[2].HeaderCell.Value = "Revue";
    }

    /// <summary>
    /// Désactivation de sélection d'une ligne dans le tableau
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    1 reference | Carl Fremault, 2 hours ago | 1 author, 1 change
    private void dgvAlerteFinAbonnements_SelectionChanged(object sender, EventArgs e)
    {
        dgvAlerteFinAbonnements.ClearSelection();
    }

    /// <summary>
    /// Événement clic sur le bouton 'continuer', ferme la fenêtre
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    1 reference | Carl Fremault, 2 hours ago | 1 author, 1 change
    private void btnAlerteFinAbonnements_Click(object sender, EventArgs e)
    {
        this.Close();
    }
}
```

Figure 94: La vue d'alerte de fin d'abonnements

Voici le résultat :

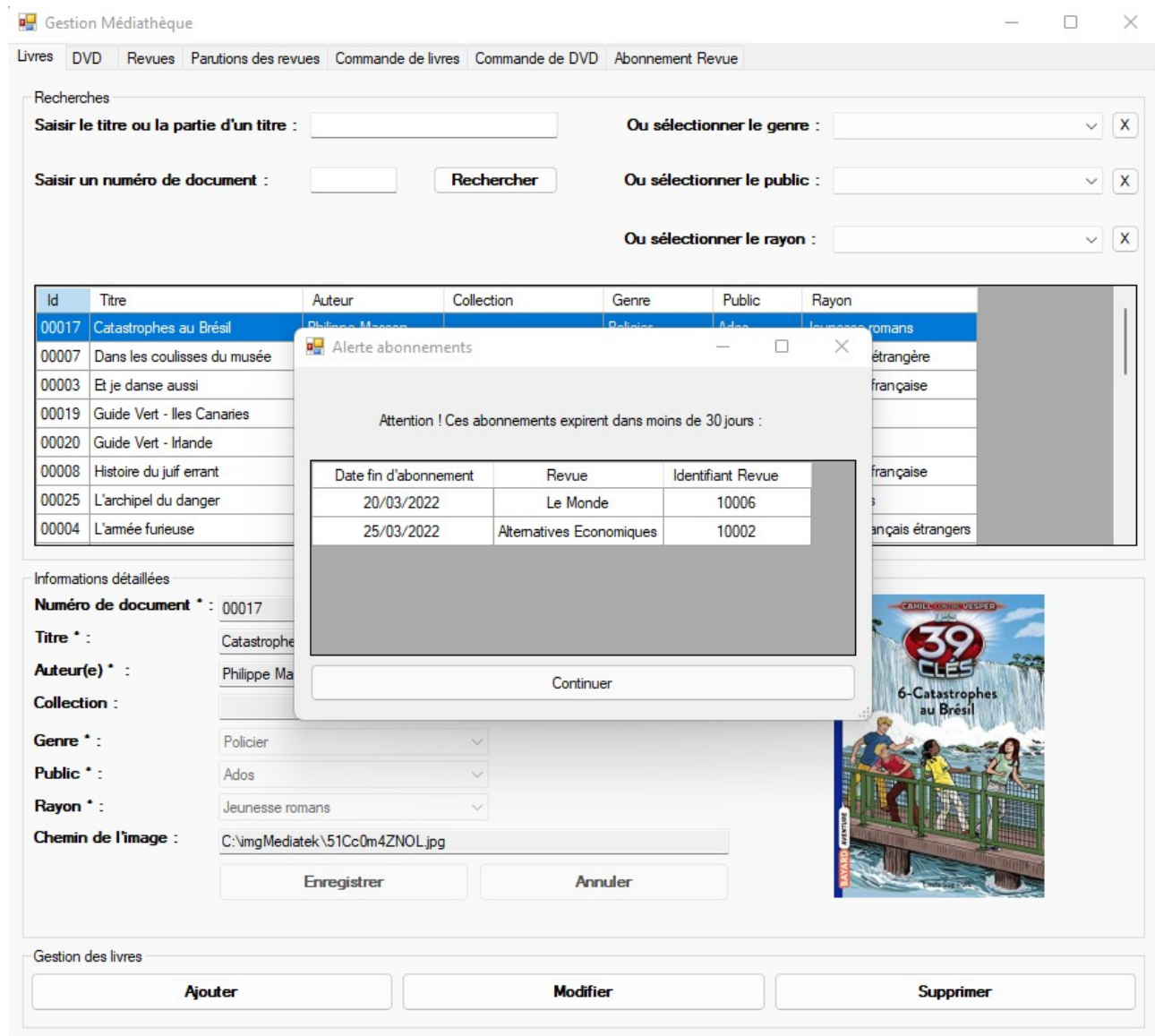


Figure 95: L'alerte s'affiche dès le démarrage de l'application

Toutes les demandes de cette mission étaient été traités maintenant. Il était temps de faire un commit, push et merge avec la branche principale, puis de faire une analyse avec SonarQube.

```
Phase 10 : Création de triggers et procédures dans la base de données ...
Suite à la création du trigger de mise à jour d'état de commande :
- modification des méthodes 'ModifEtatSuiviCommandeDocumentDvd' et '...Livrer' pour retourner un booléen, permettant affichage confirmation
d'insertion d'exemplaires après passage en état 'livrée' d'une commande

Phase 11 : Fenêtre d'alerte de fin d'abonnements qui s'affiche au démarrage
- création d'une vue 'AlerteFinAbonnements'
- ajout de classe métier 'FinAbonnement'
- ajout de méthodes GetFinAbonnement dans le Dao et le contrôleur

carlfremault committed 11 seconds ago
```

Figure 96: Commit de la phase 10

Compte tenu du petit nombre de changements de la phase précédente (Création de trigger dans la base de données, qui avait entraîné un petit changement pour afficher une MessageBox de confirmation à l'utilisateur) j'avais choisi de ne pas faire de commit tout de suite.

Toutefois, compte tenu de l'affichage des commit dans GitHub, cela semble avoir été une mauvaise décision. J'en tiendrai compte dans le futur...

SonarQube se plaint des lignes de code dupliquées et apparemment deux 'code smells' ont apparu :

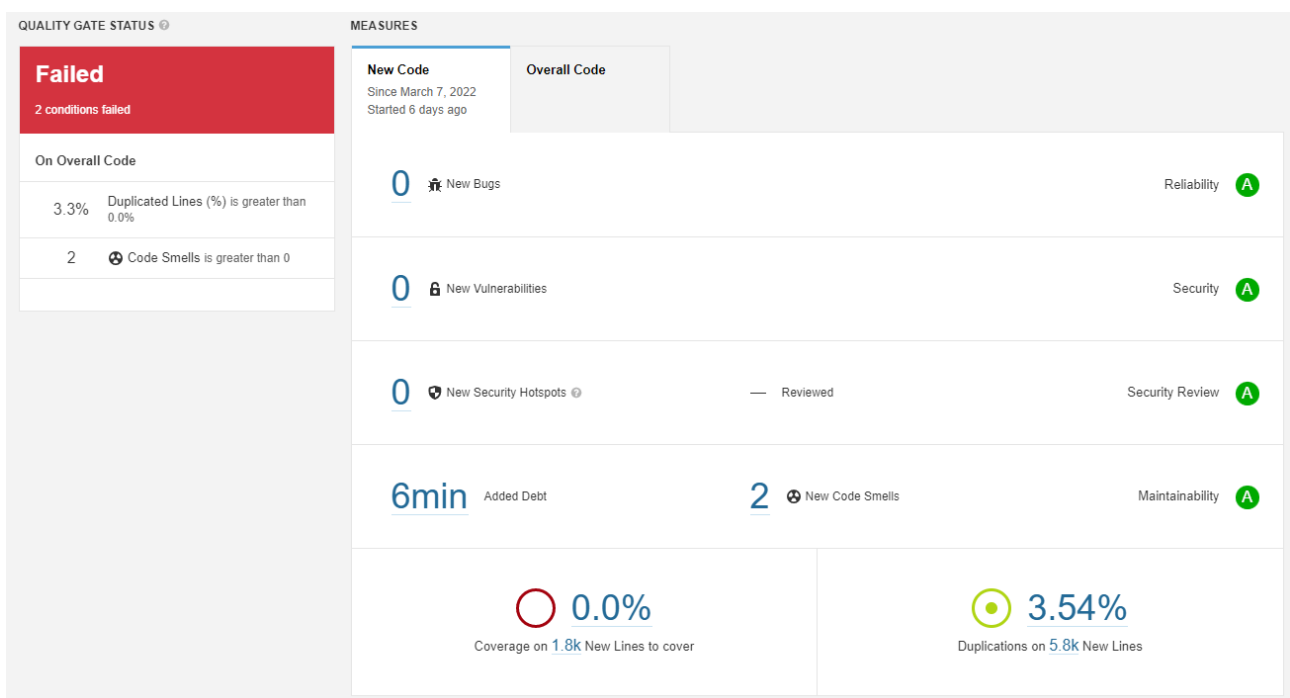


Figure 97: Analyse SonarQube

Un premier 'code smell' est vite résolu :



Figure 98: Premier 'code smell'

Pour ce qui est le deuxième 'code smell', je ne connaissais pas l'utilisation de Linq, toutefois les explications données par SonarQube sont très claires.



Figure 99: Deuxième 'code smell'

Ainsi j'ai pu améliorer la solution :

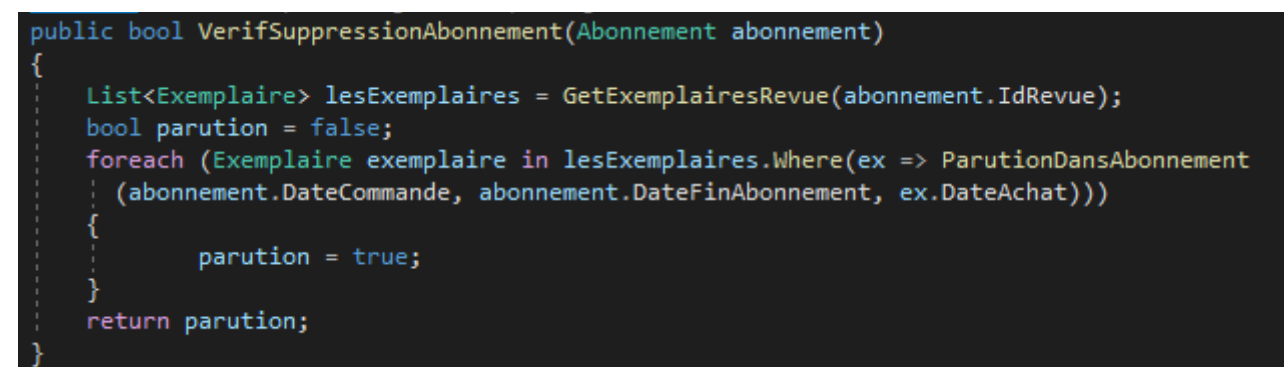


Figure 100: Optimisation du code suite aux remarques SonarQube

Au niveau des lignes de code dupliquée, évidemment il y a toujours le fichier du Dao qui en contient beaucoup. Comme évoqué précédemment j'ai préféré ne pas faire de changements.

SonarQube a trouvé des duplications aussi au niveau des méthodes événementielles lors d'un clic sur le bouton 'Enregistrer', et ce pour les Livres, DVD et Revues.

