# Compte rendu d’activité

## Evolution de l’application bureau de gestion documentaire de MediaTek86

03/04/2022

# Table des matières

# Contexte

## InfoTech Services 86

### Contexte InfoTech Services 86

InfoTech Services 86 (**ITS 86**) est une Entreprise de Services Numériques (ESN) spécialisée dans le développement informatique d'applications bureau, web ou mobile, ainsi que dans l'hébergement, l'infogérance, la gestion de parc informatique et l'ingénierie système et réseau. Elle répond régulièrement à des appels d'offres en tant que société d'infogérance et prestataire de services informatiques grâce à son pôle Développement qui permet de proposer des solutions d'hébergements ainsi qu'une expertise d'intégration de services, de développement de logiciel ou encore de gestion de bases de données.

## MediaTek86

### Contexte MediaTek86

Parmi les clients d'**ITS 86** se trouve le réseau de médiathèques de la Vienne, MediaTek86 dont la gestion du parc informatique et la numérisation des activités internes du réseau dépendent des services d'**ITS 86**. Dans l'optique d'accroître l'attractivité de son réseau de médiathèques, **MediaTek86** veut développer des outils numériques pour des usages en interne ainsi que des services en ligne pour ses usagers. Ces projets numériques sont pilotés par un chef de projet numérique chargé de la maîtrise d'ouvrage (MOA) auprès des ESN auxquelles **MediaTek86** a fait appel.

## Projet DigimediaTek86

### Contexte projet DigimediaTek86

Le projet DigimediaTek86 fait parti de cette liste de projet à mettre en place. L'objectif de ce projet s'étend sur deux axes, l'un interne puisqu'il implique la mise à disposition de services numériques pour ses employés, le second orienté vers ses usagers. Ici nous nous intéresseront à cette première partie puisque ce rapport détaille de la mise en place des nouvelles fonctionnalités dans l'application bureau de MediaTek86 dont la mise en place s'inscrit dans leur volonté à se numériser.

**Il est à noter qu'un document détaillant davantage les différents éléments du présent contexte est accessible à ce lien sous le nom de Contexte_MediaTek86.pdf**

# Application bureau MediaTek86 Gestion documentaire

## Contexte et objectifs de l'application MediaTek86 Gestion documentaire

L'application bureau de MediaTek86 est écrite en C#. Actuellement, elle permet de faire des recherches(tris et filtres) et d'afficher les informations sur les documents de la bibliothèque, qu'il s'agisse de livre, de DVD ou de revue. De plus, elle permet de gérer la réception des nouveaux numéro de revues. Cette application est actuellement utilisée sur plusieurs postes de travail au sein de la médiathèque et accède à la même base de donnée. Enfin, il n'y a actuellement qu'une seule fenêtre divisée en plusieurs onglets.

Dans un premier temps, l'objectif de cette évolution est de faire évoluer la gestion des documents en ajoutant des opération de type CRUD (ajouter, modifier et supprimer) en ajoutant un fonction de contrôle par un trigger. Dans un second temps, des interfaces de gestion de suivi des commandes des exemplaires physiques, puis des revues doivent être mises en place. Puis, un système de suivi de l'état physique des documents doit être élaboré. Enfin, il est nécessaire de mettre en place un système authentification sécurisé contre les injections SQL et que l'application n'offre pas les mêmes fonctionnalités en fonction du service de l'utilisateur de l'application.

## Rappel de l'existant

### La base de données

Pour accéder à la base de données, il faut récupérer le script de création de la base de données puis l'importer dans un logiciel de gestion de base de données sous **mySQL** permettant ainsi une gestion en local avec **phpMyAdmin**. Cette base de données, nommée **mediatek86** contient actuellement13 tables qui sont les suivantes :

- abonnement
- commande
- commandedocument
- document
- dvd
- etat
- exemplaire
- genre
- livre
- livre_dvd
- public
- rayon
- revue

Aussi, pour davantage de clarté, le script de création a été récupéré afin de construire le **schéma UML** de cette base via **WinDesign** en utilisant sa fonction de **reverse engineering**, ce qui nous permet d'avoir la figure suivante :

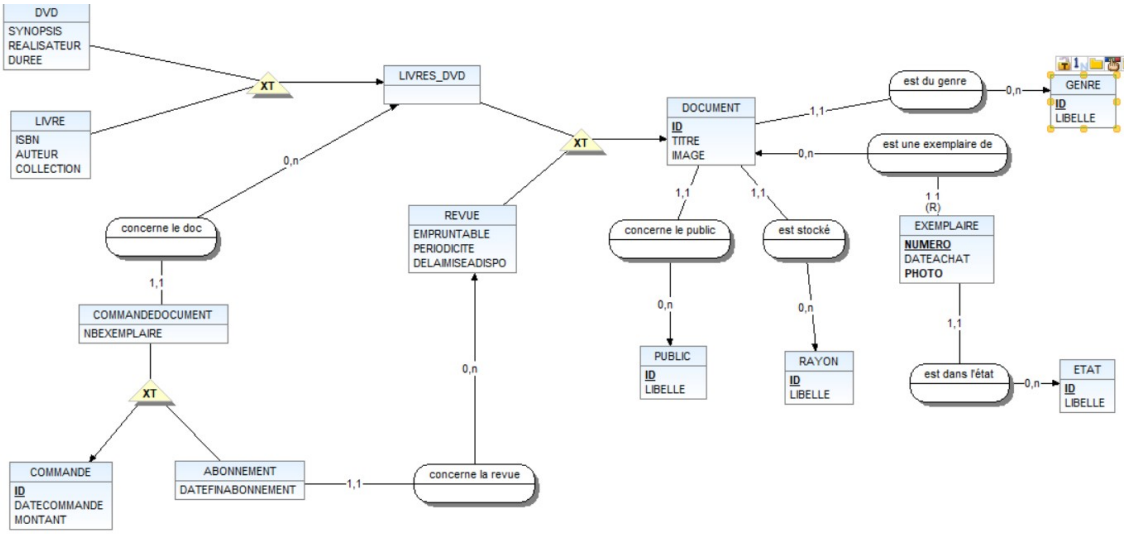



Figure 1: schéma UML de la base de donnée mediatek86 généré par WinDesign

Le script de création de cette base de donnée ainsi que son schéma UML sont accessibles via les liens suivant sous le nom script_sql_mediatek86formations.zip et SchemaUML_bd_mediatek86.png

## Les fonctionnalités existantes

A l'ouverture, l'application ouvre une fenêtre contenant plusieurs onglets : Livres, DVD, Revues et Parutions de Revues. Dans chacun de ces onglets, un ensemble de objets graphique permet d'effectuer des recherches selon différents critères et le type d'objet recherché(Public, rayon, genre, titre, numero). Au clic sur un onglet, une liste détaillée affiche l'ensemble des données contenues dans la base selon le type d'onglet ouvert, par exemple si l'on est sur l'onglet revues, l'ensemble des revues contenues dans la base seront listés. Il en va de même pour les autres onglets. Ainsi la fonction de recherche permet de n'afficher que les résultats cohérents par rapport aux critères de la recherche dans l'onglet concerné. Enfin, sur le clic d'une ligne de la liste, l'ensemble des informations détaillées de l'objet sélectionné sont affichés dans un groupbox d'affichage.

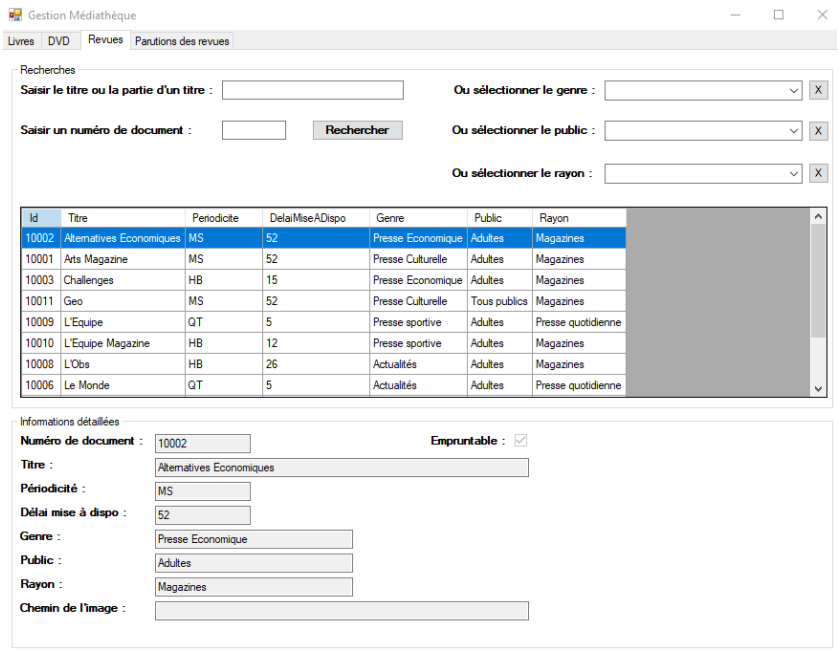



Figure 2: Exemple d'affichage de l'application à l'état initial

Enfin, les différents documents (contextes détaillés, script SQL, documentations) sont mis à dispositions à l'adresse suivante : documentations. De même, le code source de l'application d'origine est accessible ici : code source.