```
private void btnEnregistrerDvd_Click(object sender, EventArgs e)
{
    if (cbxInfosDvdGenres.SelectedIndex == -1 || cbxInfosDvdPublics.SelectedIndex == -1 || cbxInfosDvdRayons.SelectedIndex == -1 || txbDvdNumero.Text == "" || txbDvdRealisateur.Text == "" || txbDvdTitre.Text == "" || txbDvdDuree.Text == "")
    {
        MessageBox.Show("Les champs marqués d'un astérisque (*) sont obligatoires.", "Information");
        return;
    }

    Genre leGenre = (Genre)bdgInfosGenres.List[bdgInfosGenres.Position];
    String idGenre = leGenre.Id;
    String genre = leGenre.ToString();
    Public lePublic = (Public)bdgInfosPublics.List[bdgInfosPublics.Position];
    String idPublic = lePublic.Id;
    String unPublic = lePublic.ToString();
    Rayon leRayon = (Rayon)bdgInfosRayons.List[bdgInfosRayons.Position];
    String idRayon = leRayon.Id;
    String rayon = leRayon.ToString();
    String id = txbDvdNumero.Text.Trim();
    String titre = txbDvdTitre.Text;
    String realisateur = txbDvdRealisateur.Text;
    String synopsis = txbDvdSynopsis.Text;
    String image = txbDvdImage.Text;
```

Figure 101: Code répété

Effectivement, même s'il s'agit de trois fichiers séparés, toutefois ils représentent à chaque fois une 'partial class', de la vue FrmMediaTek.

Afin de préserver la lisibilité du code cependant (et surtout comme je l'ai séparé en plusieurs fichiers) j'ai préféré là aussi laisser tel quel.

7.8 Bilan pour la mission 2

La mission était volumineuse mais sans réelles difficultés.

J'ai hésité toutefois au niveau de ma solution pour l'incorporation des états de suivi de commandes. Le fait d'appeler directement des états de suivi à l'aide d'un index sur la liste, par exemple lors de la création d'une commande, s'approche du codage 'en dur' et pourra compliquer l'évolution de l'application.

```
int idSuivi = lesSuivis[0].Id;
string libelleSuivi = lesSuivis[0].Libelle;
```

Figure 102: Récupération d'identifiant et libellé

D'un côté il aurait été plus simple d'utiliser un 'comboBox' comme pour les Genres, Rayons et Publics. Toutefois, le fait d'utiliser les différents boutons qui se (dés)activent en fonction des manipulations possibles apporte une sécurisation en plus.

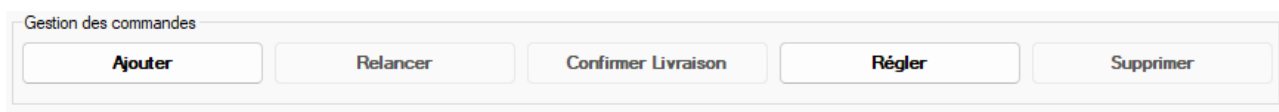


Figure 103: Verrouillage des boutons d'évolution d'état de suivi

Aussi, il semble logique que les différentes étapes soient identifiées par des numéros consécutifs, et par ailleurs, lors de la création des méthodes événementielles, il est évident de choisir le texte sur les boutons en fonction des libellés des états.

```
private void btnCommandeDvdRelancer_Click(object sender, EventArgs e)
{
    CommandeDocument commandeDocument = (CommandeDocument)bdgCommandesDvdListe.List[bdgCommandesDvdListe.Position];
    Suivi nouveauSuivi = lesSuivis.Find(suivi => suivi.Libelle == "Relancée");
    ModifEtatSuiviCommandeDocumentDvd(commandeDocument.Id, nouveauSuivi);
}
```

Figure 104: Exemple de méthode événementielle de changement d'état de suivi

Une autre constatation concerne la vue principale FrmMediaTek. Le fichier devenait très long, ainsi je l'ai découpé en 'classes partielles' ce qui facilite énormément les manipulations.

Beaucoup de fonctions sont très similaires, avec quelques différences au niveau des paramètres ou champs utilisés. Je me suis souvent demandé s'il ne serait pas mieux d'extraire une logique qui existe, par exemple, pour les livres et les DVD, dans une fonction séparée. À mon avis il y a des pour et des contre. D'un côté il y aurait moins de méthodes, mais elles seraient plus compliquées à lire, avec beaucoup de conditions 'if' pour décliner les différents cas d'usage. Au fin de compte je pense qu'il n'y a pas forcément une seule réponse à ce genre de questions. Il me faudra plus d'expérience en tant que développeur pour mieux pouvoir juger.

Finalement je ne suis pas très satisfait de ma 'gestion' des commit. J'avais pris une habitude d'incrémenter le numéro de 'phase' à chaque commit, inspiré d'un de nos cours. Dans ce cours cependant il s'agissait à chaque fois d'étapes bien séparées, ce qui n'était pas forcément le cas pendant cet atelier.

Je me suis repris en milieu de cette mission et j'ai décidé de bien délinéer les phases, puis chaque phase peut très bien comprendre plusieurs commit (avec bien évidemment des explications pour chaque commit).

Le résultat toutefois est qu'il y aura une inconsistance au niveau des messages des commit pendant ce projet.

8. Mission 4 : Mettre en place des authentifications

Les tâches de cette quatrième mission sont les suivantes :

- Ajouter une table Utilisateur et une table Service dans la base de données
 - Chaque utilisateur ne fait partie que d'un service
 - Remplir les tables d'exemples pour les tests
- Ajouter une fenêtre d'authentification qui s'ouvre en premier
- Empêcher l'accès à certaines fonctionnalités suivant le service dont dépend l'utilisateur
 - Rendre invisibles ou inactifs certains onglets ou objets graphiques
- Les personnes du service Culture n'ont accès à rien. Afficher un message pour leur en informer puis fermer l'application
- L'alerte de fin d'abonnements ne doit uniquement apparaître pour les personnes qui gèrent les commandes

J'ai d'abord mis à jour le storyboard dans Trello avec ces tâches :

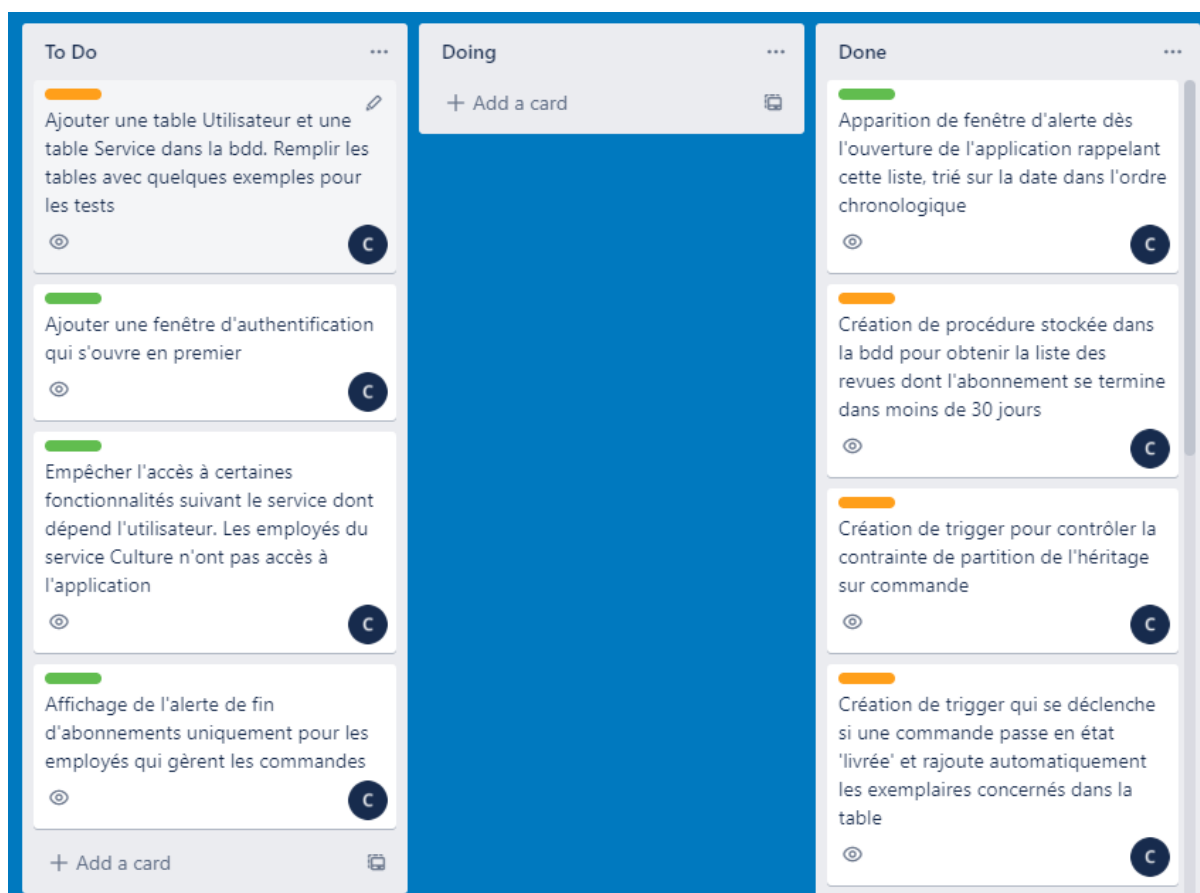


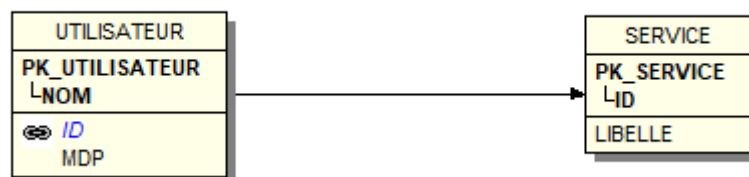
Figure 105: Storyboard pour la mission 4

8.1 Création des tables Utilisateur et Service dans la base de données

Pour la création des tables j'ai choisi de faire une modélisation dans WinDesign puis de générer le script de création des tables depuis cette application. Ces deux table sont liées entre eux, mais il n'y a pas de liaison avec le reste de la base de données. Comme spécifié dans le dossier de mission, un utilisateur ne fait partie que d'un service.



À partir du modèle conceptuel j'ai généré le modèle logique :



Bien entendu, avant d'exécuter le script j'ai enlevé les premières lignes de création de base de données.

```

DROP DATABASE IF EXISTS MLR2;

CREATE DATABASE IF NOT EXISTS MLR2;
USE MLR2;
#-----
#    TABLE : UTILISATEUR
#-----

CREATE TABLE IF NOT EXISTS UTILISATEUR
(
    NOM VARCHAR(30) NOT NULL ,
    ID INTEGER(2) NOT NULL ,
    MDP VARCHAR(30) NULL
    , PRIMARY KEY (NOM)
)
comment = '';

#-----
#    INDEX DE LA TABLE UTILISATEUR
#-----

CREATE INDEX I_FK_UTILISATEUR_SERVICE
ON UTILISATEUR (ID ASC);

#-----
#    TABLE : SERVICE
#-----

CREATE TABLE IF NOT EXISTS SERVICE
(
    ID INTEGER(2) NOT NULL AUTO_INCREMENT ,
    LIBELLE VARCHAR(30) NULL
    , PRIMARY KEY (ID)
)
comment = '';

#-----
#    CREATION DES REFERENCES DE TABLE
#-----

ALTER TABLE UTILISATEUR
ADD FOREIGN KEY FK_UTILISATEUR_SERVICE (ID)
REFERENCES SERVICE (ID) ;

```

Figure 106: Script de création des tables

Ensuite j'ai inséré les différents services :

```

✓ 4 rows inserted.
Inserted row id: 9 (Query took 0.0009 seconds.)

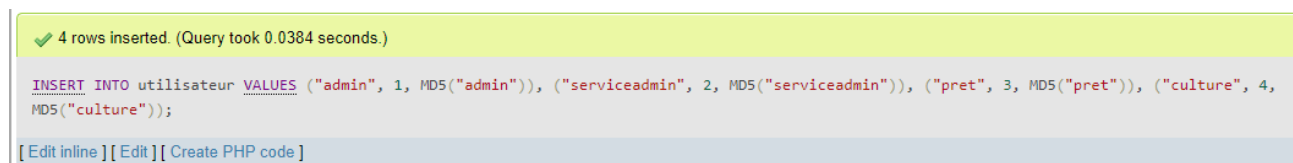
INSERT INTO SERVICE (libelle) VALUES ("admin"), ("services administratifs"), ("prêt"), ("culture");
[ Edit inline ] [ Edit ] [ Create PHP code ]

```

Figure 107: Insertion des services dans le SGBDR

Le dossier spécifie que les employés du Service administratif ont accès à toutes les fonctionnalités de l'application, et l'Administrateur à toutes les fonctionnalités de toutes les applications. Pour cet atelier il n'y avait donc pas vraiment de raison pratique pour faire un compte Administrateur séparé. Toutefois on peut s'imaginer que des évolutions futures puissent changer ce besoin. Quoi qu'il en soit, il semble prudent de prévoir un compte Administrateur à part.

Pour l'insertion des utilisateurs, afin de sécuriser au maximum il convient de stocker les hash des mots de passe. Pour des raisons de démonstration je n'ai pas cherché la complexité au niveau des mots de passe cependant.



```
✓ 4 rows inserted. (Query took 0.0384 seconds.)

INSERT INTO utilisateur VALUES ("admin", 1, MD5("admin")), ("serviceadmin", 2, MD5("serviceadmin")), ("pret", 3, MD5("pret")), ("culture", 4, MD5("culture"));

[ Edit inline ] [ Edit ] [ Create PHP code ]
```

Figure 108: Insertion des utilisateurs et mots de passe

Il s'avérait que la longueur de 30 caractères pour le hash des mots de passe n'était pas suffisant. Je l'ai modifié pour 50 caractères, directement dans le SGBDR.

8.2 Ajout de fenêtre d'authentification

Maintenant il fallait créer une nouvelle vue dans l'application, pour permettre aux utilisateurs de saisir leur nom d'utilisateur et mot de passe. Évidemment cette fenêtre doit s'ouvrir immédiatement lors de l'ouverture de l'application. L'application même ne doit s'ouvrir que si l'authentification a réussi.

J'ai fait une première implémentation sans appels à la base de données, afin de vérifier le fonctionnement d'enchaînement des fenêtres :

```

public partial class Authentification : Form
{
    private readonly Controle controle;

    /// <summary>
    /// Booléen, true si l'authentification a réussi
    /// </summary>
    2 references | 0 changes | 0 authors, 0 changes
    public bool authentificationSucces { get; private set; }

    1 reference | 0 changes | 0 authors, 0 changes
    internal Authentification(Controle controle)
    {
        InitializeComponent();
        this.controle = controle;
    }

    /// <summary>
    /// Clic sur le bouton valider vérifie si l'utilisateur et mdp sont correctes avant
    /// d'actionner le booléen authentification succes et de fermer la fenêtre
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    1 reference | 0 changes | 0 authors, 0 changes
    private void btnAuthValider_Click(object sender, EventArgs e)
    {
        authentificationSucces = true;
        Close();
    }

    /// <summary>
    /// Clic sur le bouton annuler ferme l'application
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    1 reference | 0 changes | 0 authors, 0 changes
    private void btnAuthAnnuler_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }
}

```

Figure 109: La nouvelle vue d'authentification

Il fallait modifier le constructeur du contrôleur aussi pour prendre en compte cette nouvelle fenêtre :

```

/// <summary>
/// Ouverture de la fenêtre d'authentification
/// Si l'authentification a réussi, ouverture de l'application
/// </summary>
2 references | Carl Fremault, 22 hours ago | 2 authors, 3 changes
public Controle()
{
    lesLivres = Dao.GetAllLivres();
    lesDvd = Dao.GetAllDvd();
    lesRevues = Dao.GetAllRevues();
    lesGenres = Dao.GetAllGenres();
    lesRayons = Dao.GetAllRayons();
    lesPublics = Dao.GetAllPublics();
    lesSuivis = Dao.GetAllSuivis();
    Authentification authentification = new Authentification(this);
    Application.Run(authentification);
    if (authentification.authentificationSucces)
    {
        FrmMediatek frmMediatek = new FrmMediatek(this);
        Application.Run(frmMediatek);
    }
}

```

Figure 110: Exploitation de la vue d'authentification dans le contrôleur

Après un petit test satisfaisant il était temps de créer les méthodes nécessaires pour appeler la base de données dans le contrôleur et le Dao. Étant donné le cas d'utilisation, il m'a semblé peu utile de mémoriser l'utilisateur après authentification. Ce dont on aura besoin est le rôle de l'utilisateur dans l'organisation (i.e. son service), afin de décider quelles fonctionnalités seront disponibles.

J'ai donc ajouté une nouvelle classe métier 'Service'.


```

namespace Mediatek86.metier
{
    1 reference | 0 changes | 0 authors, 0 changes
    public class Service
    {
        private readonly int id;
        private readonly string libelle;

        0 references | 0 changes | 0 authors, 0 changes
        public Service(int id, string libelle)
        {
            this.id = id;
            this.libelle = libelle;
        }

        0 references | 0 changes | 0 authors, 0 changes
        public int Id => id;

        0 references | 0 changes | 0 authors, 0 changes
        public string Libelle => libelle;
    }
}

```

Figure 111: La classe métier 'Service'

Voici la méthode créée dans le Dao :

```

/// <summary>
/// Retourne le service d'un utilisateur
/// </summary>
/// <param name="utilisateur">Le nom d'utilisateur concerné</param>
/// <param name="mdp">Le mot de passe de l'utilisateur</param>
/// <returns>Le service si l'utilisateur est trouvé dans la bdd, sinon null</returns>
0 references | 0 changes | 0 authors, 0 changes
public static Service Authentification(string utilisateur, string mdp)
{
    Service service = null;
    string req = "select s.ID, s.LIBELLE from utilisateur u join service s on u.ID = s.ID where u.NOM = @utilisateur and u.MDP = @mdp";
    Dictionary<string, object> parameters = new Dictionary<string, object>
    {
        { "@utilisateur", utilisateur },
        { "@mdp", mdp }
    };
    BddMySQL curs = BddMySQL.GetInstance(connectionString);
    curs.RegSelect(req, parameters);

    while (curs.Read())
    {
        service = new Service((int)curs.Field("ID"), (string)curs.Field("LIBELLE"));
    }
    curs.Close();
    return service;
}

```

Figure 112: Méthode 'Authentification' du Dao

Le contrôleur doit passer la demande de la vue. Toutefois, le mot de passe saisi doit être 'hashé' d'abord. Quelques recherches sur StackOverflow et les docs MSDN m'ont donné la marche à suivre :

```
O references | 0 changes | 0 authors, 0 changes
public string CreateMD5(string mdp)
{
    // Utilisation du string mdp pour calculer MD5 hash
    using (MD5 md5 = MD5.Create())
    {
        byte[] inputBytes = Encoding.ASCII.GetBytes(mdp);
        byte[] hashBytes = md5.ComputeHash(inputBytes);

        // Conversion du vecteur vers un string hexadecimal
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < hashBytes.Length; i++)
        {
            sb.Append(hashBytes[i].ToString("X2"));
        }
        return sb.ToString();
    }
}
```

Figure 113: Création d'un hash depuis une chaîne de caractères

La méthode appelée par la vue et qui va appeler la méthode du Dao pour récupérer le service de l'utilisateur qui essaye de se connecter va exploiter cette méthode :

```
/// <summary>
/// Récupère le service de l'utilisateur qui essaye de se connecter depuis la bdd
/// Valorise la propriété 'service'
/// </summary>
/// <param name="utilisateur">L'identifiant de l'utilisateur</param>
/// <param name="mdp">Le mot de passe de l'utilisateur</param>
/// <returns>Le service de l'utilisateur s'il est trouvé dans la bdd, et le mdp est
/// correct. Sinon retourne null</returns>
1 reference | 0 changes | 0 authors, 0 changes
public Service Authentification(string utilisateur, string mdp)
{
    Service service = Dao.Authentification(utilisateur, CreateMD5(mdp));
    leService = service;
    return service;
}
```

Figure 114: Appel de la méthode d'authentification par le contrôleur

J'ai également créé une propriété 'leService' de type Service dans le contrôleur. Ainsi le service de l'utilisateur connecté est mémorisé et pourra servir pour (dé)bloquer les différentes fonctionnalités.

Les méthodes nécessaires étant en place j'ai pu implémenter les fonctionnalités demandées dans la méthode événementielle de la vue, déclenchée lors d'un clic sur le bouton 'Valider'. J'ai également créé une petite méthode 'ViderChamps' en cas de saisie incorrecte.