## Les packages existants

Le code source fourni contient 5 packages qui sont les suivants : **bdd**, **controleur**, **metier,** **modele** et **vue**.

### Package bdd

Ce package contient la Classe **DbbMySql** qui permet les exécutions et envoies des requêtes vers la base de données. Le code source initial de cette classe est accessible depuis les liens suivants : BddMySql

### Package contrôleur

Ce package contient la Classe **Controle** permettant de relier les différentes Méthodes et Classes de manière contrôlée, notamment entre le package vue et le package modele. Le code source initial de cette classe est accessible depuis les liens suivants : Controle

### Package modele

Ce package contient la Classe **Dao** qui permet de rassembler l'ensemble des méthodes nécessitant la création de requêtes SQL vers la base de données mediatek86. Elle utilise

les méthodes du package bdd et de sa Classe **DbbMySql** pour leurs exécutions. Le code source initial de cette classe est accessible depuis les liens suivants : Dao

#### Package metier

Ce package contient l'ensemble des Classes types métiers permettant la création d'Objets personnalisés nécessaires à l'application, tels que **Utilisateur**, **Abonnement**, **Commande**, **Documents**, etc. Les différents codes sources initiaux de ces classes sont accessibles depuis les liens suivants : metiers

#### Package vue

Ce package contient la Classe **FrmMeditek** de type Form qui permet l'affichage des objets graphiques ainsi que leurs interactions. Le code source initial de cette classe est accessible depuis les liens suivants : vue

#### Program.cs

Ce fichier permet l'initialisation de l'application ainsi que sa mise en route. Le code source initial de cette classe est accessible depuis les liens suivants : program

## Expressions des besoins

### Remarques générales

L'évolution de l'application doit respecter la structure des couches ainsi que la logique du code actuelle. Aussi, il faut restreindre les possibilités de manipulations des utilisateurs, en mode lecture seule pour certains objets, le contrôle les injections SQL et éviter les saisies utilisateur par une liste de valeurs prédéfinies.

### Fonctionnalité gestion des documents

(non implémenté)

L'application doit pouvoir ajouter, modifier ou supprimer des documents de types livres, DVD et revues.

dans les onglets actuels (Livres, Dvd, Revues),

- Ajouter les fonctionnalités qui permettent d'ajouter, de modifier ou de supprimer un document
- Sécurités pour éviter des erreurs de manipulation
- Créer le trigger qui contrôle la contrainte de partition de l'héritage sur Document
- Idem pour LivresDvd.

### Fonctionnalité de gestion des commandes

Pour les commandes des livres ou des DVD, il faut que l'application soit capable de gérer et suivre l'évolution d'une commande qui passe par différents stade : (en cours, livrée, réglée, relancée). Cette fonctionnalité n'étant pas initialement gérée ni dans l'application, ni dans la base, il faudra mettre en place ce système. Il est aussi nécessaire de mettre en place une interface de visualisation du suivi et de gestion de ces commandes. Une fois au stade « livré », le ou les articles commandés doivent être automatiquement généré dans la BDD avec un numéro séquentiel par rapport au document concerné. Enfin, si une commande n'est pas au stade « livré », elle doit pouvoir être supprimé.

Concernant la commande de revues, une commande est un abonnement, ou un renouvellement d'abonnement. Il est donc nécessaire de mettre en place une interface de visualisation et de gestion sur laquelle il sera possible de consulter l'ensemble des abonnements de revues, ceux qui sont expirés. Enfin, une commande de revue est supprimable lorsqu'un aucun exemplaire lié à cette commande n'est enregistré.

### Fonctionnalité de gestion du suivi de l'état des documents

(non implémenté)

Cette fonctionnalité permet le suivi de l'état des documents physiques : livres, DVD et revue. A la création d'un nouveau document, il est marqué automatiquement comme « neuf », puis l'application doit pouvoir changer l'état vers : usagé, détérioré, inutilisable. Enfin, il doit être possible de supprimer un exemplaire.

### Fonctionnalité d'authentification des utilisateurs

L'application doit démarrer sur une demande d'authentification permettant de déterminer si l'employé est reconnu et son service. Dans le cas d'un employé du service Culture, un message doit l'informer que l'application n'est pas accessible pour leur service.

Une restriction des droits d'accès aux différentes fonctionnalités doivent êtres mises en place selon les services des différents employés.

#### Service Administratif :

- Tout accès

#### Service Prêts

- Consultation du catalogue disponible (livre, DVD, Revue)

#### Service Culture

- Aucuns accès

#### Administrateur

- Tout accès

### Qualité, logs et tests unitaires

Afin d'assurer la propreté du code de l'application, l'extension **SonarLint** sera mise en place afin de trouver les erreurs de code ou de mauvaises mise en forme. Aussi des logs de journalisation avec l'extension **Serilog** doivent être ajoutés dans les catchs qui contiennent les messages consoles afin de les enregistrer dans un fichier. De plus, des tests unitaires seront mis en place afin de vérifier la non régression de l'application actuelle lors du développement sur la certaines des fonctionnalités clés.

## Informations techniques et outils utilisés

### Spécificités IDE

L'évolution de l'application récupérée a été effectué avec la version 16.11.10 de l'IDE Microsoft Visual Studio 2019.

### Spécificités outils et plugins

La revue du code produit a été effectuée par l'intermédiaire de **SonarLint** qui est un plugin permettant d'analyser du code dans l'IDE. Ainsi, il permet de mettre en évidence les éventuelles vulnérabilités et bogues lors de l'écriture du code comme le montre la *figure 1*.

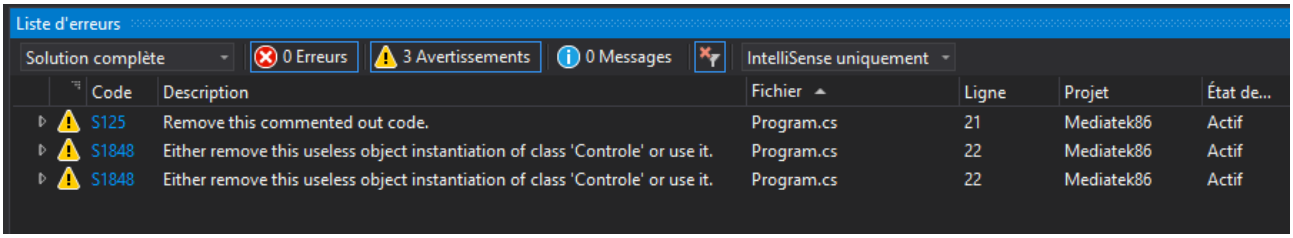



Figure 3: Exemple de contre rendu d'analyse de code avec SonarLint

De même, le logiciel SonarQube a été intégré au projet afin de mesurer la qualité du code Il est capable de recenser plusieurs niveaux de bogues ou d'erreurs tels que le niveau de documentation ou couverture de test, de repérer les duplications de code et le non respect de règles de programmation ainsi que la complexité du code produit. Enfin, il permet d'évaluer l'application sous plusieurs notes comme le montre la figure ci-dessous :

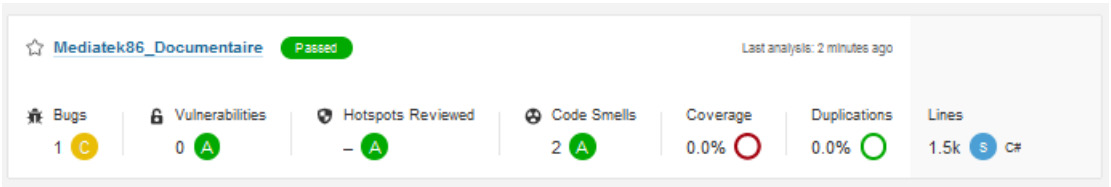



Figure 4: Exemple de résultat d'évaluation par SonarQube

### Documentation technique

La documentation technique des diverses classes, propriétés et méthodes établis lors de la poursuite du développement de l'application bureau a été établi avec Doxygen qui est un logiciel sous libre licence permettant de générer une documentation technique standard. Ce dernier a été choisit pour ses fonctionnalités de personnalisations. Cette documentation est accessible en ligne à l'adresse suivante :Documentation technique de l'application bureau.

### Suivi de projet

Le suivit de projet a été établi avec la plateforme Trello qui permet d'établir un plan d'évolution et de suivit de projet. Enfin, l'intégralité des codes sources produits ont été stocké sur la plateforme Github au nom de projet Mediatek86_Documentaire enfin le Portfolio présentant l'ensemble de ce projet est accessible ici.

# Réalisation

## Préparation de l'environnement de travail

### Établir le plan de suivi sur la plateforme Trello

La mise en place du suivi de projet a commencé par la création d'un tableau publique sur Trello du nom d'gestionmediatek86. Ainsi, il est possible de constater qu'il comporte 6 listes dont les titres vont de la mission dite 0 à la mission 7 dite Mission bilan. Chacune de ces listes contiennent des cartes qui correspondent à différentes étapes des missions à effectuer. Une fonction d'étiquette de couleur permet d'avoir un suivi visuel de ce qui est **« Fini »** en violet, **« en cours »** en jaune et **« à venir »** en rouge.