```
/// <summary>
/// Clic sur le bouton valider vérifie si l'utilisateur et mdp sont correctes avant d'actionner
/// le booléen authentication succes et de fermer la fenêtre
/// L'accès à l'application n'est pas autorisé pour le service 'Culture'
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
/// <reference | 0 changes | 0 authors, 0 changes
private void btnAuthValider_Click(object sender, EventArgs e)
{
    string utilisateur = txbAuthUtilisateur.Text.Trim();
    string mdp = txbAuthMdp.Text.Trim();
    Service leService = controle.Authentification(utilisateur, mdp);
    if (leService != null)
    {
        if (leService.Libelle == "culture")
        {
            MessageBox.Show("L'utilisation de cette application est réservée aux services administratifs et au service de prêt.", "Information");
            ViderChamps();
        }
        else
        {
            authenticationSucces = true;
            Close();
        }
    }
    else
    {
        MessageBox.Show("Nom d'utilisateur et/ou mot de passe incorrecte(s)", "Erreur");
        ViderChamps();
    }
}
```

Figure 115: Événement lors d'un clic sur le bouton 'Valider'

Les résultats sont satisfaisants :

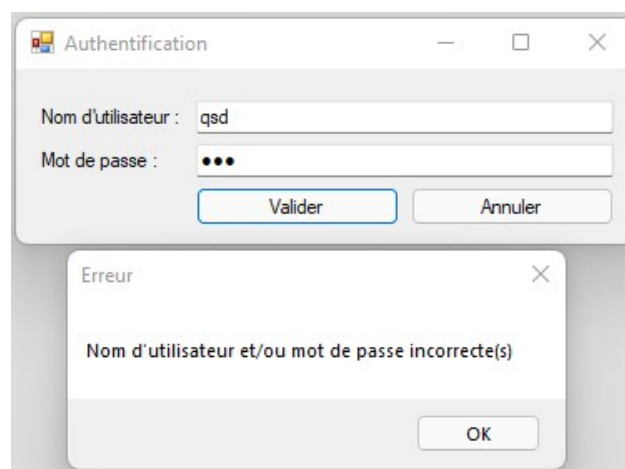


Figure 116: Saisie incorrecte

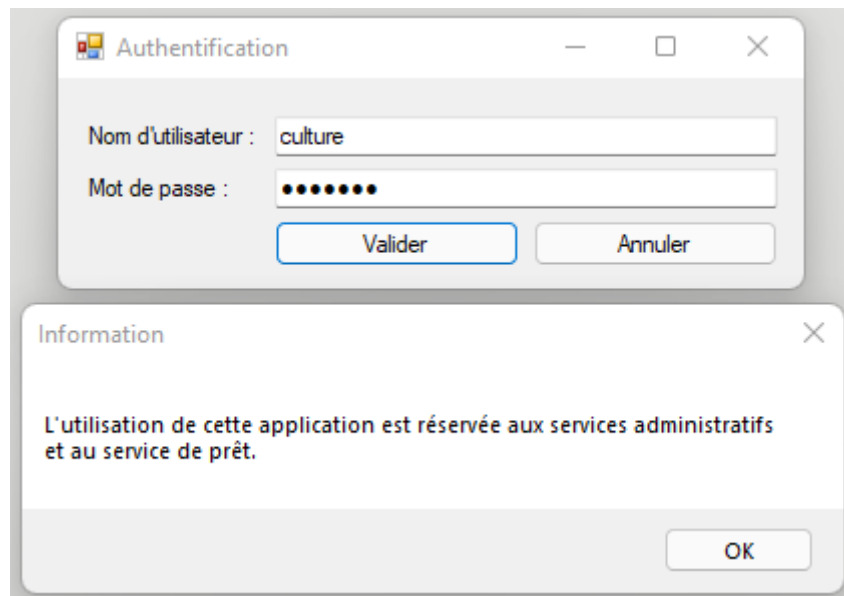


Figure 117: Les employés du service 'Culture' n'ont pas d'accès à l'application

8.3 Gérer les accès aux fonctionnalités suivant le service de l'utilisateur

Maintenant que l'authentification était fonctionnelle, j'ai procédé avec la gestion des accès aux différentes fonctionnalités, en rendant invisible les fonctionnalités 'verrouillées'.

Pour ce faire j'ai modifié le constructeur de la vue 'FrmMediatek' en y ajoutant une condition sur le libellé du Service mis en mémoire lors de l'authentification.

```

/// <summary>
/// Constructeur. Rend invisible certaines fonctionnalités en cas de connection
/// d'un utilisateur du service "prêt"
/// </summary>
/// <param name="controle"></param>
/// reference | 0 changes | 0 authors, 0 changes
internal FrmMediatek(Controle controle)
{
    InitializeComponent();
    this.controle = controle;
    if (controle.leService.Libelle == "prêt")
    {
        tabOngletsApplication.TabPages.Remove(tabCommandeLivres);
        tabOngletsApplication.TabPages.Remove(tabCommandeDVD);
        tabOngletsApplication.TabPages.Remove(tabAbonnementRevue);
        grpReceptionExemplaire.Visible = false;
        grpGestionLivres.Visible = false;
        grpGestionDVD.Visible = false;
        grpGestionRevue.Visible = false;
        btnEnregistrerLivre.Visible = false;
        btnEnregistrerDvd.Visible = false;
        btnEnregistrerRevue.Visible = false;
        btnAnnulerSaisieLivre.Visible = false;
        btnAnnulerSaisieDvd.Visible = false;
        btnAnnulerSaisieRevue.Visible = false;
    }
}

```

Figure 118: Le constructeur de la vue principale vérifie le service mémorisé

Pour finir, il fallait faire de sorte que l'alerte d'expiration des abonnements ne s'affiche pas pour les utilisateurs du service 'prêts'. J'ai procédé de la même façon :

```

private void FrmMediatek_Shown(object sender, EventArgs e)
{
    if (controle.leService.Libelle != "prêt")
    {
        AlerteFinAbonnements alerteFinAbonnements = new AlerteFinAbonnements
        (controle);
        alerteFinAbonnements.StartPosition = FormStartPosition.CenterParent;
        alerteFinAbonnements.ShowDialog();
    }
}

```

Figure 119: Vérification du service mémorisé avant d'afficher les alertes

Après quelques tests j'ai pu boucler cette mission avec un commit & push, suivi d'un merge avec la branche 'master' et une analyse SonarQube.

Cette fois un 'security hotspot' est signalé :

Make sure this weak hash algorithm is not used in a sensitive context here. [Add Comment](#) [Open in IDE](#) [Get Permalink](#)

Using weak hashing algorithms is security-sensitive [csharpsquid:S4790](#)

Category: Others

Review priority: **LOW**

Assignee: Not assigned [✎](#)

Status: To review
This Security Hotspot needs to be reviewed to assess whether the code poses a risk.
[Change status ▼](#)

Mediatek86gestion/controleur/Controle.cs

```
358    /// <param name="mdp">La chaîne d'entrée</param>
359    /// <returns>Le hash calculé</returns>
360    public string CreateMD5(string mdp)
361    {
362        // Utilisation du string mdp pour calculer MD5 hash
363        using (MD5 md5 = MD5.Create())
364        {
365            byte[] inputBytes = Encoding.ASCII.GetBytes(mdp);
366            byte[] hashBytes = md5.ComputeHash(inputBytes);
367
368            // Conversion du vecteur vers un string hexadecimal
```

Figure 120: 'Security hotspot' selon SonarQube

L'algorithme de hachage md5 n'est pas suffisamment sécurisé pour des applications sensibles. Dans le cadre des catalogues d'une médiathèque toutefois la sécurité me semble suffisante. J'ai donc changé le statut en 'safe'.

8.4 Bilan pour la mission 4

Une mission sans difficultés particulières. La mise en place de l'hachage du mot de passe était un peu plus complexe que dans d'autres langages que j'ai pu rencontrer lors de ma formation, toutefois les informations nécessaires étaient facilement disponibles sur internet.

À l'issue de la mission toutes les fonctionnalités demandées sont opérationnelles et l'application est sécurisée.

9. Mission 5 : Qualité, tests et documentation technique

Voici les tâches demandées pour la cinquième mission :

- Contrôle de qualité de code avec SonarLint
- Création de tests fonctionnels avec SpecFlow
- Ajout de fonctionnalité de journalisation pour récupérer les erreurs qui peuvent survenir dans les blocs try/catch
- Vérification des commentaires normalisés et génération de la documentation technique

Comme d'habitude j'ai commencé par mettre à jour le storyboard :



Figure 121: La nouvelle liste 'To Do' pour la mission 5

9.1 Contrôle de qualité de code avec SonarLint

Pour cette tâche j'ai lancé une analyse de code avec SonarLint, directement depuis Visual Studio. Selon cette analyse il n'y a aucun erreur ni de warning. Ceci n'est pas tout à fait surprenant, étant donné qu'à la fin des missions précédentes à chaque fois j'ai fait une analyse avec SonarQube.

Jusqu'ici j'avais ignoré les 'Messages' cependant, SonarLint en donne 118

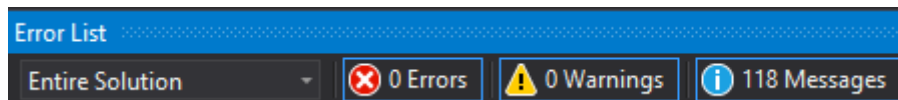


Figure 122: La liste d'erreurs dans Visual Studio 2019

IDE1006 Naming rule violation (94)

Parmi ces messages 94 des 118 concernent le nommage des variables. SonarLint est configuré pour demander l'utilisation de PascalCase partout. Toutefois, pour les fonctions événementielles créées automatiquement par Visual Studio (90 'messages') je préfère ne pas modifier. Les 4 autres cas étaient des oublis et sont dorénavant corrigés.

IDE0017 Object Initialization can be simplified (2)

En suivant les conseils de SonarLint j'ai appris une manière d'initialiser les objets que je ne connaissais pas encore. Au lieu de :