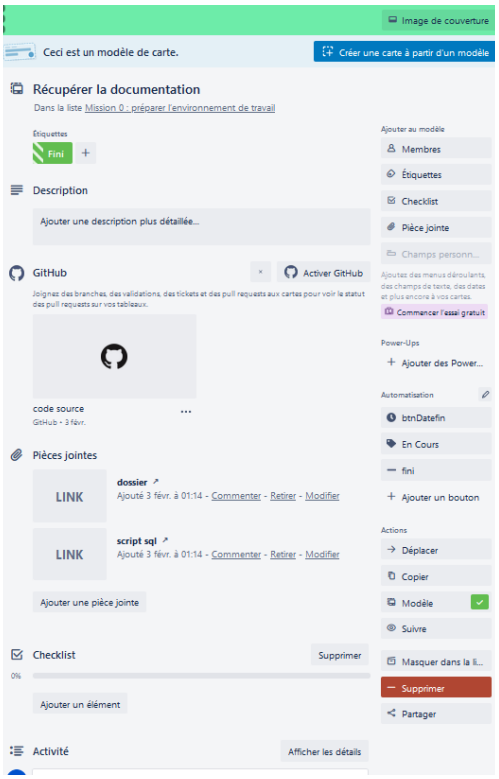



Figure 5: Exemples de carte de suivi avec étiquette, dates de fin et check-list

La *figure 5* ci-contre illustre un exemple de contenu d'une de ces listes et la *figure 6* montre les différents menus de création.

Enfin, un intervalle de temps en jours a été établi afin d'estimer le temps de réalisation de chacune des tâches. De même, des check-lists par tache ont été mise en place afin d'obtenir un meilleur suivi.

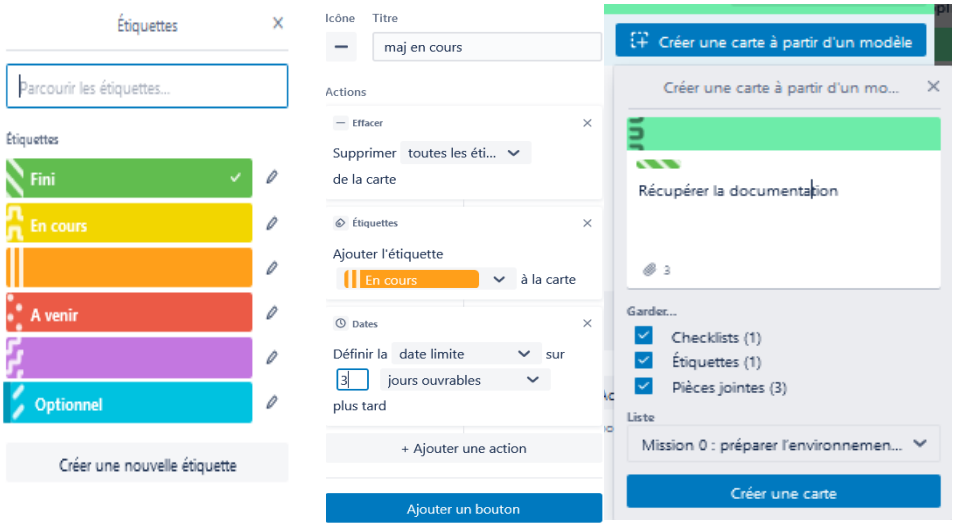



Figure 6: Menus de création d'étiquettes et des boutons d'automatisation

Aussi, afin d'accélérer la mise à jour des étiquettes, un bouton d'automatisation, « fini » à été mis en place, comme le montre la figure , ainsi en appuyant sur le bouton de la carte, lorsque l'étape est terminée l'étiquette passe de **« En cours »** à **« Fini »**. De plus, à la création d'une carte il est possible de la marquer comme carte modèle, ce qui permet de créer plus rapidement de nouvelles cartes.

Enfin, le dépôt sur GitHub a été liée à ce suivi de projet permettant d'obtenir en tant que PowerUp, qui sont des plugins permettant d'ajouter des fonctionnalités supplémentaires au tableau.


Figure 7: Powerup Github : association repository github de l'application

Concernant ce lien entre le tableau et le dépôt GitHub, il permet d'abord d'établir un lien constant vers le projet, mais aussi de faire apparaître le message du dernier commit, renseignant ainsi de l'avancé du projet. Pour finir, ce tableau Trello étant publique, il est consultable à l'adresse suivante .

Il est ainsi possible d'énumérer les différentes missions :

- **Mission 0 : Préparer l'environnement de travail – temps estimé 2h**
  - **Mise en place du tableau sur Trello**
  - **Récupération de la documentation**
    - Script SQL de la base de données
    - Code source de l'application initial
    - Dossier de l'existant et des besoins
  - **Mise en place de l'environnement de développement**
    - Installation de WampServer
    - Création de la base de donnée distante dans PhpMyAdmin
    - Installation de l'IDE Visual Studio 2019
    - Tester le code source sur l'IDE
    - Création du dépôt sur Github


Figure 8: Cartes de la mission 0 préparer l'environnement de travail

- Mise en place d'analyse SonarQube

- **Mission 1 : Gérer les opération CRUD sur les documents – temps estimé 7h** *(Optionnel)*
  - 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.

- **Mission 2 : Coder les fonctionnalités de gestion des commandes – temps estimé 15h**
  - **Dans la base de données**
    - Créer la table 'Suivi' qui contient les différentes étapes de suivi d'une commande de document de type livre ou dvd.
    - Relier cette table à CommandeDocument.
  - **Dans l'application**
    - **Onglet pour gérer les commandes de livres.**
      - Dans l'onglet, permettre la sélection d'un livre par son numéro, afficher les informations du livre ainsi que la liste des commandes, triée par date (ordre inverse de la chronologie).
      - La liste doit comporter les informations suivantes : date de la commande, montant, nombre d'exemplaires commandés et l'étape de suivi de la commande.
      - Créer un groupbox qui permet de saisir les informations d'une nouvelle commande et de l'enregistrer.
      - Lors de l'enregistrement de la commande, l'étape de suivi doit être mise à "en cours".
      - Permettre de modifier l'étape de suivi d'une commande en respectant certaines règles (une commande livrée ou réglée ne peut pas revenir à une étape précédente : en cours ou relancée, une commande ne peut pas être réglée si elle n'est pas livrée).
      - Permettre de supprimer une commande uniquement si elle n'est pas encore livrée.
      - Créer le trigger qui se déclenche si une commande passe à l'étape "livrée" et qui crée autant de tuples dans la table "Exemplaire" que

nécessaires, en valorisant la date d'achat avec la date de commande et en mettant l'état de l'exemplaire à "neuf". Le numéro d'exemplaire doit être séquentiel par rapport au livre concerné.

- **Onglet pour gérer les commandes de DVD**
  - Créer un onglet pour gérer les commandes de revues
  - une commande représente un nouvel abonnement ou le renouvellement d'un abonnement. Dans le cas d'un nouvel abonnement, la revue sera préalablement créée dans l'onglet Revues. Donc, dans l'onglet des commandes de revues, il n'y a pas de distinction entre un nouvel abonnement et un renouvellement.
  - Permettre la sélection d'une revue par son numéro, afficher les informations de la revue ainsi que la liste des commandes (abonnements), triée par date (ordre inverse de la chronologie). La liste doit comporter les informations suivantes : date de la commande, montant et date de fin d'abonnement.
  - Créer un groupbox qui permet de saisir les informations d'une nouvelle commande (nouvel abonnement ou renouvellement, le principe est identique) et de l'enregistrer. Une commande de revue peut être supprimée, si aucun exemplaire n'est rattaché (donc, en vérifiant la date de l'exemplaire, comprise entre la date de la commande et la date de fin d'abonnement).
  - Créer et utiliser la méthode 'ParutionDansAbonnement' qui reçoit en paramètre 3 dates (date commande, date fin abonnement, date parution) et qui retourne vrai si la date de parution est entre les 2 autres dates.
  - **Créer le test unitaire sur cette méthode.**
- La présentation de chaque onglet de gestion des commandes doit être similaire à l'onglet "Parutions des revues".
- Dans toutes les listes, permettre le tri sur les colonnes.
- 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 Commande.
- Créer une procédure stockée qui permet d'obtenir la liste des revues dont l'abonnement se termine dans moins de 30 jours. Dès l'ouverture de l'application, ouvrir une petite fenêtre d'alerte rappelant la liste de ces revues (titre et date de fin abonnement) triée sur la date dans l'ordre chronologique.

- **Mission 3: Gérer le suivi des documents– temps estimé 6h**
  - Agrandir la fenêtre en hauteur
  - Onglet Livres, partie basse,
    - Ajouter la liste des exemplaires du livre sélectionné.

        - Contient les colonnes : numéro d'exemplaire, date achat et libellé de l'état.
        - La liste doit être triée par date d'achat, dans l'ordre inverse de la chronologie, et le clic sur une colonne doit permettre le tri sur la colonne.
        - Sur la sélection d'un exemplaire, il doit être possible de changer son état
    - le principe est le même pour les DVD ;
    - Onglet "Parutions des revues" :
        - Liste des parutions,
        - remplacer la colonne "Photo" par "Etat".
        - Permettre aussi le changement d'état ;
    - Permettre de supprimer un exemplaire.

- **Mission 4: Mettre en place les authentification – temps estimé 6h**
    - Dans la base de données, ajouter une table Utilisateur et une table Service, sachant que chaque utilisateur ne fait partie que d'un service. Pour réaliser les tests, remplir les tables d'exemples ;
    - Ajout d'une première fenêtre d'authentification que le contrôleur ouvre en première ;
    - Selon le service de la personne authentifiée, empêcher certains accès en rendant invisibles ou inactifs certains onglets ou objets graphiques ;
    - dans le cas du service Culture qui n'a accès à rien, afficher un message précisant que les droits ne sont pas suffisants pour accéder à cette application, puis fermer l'application ;
    - faire en sorte que l'alerte de fin d'abonnement n'apparaisse que pour les personnes concernées (qui gèrent les commandes).
- **Mission 5: Qualité, tests et documentation technique – temps estimé 6h**
    - Contrôler la qualité du code avec SonarLint
    - Créer des tests fonctionnels avec MS Test
    - Ajouter des logs dans les catch qui contiennent des affichages consoles et enregistrer ces logs dans un fichier texte avec Serilog
    - Création de la documentation technique de l'application
- **Mission 6: Création d'une vidéo de démonstration via OBS – temps estimé 1h** (+ 24h pour son acceptation par la modération de la plateforme d'hébergement)
- **Mission Bilan**
    - Déployer l'application et la mettre à disposition sur la plateforme de dépôt.
    - Mise en ligne de la base de donnée distante
    - Rédaction du contre rendu d'activité.
    - Mise en place du portfolio de l'activité.

La *figure 9* suivante illustre ce tableau de suivi à la fin de son élaboration, soit la Mission 0.

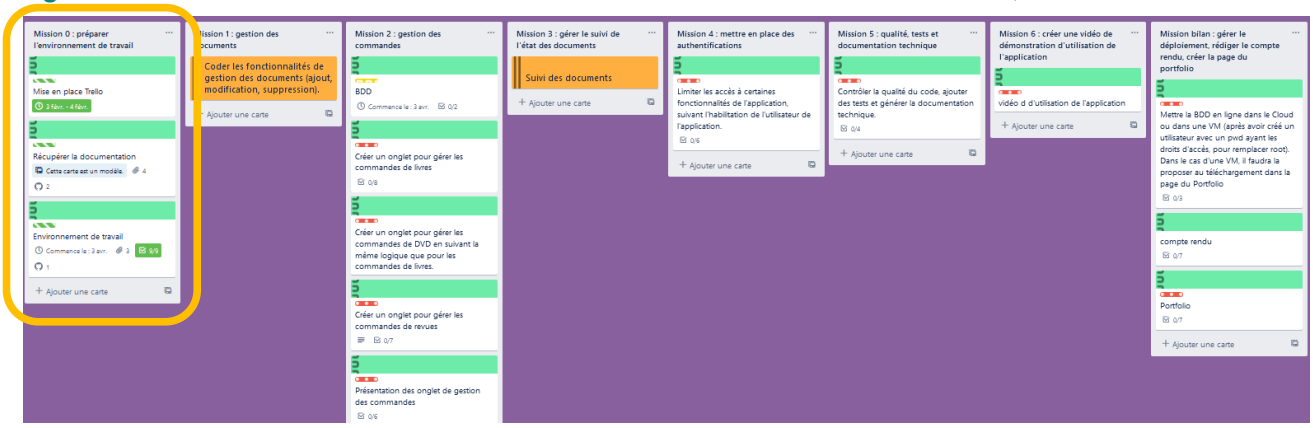



Figure 9: Tableau de suivi de projet Trello, début de la mission 0

## Mise en place de l'environnement de développement

### Importation de la base de donnée

Après ouverture de WampServer, dans la barre des taches de Windows, il est nécessaire de cliquer sur **phpMyAdmin**. A cet instant, une nouvelle page du navigateur s'ouvre pour afficher un nouveau formulaire, cf *figure 10*. Il s'agit de la page de connexion à phpMyAdmin, il faut alors remplir en tant qu'utilisateur : « root » et laisser vide pour le mot de passe, à moins que ce dernier ait été configuré.

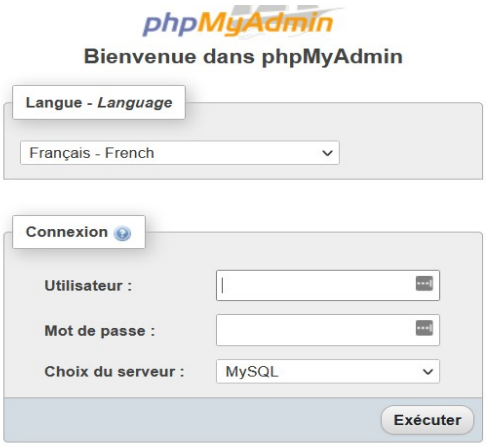



Figure 10: Fenêtre de connexion à phpMyAdmin

En arrivant dans le menu de phpMyAdmin, il faut alors aller dans l'onglet « importer » qui affiche alors la *figure 12*.

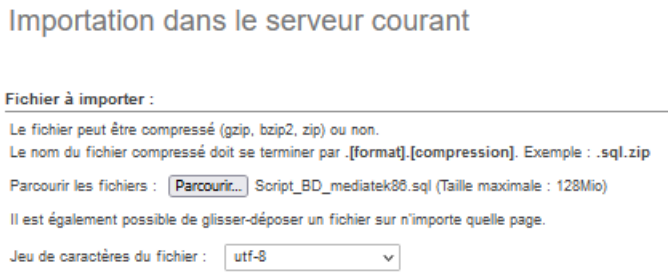



Figure 11: Menu d'importation de script SQL de phpMyAdmin

Il convient ainsi de renseigner le script de création de la base de données, soit **Script_BD_mediatek86.sql**, puis en bas de page cliquer sur « Executer ». La page finit par s'actualiser pour afficher plusieurs messages indiquant que l'importation a réussi, comme l'illustre la *figure 12*.


Figure 12: Exemples de messages confirmant la bonne importation du script de création de la base de données

Il est alors possible de voir la base de donnée **mediatek86** apparaître dans la liste des bases disponibles, puis en cliquant dessus l'ensemble de ses 13 tables apparaissent.

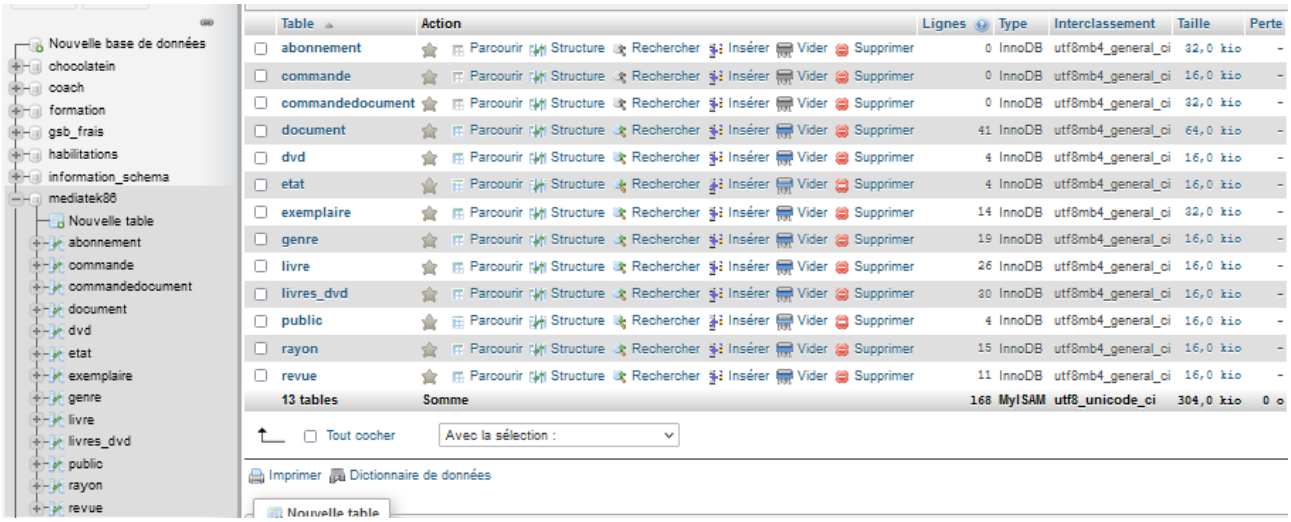



Figure 13: Structure des tables de la base de donnée

### Mise en place de l'IDE Visual Studio

Visual Studio est un IDE permettant le développement d'application C#. L'IDE est téléchargeable à ce lien, ici il s'agit de la version 2019. Après l'installation de cet IDE avec les options par défaut, il convient de le démarrer en modeadministrateur afin de simplifier les différents éléments de démarrage. De plus, ici le logiciel a été configuré pour s'ouvrir automatiquement dans ce mode. Pour cela, il convient d'effectuer un clic droit sur l'icone de lancement de Visual Studio, d'aller dans « Propriétés», puis à l'ouverture de la fenêtre des propriétés, de cliquer sur « Avancé » puis de cocher « exécuter en tant qu'administrateur comme le montre la *figure 14.*