```
AlerteFinAbonnements alerteFinAbonnements = new AlerteFinAbonnements
    (controle);
alerteFinAbonnements.StartPosition = FormStartPosition.CenterParent;
alerteFinAbonnements.ShowDialog();
```

Figure 123: Initialisation 'classique'

SonarLint préconise :

```
AlerteFinAbonnements alerteFinAbonnements = new AlerteFinAbonnements
{
    StartPosition = FormStartPosition.CenterParent
};
alerteFinAbonnements.ShowDialog();
```

Figure 124: Initialisation optimisée

J'ai fait les modifications sur les deux endroits signalés.

IDE0018 Variable declaration can be inlined (3)

```
Double montant;  
bool success = Double.TryParse(montantSaisie, out montant);
```

Figure 125: Déclaration et initialisation séparées

devient :

```
bool success = Double.TryParse(montantSaisie, out double montant);
```

Figure 126: Déclaration 'inline'

IDE0028 Collection initialization can be simplified (18)

Dans le même esprit que pour le message IDE0017 l'initialisation des collections peut être simplifiée aussi :

```
List<string> requetes = new List<string>();  
requetes.Add("insert into exemplaire values  
(@idDocument,@numero,@dateAchat,@photo,@idEtat)");
```

Figure 127: Initialisation de collection 'classique'

devient :

```
List<string> requetes = new List<string>  
{  
    "insert into exemplaire values  
    (@idDocument,@numero,@dateAchat,@photo,@idEtat)"  
};
```

Figure 128: Initialisation de collection optimisée

Ceci est particulièrement appréciable quand on insère plusieurs éléments dans la collection :

```
List<string> requetes = new List<string>  
{  
    "insert into document values (@id, @titre, @image, @idRayon, @idPublic,  
    @idGenre)",  
    "insert into livres_dvd values (@id)",  
    "insert into livre values (@id, @isbn, @auteur, @collection)"  
};
```

Figure 129: Initialisation de collection à plusieurs éléments

IDE0044 Make field readonly (1)

Un oubli de ma part, qui était vite corrigé.

S1075 Refactor your code not to use hardcoded absolute paths or URIs.

Le chemin vers le dossier initial lors d'une recherche d'image pour une parution de revue est codé en dur dans la méthode. Ceci est vite réglé par l'implémentation d'une constante.

Toutefois, ce message m'a fait réaliser un oubli de ma part : je n'ai aucunement sécurisé la sélection d'images lors de l'ajout ou la modification de livres, DVD ou revues.

Du coup j'ai implémenté cette fonctionnalité pour les trois onglets :

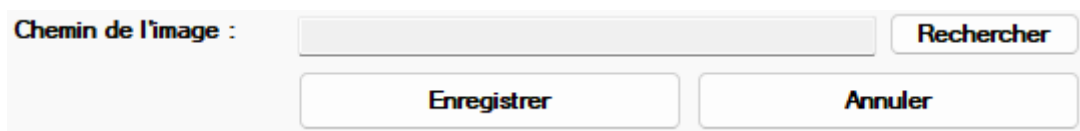


Figure 130: Adaptation de la vue

```
/// <summary>
/// Recherche image de la revue sur disque
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference | 0 changes | 0 authors, 0 changes
private void btnRevueRechercheImage_Click(object sender, EventArgs e)
{
    string filePath = "";
    OpenFileDialog openFileDialog = new OpenFileDialog
    {
        InitialDirectory = DOSSIERINITIALRECHERCHEIMAGE,
        Filter = "Files|*.jpg;*.bmp;*.jpeg;*.png;*.gif"
    };
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        filePath = openFileDialog.FileName;
    }
    txbRevueImage.Text = filePath;
    try
    {
        pcbRevueImage.Image = Image.FromFile(filePath);
    }
    catch
    {
        pcbRevueImage.Image = null;
    }
}
```

Figure 131: Méthode sécurisée, avec aussi l'utilisation de la constante pour le chemin de dossier initial.

Pour ce qui est la qualité de code, ils ne restent plus que deux messages (mise à part des IDE1006 que j'ai choisi d'ignorer) :

S107 Methods should not have too many parameters

J'avais rencontré ce message en début de l'atelier et j'avais choisi de l'ignorer. Il s'agit du constructeur pour la classe 'Document'. Effectivement, il n'y a pas le choix que d'utiliser autant de paramètres.

```
#pragma warning disable S107 // Methods should not have too many parameters
2 references | Carl Fremault, 6 days ago | 2 authors, 2 changes
public Document(string id, string titre, string image, string idGenre, string genre,
    string idPublic, string lePublic, string idRayon, string rayon)
#pragma warning restore S107 // Methods should not have too many parameters
{
```

Figure 132: Désactivation d'avertissement SonarLint concernant le nombre de paramètres

S1848 Objects should not be created to be dropped immediately without being used

Ce message est un faux positif que j'avais choisi d'ignorer en début de l'atelier. Il s'agit de l'initialisation du contrôleur dans la méthode 'main' :

```
namespace Mediatek86
{
    0 references | Carl Fremault, 6 days ago | 2 authors, 3 changes
    static class Program
    {
        /// <summary>
        /// Point d'entrée principal de l'application.
        /// </summary>
        [STAThread]
        0 references | Carl Fremault, 6 days ago | 2 authors, 3 changes
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
#pragma warning disable S1848 // Objects should not be created to be dropped immediately
            without being used
            new Controle();
#pragma warning restore S1848 // Objects should not be created to be dropped immediately
            without being used
        }
    }
}
```

Figure 133: Désactivation d'avertissement SonarLint d'objet 'inutilisé' (faux positif)

J'ai également fait un changement : le projet des tests unitaires étant situé dans un dossier séparé dans la solution principale, il n'était pas pris en compte par Git. Ainsi j'ai recréé ce projet, cette fois à l'intérieur de l'arborescence du projet principal.

Avant de passer à la prochaine tâche j'ai fait un autre commit & push.

9.2 Création de tests fonctionnels avec SpecFlow

J'ai commencé par créer un nouveau projet 'SpecFlowMediatek86' dans la solution puis j'ai ajouté un 'feature' et une classe test :

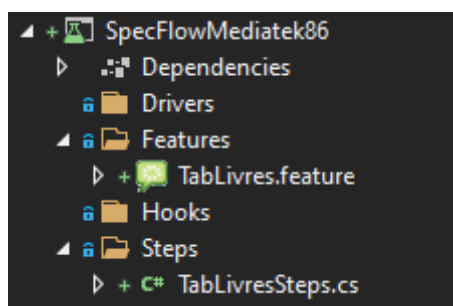


Figure 134: Création de nouveau projet dans la solution

Par la suite j'ai inséré un premier test simple dans les deux fichiers créés. Il s'agit d'un test de la fonctionnalité de recherche d'un livre en tapant son numéro et en cliquant sur le bouton 'Recherche'.

```
Feature: RechercheLivreNumero
    Test sur la recherche d'un document par son numéro

@rechercheLivreNumero
Scenario: Chercher un livre par son numero
    Given Je saisis la valeur 00025
    When Je clic sur le bouton Rechercher
    Then Les informations détaillées doivent afficher le titre L'archipel du danger
```

Figure 135: Scénario de test fonctionnel en langage Gherkin

```

public class RechercheLivreNumeroSteps
{
    private readonly FrmMediatek frmMediatek = new FrmMediatek(new Controle());

    [Given(@"Je saisis la valeur (.*)")]
    0 references | 0 changes | 0 authors, 0 changes
    public void GivenJeSaisisLaValeur(string valeur)
    {
        TextBox TxtValeur = (TextBox)frmMediatek.Controls["tabOngletsApplication"].Controls
            ["tabLivres"].Controls["grpLivresRecherche"].Controls["txbLivresNumRecherche"];
        frmMediatek.Visible = true;
        TxtValeur.Text = valeur;
    }

    [When(@"Je clic sur le bouton Rechercher")]
    0 references | 0 changes | 0 authors, 0 changes
    public void WhenJeClicSurLeBoutonRechercher()
    {
        Button BtnRechercher = (Button)frmMediatek.Controls["tabOngletsApplication"].Controls
            ["tabLivres"].Controls["grpLivresRecherche"].Controls["btnLivresNumRecherche"];
        frmMediatek.Visible = true;
        BtnRechercher.PerformClick();
    }

    [Then(@"Les informations détaillées doivent afficher le titre (.*)")]
    0 references | 0 changes | 0 authors, 0 changes
    public void ThenLesInformationsDetailleesDoiventAfficherLeTitre(string titreAttendu)
    {
        TextBox TxtTitre = (TextBox)frmMediatek.Controls["tabOngletsApplication"].Controls
            ["tabLivres"].Controls["grpLivresInfos"].Controls["txbLivresTitre"];
        string titreObtenu = TxtTitre.Text;
        Assert.AreEqual(titreAttendu, titreObtenu);
    }
}

```

Figure 136: Les 'steps' du test fonctionnel

Le test passe :

Test Detail Summary

- ✓ Chercher un livre par son numero in RechercheLivreNumero
 - Source: [RechercheLivreNumero.feature](#) line 5
 - Duration: 6.4 sec

Standard Output:

```

-> -> Loading plugin C:\Users\carlf\source\repos\Mediatek86gestion\SpecFlowMediatek86\bin\Debug\netcoreapp3.1\SpecFlowM
-> -> Missing [assembly:RuntimePlugin] attribute in SpecFlowMediatek86, Version=1.0.0.0, Culture=neutral, PublicKeyToka
-> -> Loading plugin C:\Users\carlf\source\repos\Mediatek86gestion\SpecFlowMediatek86\bin\Debug\netcoreapp3.1\LivingDoc
-> -> Loading plugin C:\Users\carlf\source\repos\Mediatek86gestion\SpecFlowMediatek86\bin\Debug\netcoreapp3.1\SpecRun.F
-> -> Using default config

Given Je saisis la valeur 00025
-> done: RechercheLivreNumeroSteps.GivenJeSaisisLaValeur("00025") (6.2s)

When Je clic sur le bouton Rechercher
-> done: RechercheLivreNumeroSteps.WhenJeClicSurLeBoutonRechercher() (0.0s)

Then Les informations détaillées doivent afficher le titre L'archipel du danger
-> done: RechercheLivreNumeroSteps.ThenLesInformationsDetailleesDoiventAfficherLeTitre("L'archipel du danger") (0.0s)

```

Figure 137: Détails du déroulement du test

Ensuite j'ai créé un test un peu plus compliqué, qui ajoute un nouveau livre puis tente de le retrouver :

```
Feature: AjouterLivrePuisChercher
  Ajouter puis rechercher un livre

  @ajouterLivrePuisChercher
  Scenario: Ajouter puis rechercher un livre
    Given Je clic sur le bouton ajouter
    And Je saisis le numéro de document 100
    And Je saisis le titre testTitre
    And Je saisis l'auteur testAuteur
    And Je saisis le genre Roman
    And Je saisis le public Adultes
    And Je saisis le rayon Littérature française
    And Je clic sur Enregistrer
    When Je saisis le titre testTitre
    Then Les informations détaillées doivent afficher le numéro de document 100
```

Figure 138: Scénario de test fonctionnel d'ajout d'un document

La logique des 'steps' est similaire, le test passe aussi. Quand j'ai voulu tourner le test une deuxième fois cependant je me suis rendu compte que (évidemment) le test avait inséré le livre dans la base de données. J'ai donc voulu créer un troisième test dans la suite pour supprimer un livre de la base de données.

Plusieurs problèmes ont surfacés, pour lesquels je n'ai trouvé aucune solution :

- Je n'ai pas réussi à 'simuler' le clic sur une ligne du dataGridView. J'ai trouvé plusieurs réponses à la question sur internet, notamment sur StackOverflow, mais aucune solution ne fonctionnait.
- En cherchant le livre 'supprimé' par son numéro (comme dans le premier test) l'application montre un 'messageBox' en signalant que le numéro n'a pas été trouvé. En soit c'est un comportement correct, toutefois je n'ai pas trouvé comment faire interagir SpecFlow avec les messageBox, et non plus comment faire 'réussir' le test suite à l'affichage de ce message.

Finalement j'ai procédé par la saisie du titre du livre dans le champ de recherche sur titre. Toutefois, avant de réellement supprimer le livre un bouton 'OK' doit être cliqué dans une MessageBox.

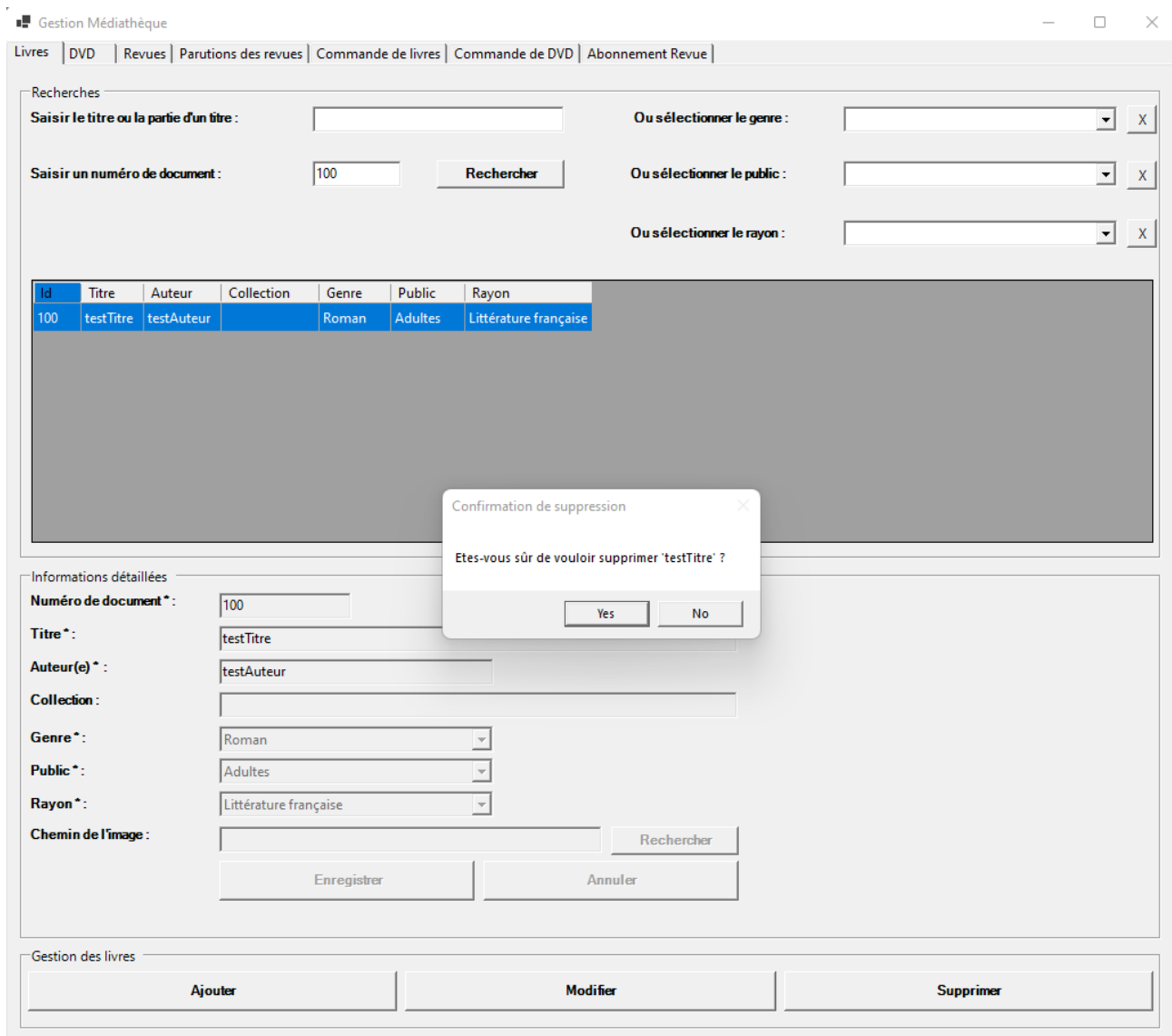


Figure 139: Demande de confirmation de suppression

De cette façon le test passe, mais il ne peut donc pas tourner sans interaction de la part de l'utilisateur (testeur), ce qui est décevant. Je n'ai pas trouvé de solution cependant.

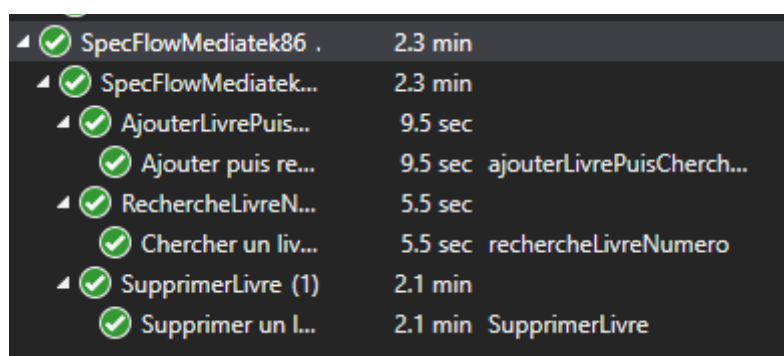


Figure 140: Les tests ont réussi

9.3 Journalisation pour récupérer les erreurs dans les blocs try/catch

La prochaine tâche demande de gérer la journalisation pour récupérer les erreurs qui peuvent se produire dans les blocs try/catch.

Une recherche montrait les différents endroits où ces blocs ont été utilisés :

- Dans la classe BddMySQL qui s'occupe de la connexion avec la base de données.
- Dans la classe Dao qui gère les requêtes à la base de données

Il y a quelques endroits dans la vue où des blocs try/catch ont été utilisés aussi, et ce pour trois cas de figure différents :

- La sélection d'un livre / DVD / Revue dans le dataGridView. Si le document n'est pas trouvé (donc : dans le catch) la zone de recherche est vidée. Ceci est un fonctionnement voulu et il ne génère pas d'exception. Il est inutile de créer une entrée dans les logs ici.
- L'affichage de l'image d'un document. Si l'affichage échoue (car aucun image est spécifié pour le document en question) la zone d'affichage de l'image est vidée. À nouveau, aucune utilité à journaliser ces événements.
- La validation lors de la réception d'un exemplaire d'une revue. Si le numéro d'exemplaire saisi n'est pas numérique, le 'parse' du champ échoue et le 'catch' affiche un MessageBox pour en informer l'utilisateur. À nouveau, pas de raison pour journaliser cet événement.

Pour la journalisation des interactions avec la base de données j'ai utilisé Serilog. La mission demande d'enregistrer les messages dans un fichier, du coup j'ai installé les packages 'Serilog' et 'Serilog.Sinks.File'.

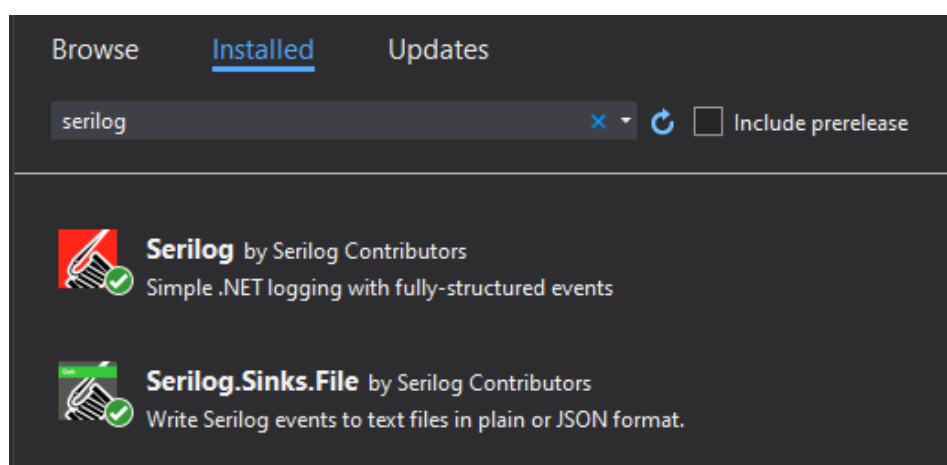


Figure 141: Installation des packages Serilog

Comme conseillé dans la documentation officielle de Serilog, j'ai initialisé le Logger dans la méthode 'main' de la classe Program.cs. Le niveau minimum que j'ai choisi est '2 : Information' (qui par ailleurs est le niveau par défaut). De toute façon je n'ai pas généré de logs en dessous du niveau '4 : Error' mais peut être qu'une évolution future de l'application en aurait besoin. J'ai choisi de générer un nouveau fichier par jour et aussi d'inscrire les événements au format JSON. Cela assurera une compatibilité large avec les différents logiciels d'analyse de journaux.