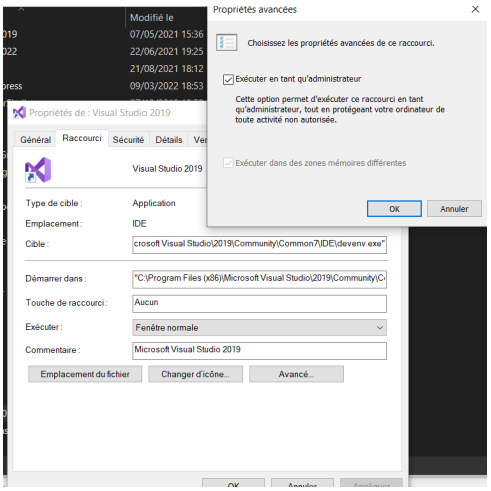



Figure 14: Fenêtre des propriétés de l'icône de lancement de l'IDE

### Récupération du code source

La récupération du code source ce fait sur la plateforme GitHub à cette adresse en cliquant sur le bouton vert à droite avec inscrit « Code » puis de sélectionner « **Download zip** ». Après téléchargement, avoir dézipper le dossier et avoir placer le projet dans le dossier de traitement, ici dans : **%user\Programmation\Git\ Mediatek86_Documentaire**

Pour ouvrir le projet, il convient de se rendre dans le répertoire de dépôt local puis de cliquer sur le lien du projet pour Visual Studio : **Mediatek86.sln**. Enfin, il suffit de cliquer sur l'icône de lecture verte à côté de démarrer afin de lancer l'application.

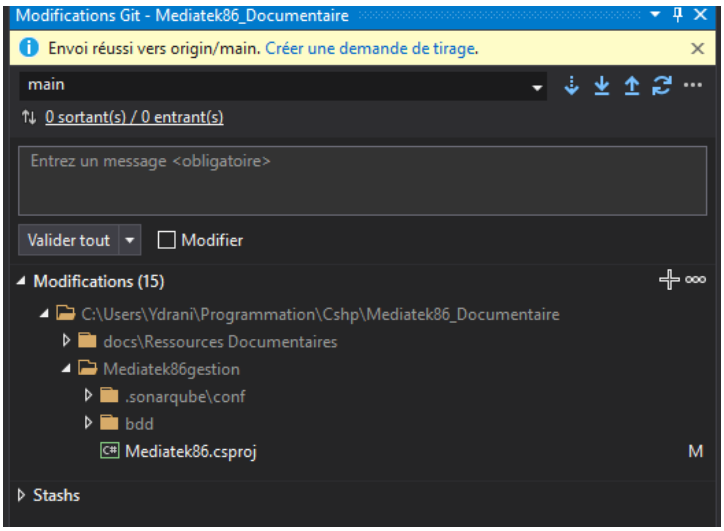



Figure 15: Fenêtre de controle de version Github sur VisualStudio

### Lier le dépôt Github avec Visual Studio

Afin de pouvoir directement gérer le dépôt GitHub depuis Visual Studio, il convient d'aller dans l'onglet «**Modification Git** » puis de saisir un message de commit dans la fenêtre qui

apparaît. A cet instant, les modifications non intégrés au dépôt s'affiche en dans l'onglet **Modification**. Dans le cas présent, le dépôt étant préalablement créé et le compte de dépôt étant déjà lié à Visual Studio, il suffit de commiter puis de cliquer sur « **valider tout et pousser** ». A la réussite de l'envoi, une pop-up s'affiche.

### Mise en place de sonarQube

Pour utiliser cette plateforme, il convient de télécharger l'application puis de l'installer dans le dossier racine **C :** du poste de travail. Ensuite, il suffit de lancer le fichier **StartSonar.bat.** Plusieurs fenêtres d'invite de commande s'exécute afin de lancer le serveur SonarQube local, de se rendre à l'adresse **localhost:9000** dans un navigateur, puis de se connecter avec login et mot de passe admin.

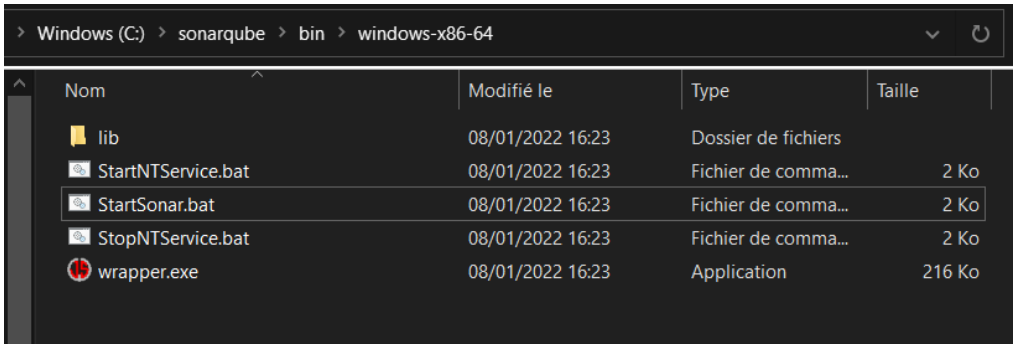



Figure 16: Localisation du launcher du server de SonarQube

Aussi dans le dossier racine du poste de travail, dans le dossier de **sonarscanner**, il faut modifier le fichier **SonarQube.Analysis.xml** afin de remplacer la valeur de **sonar.login** par celle d'un token généré pour sonarQube et éventuellement celui de **sonar.host.url** par celui de la valeur de **url** du serveur **sonarQube**. Puis, dans le dossier du projet de l'application, il faut ajouter la ligne de code suivante dans le fichier **Mediatek86.csproj**

```
<PropertyGroup>
<!-- Project is not a test project -->
    <SonarQubeTestProject>false</SonarQubeTestProject>
</PropertyGroup>
```

Une fois dans le menu de **sonarQube**, il convient de créer un nouveau projet et de le créer avec les options en local, de générer un token ou en récupérer un déjà existant, puis de choisir le build .**NET**, puis .**NETFramework**. A la sélection de ce dernier plusieurs lignes de commandes apparaissent qu'il convient d'exécuter dans une invite de commande dans le répertoire de l'application.

```
SonarScanner.MSBuild.exe begin /k:"Mediatek86_Documentaire" /d:sonar.host.url=
"http://localhost:9000" /d:sonar.login="xxxxx"

MsBuild.exe /t:Rebuild
```

```
SonarScanner.MSBuild.exe end /d:sonar.login="xxxx"
```

Où, « **xxxx** » est le token généré à la création du projet. Afin de mettre l'analyse du projet à jour, il suffit d'exécuter la dernière ligne. Ainsi, une fois la dernière ligne exécuté et après avoir refresh la page il est possible d'obtenir l'affichage suivant :

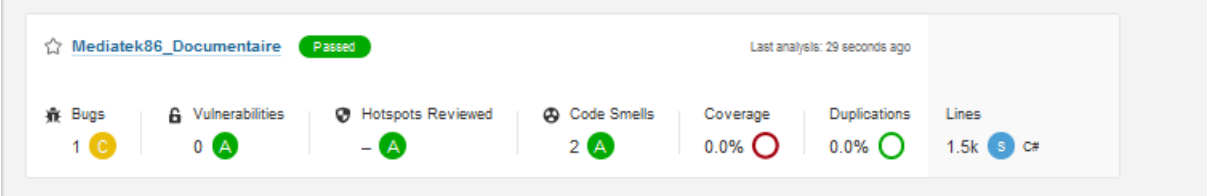



Figure 17: Analyse de l'application à l'Etat initial dans SonarQube

Une fois de retour dans **Visual Studio**, il est possible d'obtenir directement les informations de **sonarQube**. Pour cela il est nécessaire d'avoir installé les extensions nécessaires depuis l'onglet Extension présent dans l'IDE puis de choisir les extensions pour **SonarLint** et **SonarQube**. Dans la fenêtre **Team explorer,** en cliquant sur l'onglet de SonarQube, puis sur **connect** une demande d'authentification est demandé. Il convient alors de renseigner les informations de connexion au serveur de **SonarQube**. Puis la liste des projets existant sur le serveur apparaissent. Il suffit de double cliquer sur celui correspondant afin de le lier à Visual Studio.