```
0 references | Carl Fremault, 7 days ago | 2 authors, 3 changes
static class Program
{
    /// <summary>
    /// Point d'entrée principal de l'application.
    /// </summary>
    [STAThread]
    0 references | Carl Fremault, 7 days ago | 2 authors, 3 changes
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);

        Log.Logger = new LoggerConfiguration()
            .MinimumLevel.Information()
            .WriteTo.File(new JsonFormatter(), "logs/log.txt",
                rollingInterval: RollingInterval.Day)
            .CreateLogger();
    }
}
```

Figure 142: Initialisation du logger

Il fallait générer un ajout au log dans chaque bloc try/catch concerné, en s'assurant que les informations sont claires et complètes.

Par exemple, pour la méthode 'ReqSelect' :

```

public void ReqSelect(string stringQuery, Dictionary<string, object> parameters)
{
    MySqlCommand command;

    try
    {
        command = new MySqlCommand(stringQuery, connection);
        if (!(parameters is null))
        {
            foreach (KeyValuePair<string, object> parameter in parameters)
            {
                command.Parameters.Add(new MySqlParameter(parameter.Key, parameter.Value));
            }
        }
        command.Prepare();
        reader = command.ExecuteReader();
    }
    catch (MySqlException e)
    {
        Log.Error("BddMySQL.ReqSelect catch **** stringQuery = " + stringQuery + " **** MySqlException = " + e.Message);
    }
    catch (InvalidOperationException e)
    {
        Log.Error("BddMySQL.ReqSelect catch **** stringQuery = " + stringQuery + " **** InvalidOperationException = " + e.Message);
        ErreurGraveBddNonAccessible(e);
    }
}

```

Figure 143: Inscription d'événements dans les logs pour la méthode 'ReqSelect'

Ensuite j'ai fait un petit test en insérant quelques erreurs :

- Erreur au niveau de la chaîne de connexion à la base de données
- Erreur au niveau d'une méthode du Dao qui utilise les méthodes ReqSelect et, par la suite, la méthode Read.
- Erreur au niveau d'une méthode du Dao qui utilise la méthode ReqUpdate

Le fichier log a bien été généré :


Name	Date modified	Type	Size
 log20220316.txt	16/03/2022 09:10	Text Document	2 KB

Figure 144: Le fichier généré dans l'explorateur de fichiers

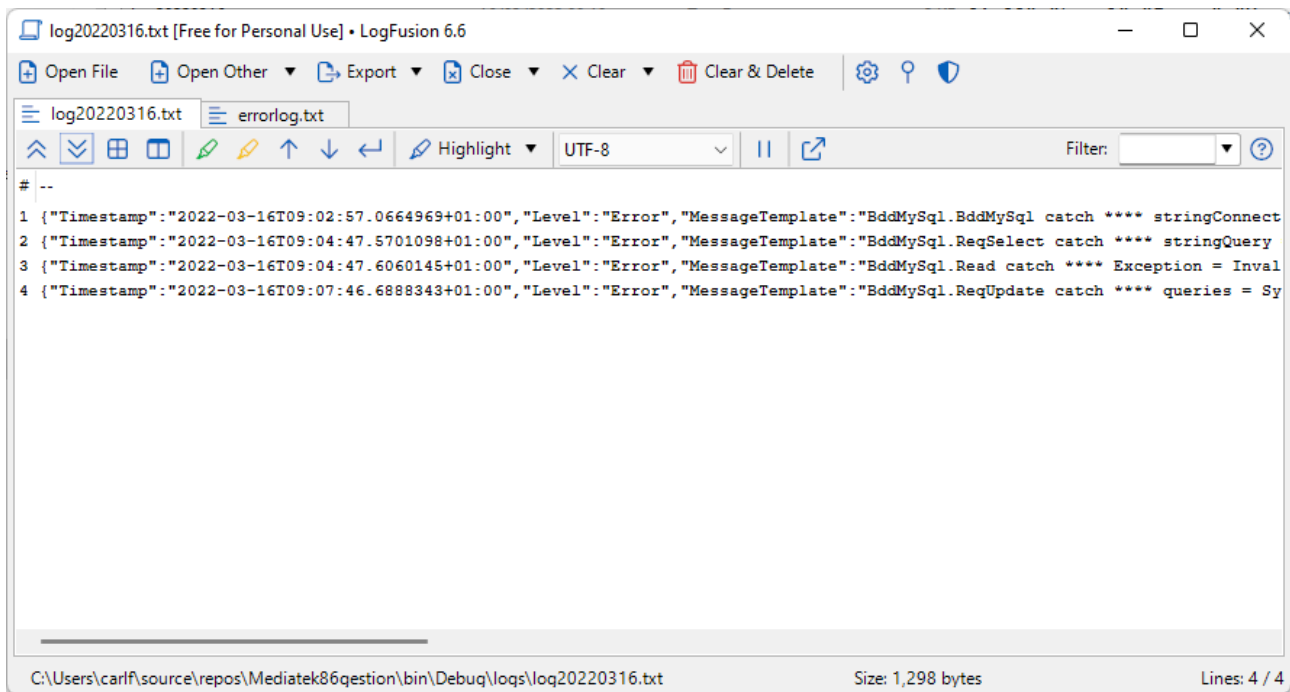


Figure 145: Exploitation du log avec LogFusion

Les messages sont satisfaisants sauf pour la dernière erreur : au niveau de la méthode ReqUpdate j'avais choisi d'utiliser une collection ('List') de requêtes. L'insertion de cette liste dans le message d'erreur ne donne aucune information intéressante :

```
{
  "Timestamp": "2022-03-16T09:07:46.6888343+01:00",
  "Level": "Error",
  "MessageTemplate": "BddMySQL.ReqUpdate catch **** queries = System.Collections.Generic.List`1[System.String] **** MySQLException = You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'insert into document values (?, ?, ?, ?, ?, ?)' at line 1"
}
```

J'ai fait une modification dans le code afin d'obtenir le détail des requêtes dans la liste.

```
catch (MySQLException e)
{
    transaction.Rollback();
    Log.Error("BddMySQL.ReqUpdate catch **** queries = " + string.Join(" --- ", queries) + " **** MySQLException = " + e.Message);
    throw;
}
catch (InvalidOperationException e)
{
    Log.Error("BddMySQL.ReqUpdate catch **** queries = " + string.Join(" --- ", queries) + " **** InvalidOperationException = " + e.Message);
    ErreurGraveBddNonAccessible();
}
```

Figure 146: Adaptation de l'implémentation des variables pour les logs

Le résultat est meilleur :

```
{ "Timestamp": "2022-03-16T09:26:05.6856335+01:00", "Level": "Error", "MessageTemplate": "BddMySQL.ReqUpdate catch **** queries = insert into document values (@id, @titre, @image, @idRayon, @idPublic, @idGenre) --- insert into livres_dvd values (@id) --- insert into livre values (@id, @isbn, @auteur, @collection) **** MySQLException = You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'insert into document values (?, ?, ?, ?, ?, ?)' at line 1" }
```

Par la suite j'ai examiné la classe Dao.cs. J'ai conclu qu'il n'y a pas de raison d'ajouter des événements de journalisation, comme les erreurs possibles résultent des erreurs dans la classe BddMySQL qui seraient déjà journalisées. Effectivement un test sur l'insertion d'un exemplaire avec un numéro déjà existant, qui retourne donc 'faux' depuis le Dao et affiche un messageBox à l'utilisateur lui précisant que le numéro est déjà utilisé, donne le message d'erreur suivant dans le log, provenant directement de la méthode ReqUpdate :

```
{ "Timestamp": "2022-03-16T09:59:18.4876585+01:00", "Level": "Error", "MessageTemplate": "BddMySQL.ReqUpdate catch **** queries = insert into document values (@id, @titre, @image, @idRayon, @idPublic, @idGenre) --- insert into livres_dvd values (@id) --- insert into livre values (@id, @isbn, @auteur, @collection) **** MySQLException = Duplicate entry '00017' for key 'PRIMARY'" }
```

Ajouter un deuxième message depuis le Dao serait inutile et encombrerait le journal. Il en est de même pour les autres blocs try/catch dans le Dao, comme ils exploitent tous les méthodes de la classe BddMySQL.

Il était donc temps de faire un autre commit & push.

9.4 Génération de la documentation technique

Pour la génération de la documentation technique j'ai utilisé SandCaste. Avant de procéder il fallait évidemment vérifier que les commentaires ont été insérés partout et qu'ils soient clairs et pertinents.

Par la suite j'ai généré la documentation technique au format XML. Cette génération se fait automatiquement lorsqu'on coche la case correspondante dans les propriétés du projet.

Avant de 'transformer' ces documents dans un format plus lisible à l'aide de SandCastle, et comme on approche la fin de l'atelier, j'ai fait un autre commit et push, suivi d'un merge avec la branche principale et une analyse SonarQube. Ceci pour éviter de devoir régénérer la documentation en cas de changement demandé par SonarQube (qui pourrait éventuellement nécessiter un changement dans les commentaires).

SonarQube demande de vérifier si l'utilisation du Logger est sécurisée :

The screenshot shows a SonarQube Security Hotspot interface. At the top, a message states: "Make sure that this logger's configuration is safe." with a link to "Configuring loggers is security-sensitive" (csharpersquid:S4792). Action buttons include "Add Comment", "Open in IDE", and "Get Permalink".

Metadata fields include:

- Category: Log Injection
- Review priority: LOW (highlighted in yellow)
- Assignee: Not assigned (with an edit icon)
- Status: To review (with a "Change status" dropdown)

The code snippet is from `/Program.cs` and shows the `Main()` method. Lines 21-25 are highlighted in yellow, indicating the hotspot location:

```
16 static void Main()
17 {
18     Application.EnableVisualStyles();
19     Application.SetCompatibleTextRenderingDefault(false);
20
21     Log.Logger = new LoggerConfiguration()
22         .MinimumLevel.Information()
23         .WriteTo.File(new JsonFormatter(), "logs/log.txt",
24             rollingInterval: RollingInterval.Day)
25         .CreateLogger();
26 }
```

Figure 147: Demande de vérification par SonarQube

Effectivement, certaines vulnérabilités suite à l'utilisation de logs sont connues :

The screenshot shows a table with three columns: "What's the risk?", "Are you at risk?", and "How can you fix it?". Below the table, a text block states: "Configuring loggers is security-sensitive. It has led in the past to the following vulnerabilities:"

- [CVE-2018-0285](#)
- [CVE-2000-1127](#)
- [CVE-2017-15113](#)
- [CVE-2015-5742](#)

Figure 148: Liste de vulnérabilités connues liés à la journalisation

Une vérification des liens vers la 'Common Vulnerabilities Enumeration' m'a assuré que nous ne sommes pas concernés par ces risques. Je peux donc ignorer l'avertissement et procéder avec SandCastle.

9.5 Bilan pour la mission 5

Même si dans l'ensemble cette mission s'est bien passée, avec une qualité de code satisfaisante, des logs fonctionnels et une documentation complète, je dois constater un petit bémol concernant l'implémentation des tests fonctionnels.

Non seulement SpecFlow était difficile à intégrer, avec notamment le besoin de modifier manuellement le fichier de configuration du projet 'SpecFlowMediatek86.csproj' pour que la solution puisse implémenter l'utilisation de Windows Forms, la manipulation de certains objets graphiques s'est avéré difficile voire impossible.

Pour les 'DataGridView' j'ai trouvé quelques pistes sur internet, sans toutefois parvenir à des résultats fonctionnels. Pour la manipulation des dialogues de confirmation (par exemple pour la suppression d'un document) je n'ai pas trouvé de solutions tout court.


Au final j'ai pu implémenter quelques tests qui requièrent toutefois quelques interactions de la part de l'utilisateur : la saisie du nom d'utilisateur et du mot de passe, la confirmation d'une demande de suppression.

10. Déploiement de l'application

La dernière étape avant de pouvoir 'publier' l'application sous forme de fichier .exe était la migration de la base de données dans le Cloud. J'ai opté pour la solution 'Azure Database pour MySQL' sur la plateforme cloud de Microsoft et j'ai donc créé la ressource nécessaire dans le portail Azure.

Serveur flexible ...

Microsoft

 Impossible de modifier les noms de serveur, la méthode de connectivité réseau, le HA redondant de zone et la redondance

De base (Changer)

Abonnement	Azure for Students
Groupe de ressources	MediatekA3
Nom du serveur	mediatek86
Nom de connexion de l'administrateur du serveur	mediadmin
Emplacement	France Central
Zone de disponibilité	Aucune préférence
Haute disponibilité	Non activé
Version de MySQL	5.7
Calcul + stockage	Expansible, B1s, 1 vCores, mémoire RAM Gio 1, stockage Gio 20
Période de conservation des sauvegardes (en jours)	7 jour(s)
Croissance automatique du stockage	Activé
Géoredondance	Non activé

Mise en réseau (Changer)


Méthode de connectivité	Accès public (adresses IP autorisées)
Autoriser l'accès public à partir d'un service Azure dans Azure sur ce serveur	Non
Règles de pare-feu	1
SSL/TLS	SSL est appliqué et la version de TLS est 1.2. Vous pouvez modifier cette valeur une fois que le serveur a été créé. En savoir plus 

Figure 149: Création d'une ressource 'Serveur Flexible Azure Database pour MySQL' dans Azure

Pour importer le script de la base de données j'ai installé MySQL WorkBench et j'ai crée une nouvelle connexion :

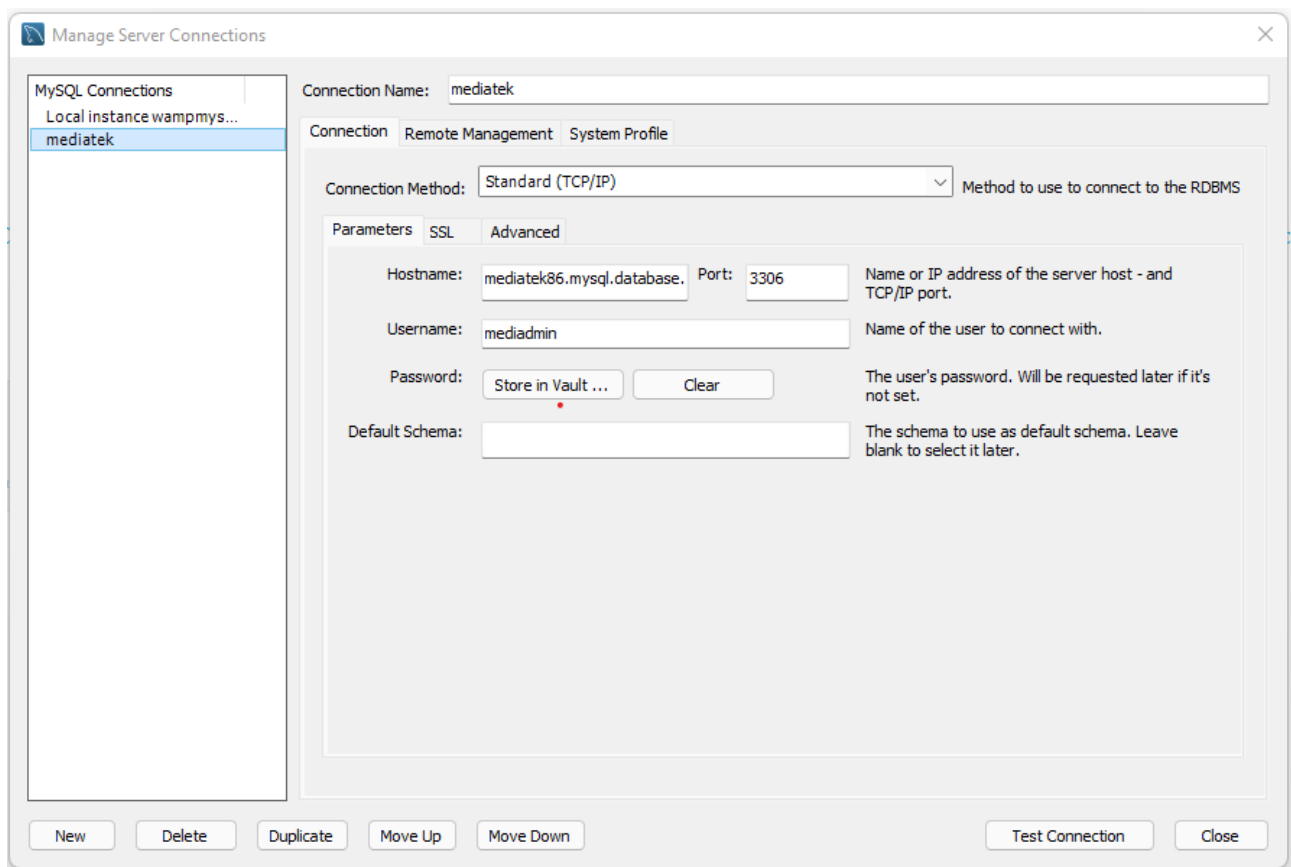


Figure 150: Connexion à la base de données Azure depuis MySQL Workbench

Puis j'ai importé le script, la base de données est créée :

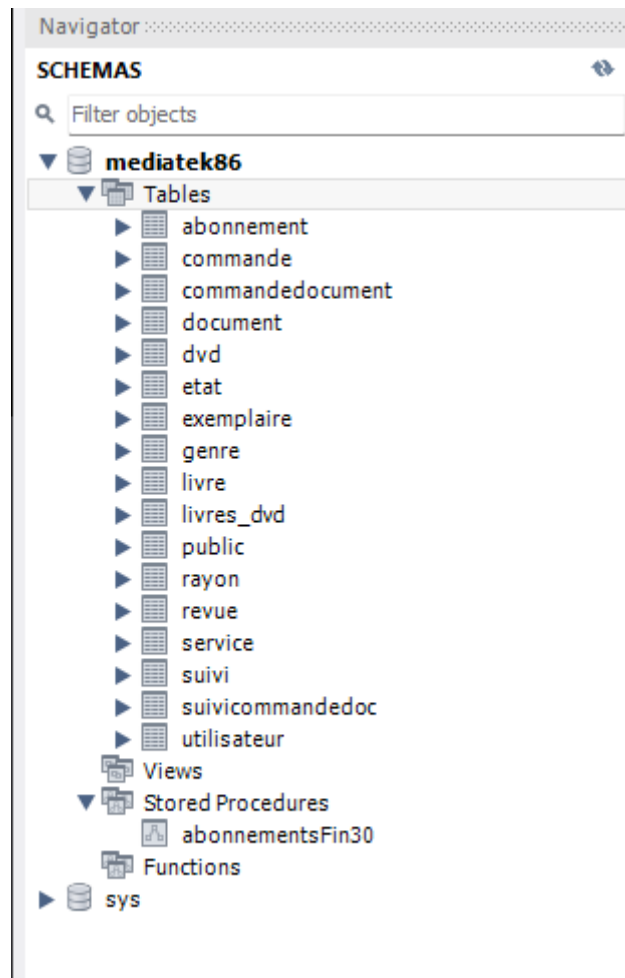


Figure 151: Arborescence de la base de données dans MySQL Workbench

Pour terminer j'ai modifié les données de connexion dans le Dao, pour permettre la connexion à la base de données dans le cloud Azure. Un test satisfaisant et un dernier commit, push et merge m'ont permis de terminer ce projet.

11. Bilan final

Cet atelier demandait d'utiliser la totalité des compétences acquises pendant les deux années des cours du Bloc 2 'Conception et développement d'applications'. Création et évolution d'application, modélisation et exploitation de la base de données y compris l'utilisation de transactions, triggers et procédures stockées, optimisation du code, génération de documentation technique ainsi que tests unitaires et fonctionnels.

À l'exception près des tests fonctionnels, qui ne sont pas autant automatisés que je l'aurais souhaité, je suis assez satisfait des résultats. Toutes les demandes exprimées dans le dossier documentaires ont été implémentées, tout en respectant le code et l'interface utilisateur existant, et en sécurisant l'interface afin de prévenir des erreurs de manipulation. Aussi, j'ai fait attention aux comforts 'utilisateur', en essayant d'optimiser l'utilisation du clavier.

J'ai également essayé de prévoir des évolutions futures de l'application, autant dans le code même, autant en maximisant les informations données dans les commentaires (et la documentation technique).

Évidemment quelques difficultés ont été rencontrées en route. En général il était facile d'y remédier à l'aide de la documentation officielle de Microsoft ou en cherchant sur des différents forums, notamment StackOverflow.

Pour finir j'étais particulièrement satisfait de mon implémentation de la base de données directement dans le cloud Azure, certes pas très compliqué au fin de compte, toutefois ce n'est pas un sujet qui avait été abordé en cours, où nous avons plutôt utilisés des machines virtuelles dans le cloud.

12. Compétences mobilisées pendant cet Atelier

- Répondre aux incidents et aux demandes d'assistance et d'évolution
 - Collecter, suivre et orienter des demandes
 - Traiter des demandes concernant les applications
- Travailler en mode projet
 - Analyser les objectifs et les modalités d'organisation d'un projet
 - Planifier les activités
- Mettre à disposition des utilisateurs un service informatique
 - Réaliser les tests d'intégration et d'acceptation d'un service
 - Déployer un service
 - Accompagner les utilisateurs dans la mise en place d'un service