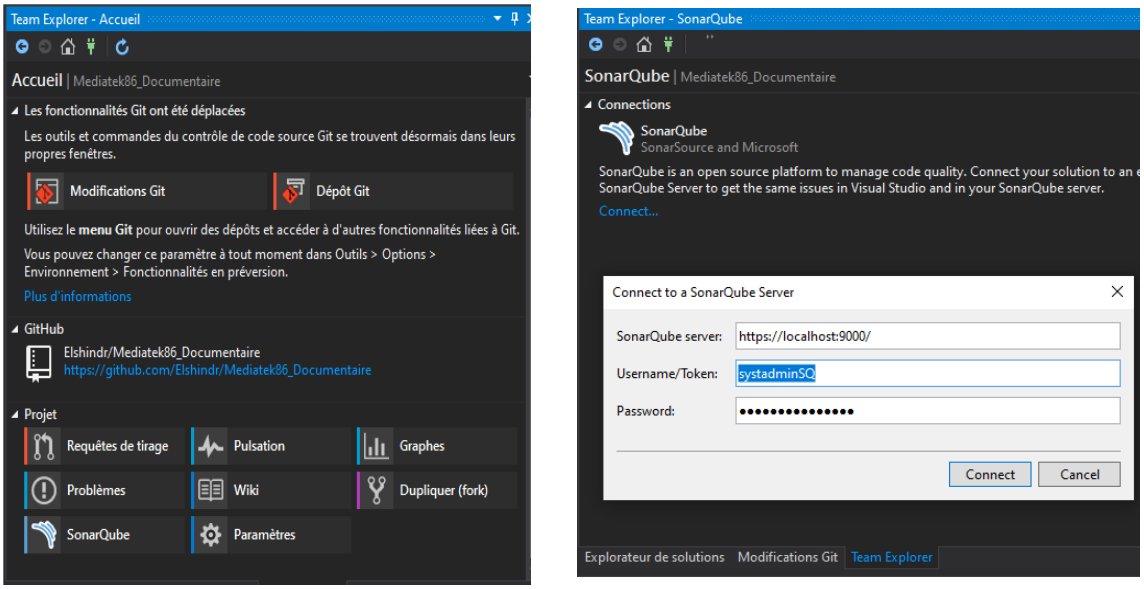



Figure 18: Fenêtre de Team explorer dans VisualStudio

## Gestion des documents *(optionnel, non implémenté pour l'instant)*

### Suivi de projet

Selon le tableau de suivi établi dans Trello illustré par la *figure* , nous nous situons dans la mission 1 qui consiste à gérer les fonctionnalités de type CRUD sur les documents. Il est possible de constater que les check-list de la mission 0 ont bien été rempli et les étiquettes sont passés de jaune à violet, indiquant ainsi leur fin de traitement.

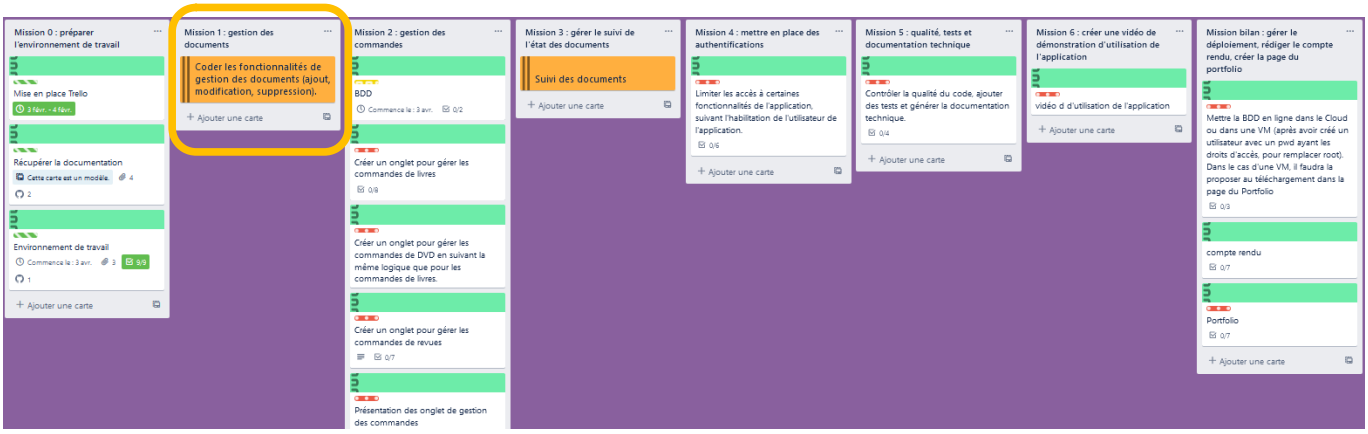



Figure 19: Tableau de suivi début de mission 1

### Modifications dans le contrôleur

### Modifications dans la vue

### Dépot Github

## Mission 2 : Gestion des commandes

### Suivi de projet

Selon le suivi de projet, nous nous situons au début de la mission 2 : gestion des commandes. Dans cette intention, la liste est composée de 5 cartes chacune correspondant à une étape différente, à savoir :

- Modifier la base de donnée en conséquence
- Création d'un onglet pour gérer les commandes des livres
- Création d'un onglet pour gérer les commandes des DVD
- Création d'un onglet pour gérer les commandes des revues
- Présentation des onglets de gestion des commandes et création de procédures dans la base de données

Les étiquettes des cartes de la **Mission 0** sont passées de jaune à vert indiquant ainsi la fin de ces étapes. De même, les différentes check-lists ont été mises à jours.

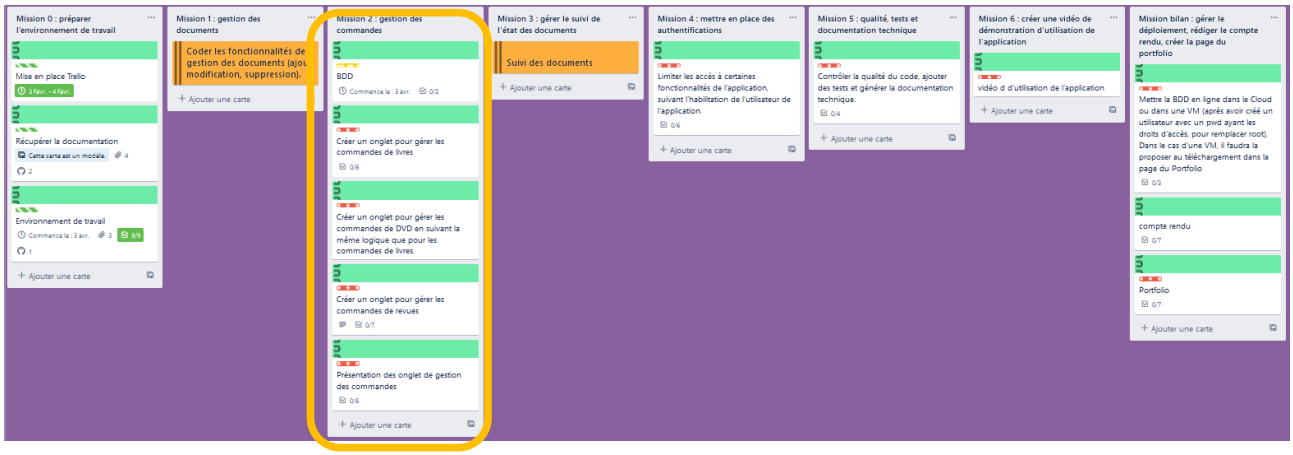



Figure 20: Suivi de mission Début de l'étape 2 gérer la fonctionnalité des commandes

Concernant à nouveau la mission 2, chaque carte comportent une check-list avec ses différentes étapes.


Figure 21: Liste des étapes de la mission 2

- **Gestion de la base de données**
  - Créer la table 'Suivi' qui contient les différentes étapes de suivi d'une commande de document de type livre ou dvd.
  - Relier cette table à CommandeDocument.

- **Créer un onglet pour gérer les commandes de livres**
  - Dans l'onglet, permettre la sélection d'un livre par son numéro, afficher les informations du livre ainsi que la liste des commandes, triée par date (ordre inverse de la chronologie).
  - La liste doit comporter les informations suivantes : date de la commande, montant, nombre d'exemplaires commandés et l'étape de suivi de la commande.
  - Créer un groupbox qui permet de saisir les informations d'une nouvelle commande et de l'enregistrer.
  - Lors de l'enregistrement de la commande, l'étape de suivi doit être mise à "en cours".
  - Permettre de modifier l'étape de suivi d'une commande en respectant certaines règles (une commande livrée ou réglée ne peut pas revenir à une étape précédente : en

cours ou relancée, une commande ne peut pas être réglée si elle n'est pas livrée).
    - Permettre de supprimer une commande uniquement si elle n'est pas encore livrée.
    - Créer le trigger qui se déclenche si une commande passe à l'étape "livrée" et qui crée autant de tuples dans la table "Exemplaire" que nécessaires, en valorisant la date d'achat avec la date de commande et en mettant l'état de l'exemplaire à "neuf".
    - Le numéro d'exemplaire doit être séquentiel par rapport au livre concerné.

- **Créer un onglet pour gérer les commandes de DVD**
    - Doit suivre  la même logique que pour les commandes de livres.

- **Créer un onglet pour gérer les commandes de revues**
    - dans le cas d'un nouvel abonnement, la revue sera préalablement créée dans l'onglet Revues. Donc, dans l'onglet des commandes de revues, il n'y a pas de distinction entre un nouvel abonnement et un renouvellement.
    - Permettre la sélection d'une revue par son numéro, afficher les informations de la revue ainsi que la liste des commandes (abonnements), triée par date (ordre inverse de la chronologie).
    - La liste doit comporter les informations suivantes : date de la commande, montant et date de fin d'abonnement.
    - Créer un groupbox qui permet de saisir les informations d'une nouvelle commande (nouvel abonnement ou renouvellement, le principe est identique) et de l'enregistrer.
    - Une commande de revue peut être supprimée, si aucun exemplaire n'est rattaché (donc, en vérifiant la date de l'exemplaire, comprise entre la date de la commande et la date de fin d'abonnement)
    - Pour cela, créer et utiliser la méthode 'ParutionDansAbonnement' qui reçoit en paramètre 3 dates (date commande, date fin abonnement, date parution) et qui retourne vrai si la date de parution est entre les 2 autres dates.
    - Créer le test unitaire sur cette méthode.

- **Présentation des onglets de gestion des commandes de revue**
    - La présentation de chaque onglet de gestion des commandes doit être similaire à l'onglet "Parutions des revues".
    - Dans toutes les listes, permettre le tri sur les colonnes.
    - 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 Commande

- Créer une procédure stockée qui permet d'obtenir la liste des revues dont l'abonnement se termine dans moins de 30 jours.
- Dès l'ouverture de l'application, ouvrir une petite fenêtre d'alerte rappelant la liste de ces revues (titre et date de fin abonnement) triée sur la date dans l'ordre chronologique.

## Gestion de la base de données

### Création de la table suivi

Pour cela, il faut se connecter à **phpMyAdmin**, se rendre dans la base de données mediatek86 puis dans l'onglet Concepteur afin de récupérer le schéma suivant :

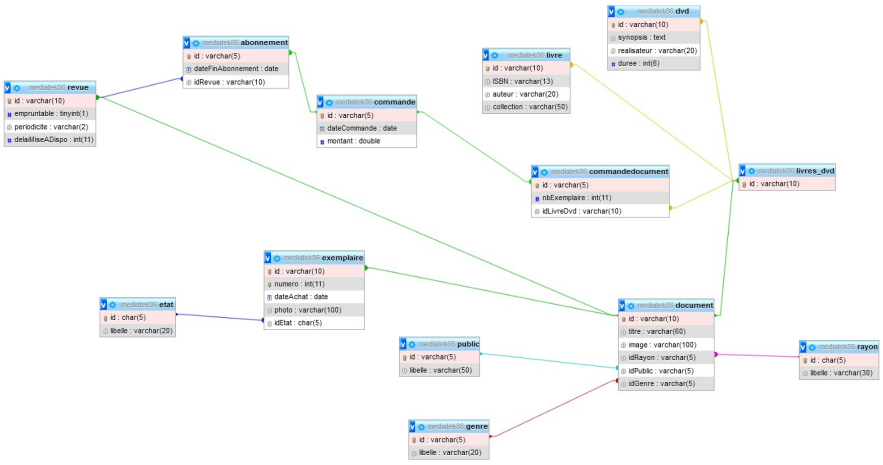



Figure 22: Schéma de la base mediatek86 du concepteur de phpMyAdmin

Puis dans l'onglet de SQL, la table suivi est créée de la manière suivante :

```sql
CREATE TABLE suivi (
  idSuivi INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
  label varchar(15) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Concernant les informations nécessaires à cette table, elle concerne les étapes de suivi d'une commande qui, selon l'analyse des besoins, doit avoir les statuts suivant : *en cours*, *livrée*, *réglée* et *relancé.* Ainsi, nous avons un id de type integer en tant que clé primaire de la table, puis un label de type (15)varchar, Enfin, dans ce même onglet SQL, la requête SQL exécutée afin de remplir la table est la suivante :

```
INSERT INTO suivi (label)
VALUES
        ('en cours'),
        ('livrée'),
        ('réglée'),
        ('relancé');
```

### Modification dans la table Commande

Afin de pouvoir récupérer les données de la table suivi, il est nécessaire de modifier la table commande en lui ajoutant une colonne supplémentaire avec les requêtes suivante s:

```
ALTER TABLE commande ADD idSuivi INTEGER NOT NULL

ALTER TABLE commande
  ADD CONSTRAINT commande_ibfk_1 FOREIGN KEY (idSuivi) REFERENCES suivi(idSuivi);
```

La première permet d'ajouter la colonne, la seconde d'ajouter idSuivi de la table suivi en tant que clé étrangère dans la table commande. Ainsi dans le concepteur, on peut alors obtenir la figure suivante :

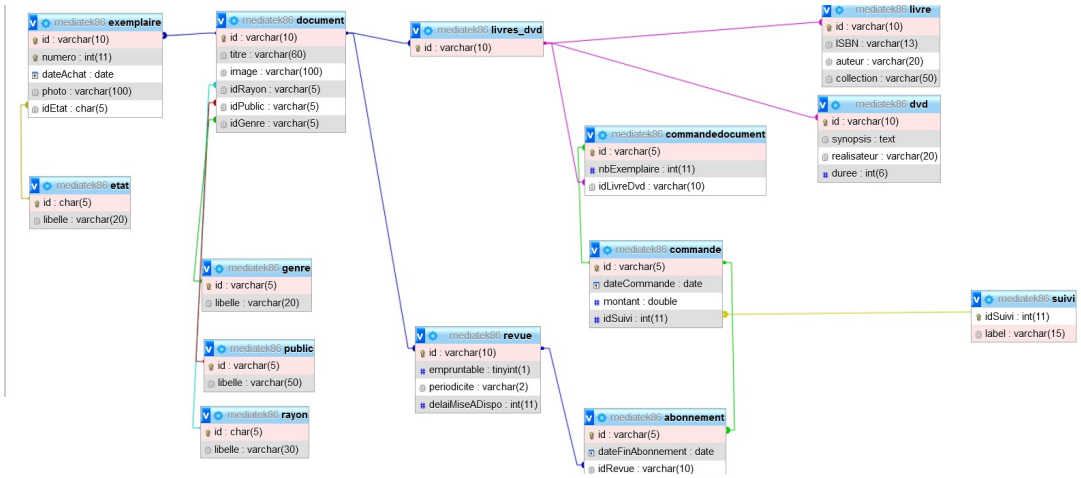



Figure 23: Schéma de la base mediatek86 avec les modifications

### Requête d'ajout de données tests dans la table CommandeDocuments

Ensuite la table est valorisée avec les valeurs de tests suivantes depuis l'onglet SQL de phpMyAdmin

```
INSERT INTO commandedocument(id, nbExemplaire, idLivreDvd)
VALUES
('00001', 3, '00017'),
('00002', 4, '00017'),
('00003', 12, '00017'),
('00004', 1, '00017'),
('00005', 5, '00004'),
('00006', 5, '00004'),
('00007', 67, '00004');
```

### Requête d'ajout de données tests dans la table Commande

Puis de même, dans la table commande

```
Insert INTO `commande`(id, dateCommande, montant, idSuivi)
VALUES
('00001', '2021-01-13', 1.99, 3),
('00002', '2021-12-23', 123.43, 4),
('00003', '2021-11-08', 123.43, 1),
('00004', '2021-11-23', 55.10, 2),
('00005', '2021-12-24', 23.3, 3),
('00006', '2021-01-24', 1.03, 1),
('00007', '2022-11-24', 1.43, 2);
```

### Création du trigger :Après une update dans commande si une commande est livrée

A la mise à jour d'un suivi de commande sur « livré », désormais un trigger, nommé **trigInsertCommandes**, se déclenche après la prise en compte de la validation du nouveau suivi. Il permet de créer autant de tuple dans la table exemplaire qu'il y a eu de nombre d'exemplaire du document commandé. Cette requête SQL a directement été intégré dans la SGBD de phpMyAdmin par l'onglet SQL.

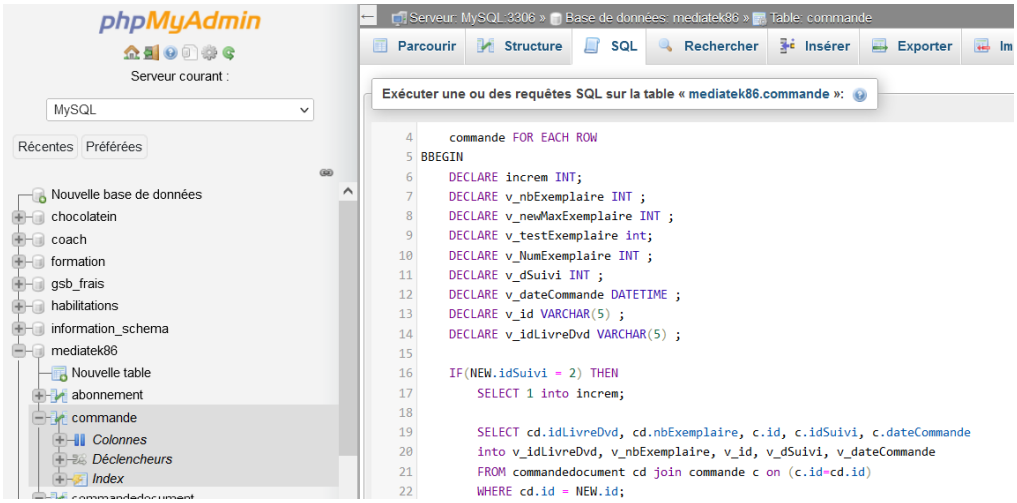



Figure 24: Exemple d'exécution de requête SQL dans phpMyAdmin

```
CREATE TRIGGER `trigInsertExemplrs` AFTER UPDATE ON `commande`
 FOR EACH ROW BEGIN
  DECLARE increm INT;
  DECLARE v_nbExemplaire INT ;
  DECLARE v_newMaxExemplaire INT ;
  DECLARE v_testExemplaire int;
  DECLARE v_NumExemplaire INT ;
  DECLARE v_dSuivi INT ;
  DECLARE v_dateCommande DATETIME ;
  DECLARE v_id VARCHAR(5) ;
  DECLARE v_idLivreDvd VARCHAR(5) ;

  IF(NEW.idSuivi = 2) THEN
    SELECT 1 into increm;

    SELECT cd.idLivreDvd, cd.nbExemplaire, c.id, c.idSuivi, c.dateCommande
    into v_idLivreDvd, v_nbExemplaire, v_id, v_dSuivi, v_dateCommande
    FROM commandedocument cd join commande c on (c.id=cd.id)
    WHERE cd.id = NEW.id;

    SELECT COUNT(numero) into v_newMaxExemplaire
    FROM exemplaire
    WHERE id = v_idLivreDvd;

   REPEAT

      INSERT INTO exemplaire(id, numero, dateAchat, photo, idEtat)
      VALUES ( v_idLivreDvd, v_newMaxExemplaire+increm, v_dateCommande,
"",'00001');
```

```
      SELECT (increm+1) into increm ;

    UNTIL (increm > v_nbExemplaire) END REPEAT ;

  END IF ;
END
```

## Mise en place des interfaces des gestion des commandes de Livres, de Dvd et de Revue

### Modifications pour la gestion de commande de livre et Dvd

Dans l'IDE, pour ajouter un nouvel onglet dans la vue frmMediatek, il faut se rendre dans le package vue, puis dans le designer **FrmMediatek.Designer.cs**. Après avoir sélectionné un onglet déjà présent, il est possible d'ajouter d'autres onglets depuis la fenêtre des propriétés de **tabOngletsApplication** qui est l'objet graphique représentant les onglets. Afin de modifier son texte ainsi que de déterminer son identifiant de code, il faut se rendre dans l'attribut **TabPage** qui ouvre une fenêtre permettant de gérer une collection d'onglets.

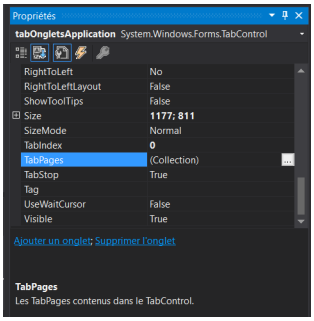



Figure 25: Fenêtre des propriétés de l'objet tabOngletsApplication

Une fois dans cette fenêtre, il est possible d'ajouter ou de supprimer d'autres onglets, ainsi que de saisir les attributs de **Text** et de **Name** qui correspondent réciproquement au titre de l'onglet et à son nom identifiant dans le code.

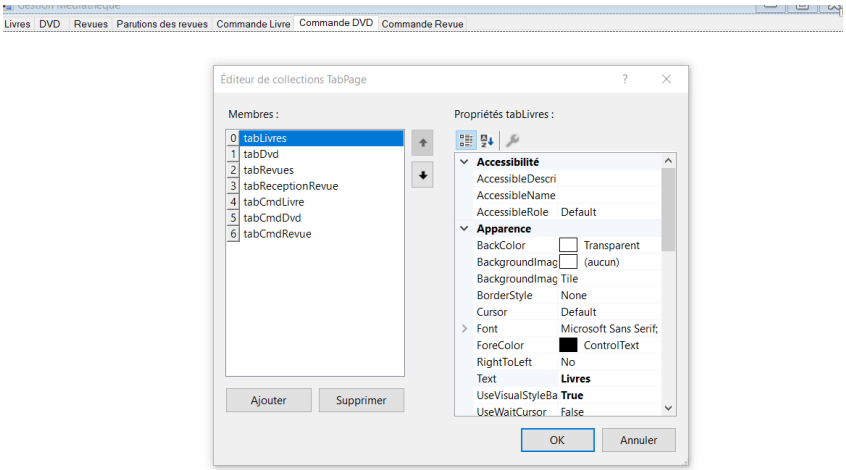


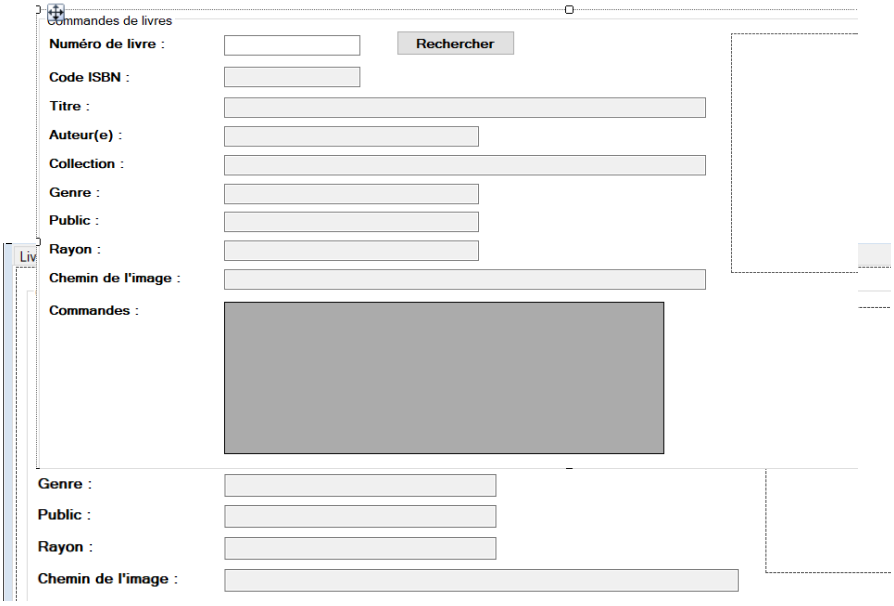

Figure 26: Propriétés de Collection de tabOngletsApplication et ses nouveaux onglets

**Étant donné que la mission actuelle demande la création des onglets pour le suivi de livre, de dvd et de revue, ces trois onglets ont été directement créés à cette étape**. Il est à noter que pour l'instant les autres propriétés de cet objet graphique n'ont pas à être modifiés afin de conserver la structure et l'aspect d'origine de l'application.

Afin de mettre en place la saisie du numéro du livre, un **label**, suivi d'une **TextBox** puis d'un **bouton** de Recherche ont été mis en place. Concernant l'affichage des informations d'un livre, l'ensemble du **groupbox** de l'onglet livre a été récupéré afin de préserver la structure. Seuls les propriétés **Name** des différents objets graphiques ont été modifié afin de s'adapter à l'onglet **Commande Livre**. Par exemple, si le nom **txbLivresTitre** correspond à l'onglet Livre son équivalent dans l'onglet **Commande de Livres** est **txbCmdLivresTitre**. Il en va de même pour les onglets Dvd.

Aussi, l'onglet de commande de livre contiendra des informations différentes des Dvd ou Revue, de part le code ISBN, l'auteur, la collection. Pour les Dvd, cela sera sur des informations sur les réalisateurs ou le synopsis. Enfin pour les Revues, les informations supplémentaires sont sur le délai de mise à dispo, sa périodicité et si il est emprutable ou non.

La mise en place d'un **DataGridView** permet d'afficher la liste des commandes d'un livre, ici nommé **dgvCmdLivresListe**. L'ensemble de ses paramètres sont laissés par défauts et identiques à ceux des autres DataGridView afin de respecter les normes de l'application




Par la suite, il est nécessaire d'ajouter des groupBox pour la création de commande , puis un second pour leur modification de statut. Pour le groupbox de création de commande deux objets NumericUpDown ont été ajouté, ainsi qu'un de type DatePicker. Réciproquement, ces objets graphiques permettront de récupérer les valeurs du nombre

d'exemplaire a commandé, du montant de la commande, puis de la date de cette commande. Enfin, un bouton de validation de commande avec une confirmation a été ajouté. Cette confirmation est là afin d'éviter des erreurs d'utilisations.

Pour le groupbox de modification de commande, des textBox en lecture seule permettant le rappels des différentes informations de la commande sélectionnée dans la datagridview ont été placé (IdCommande, IdLivre, NbExemplaire et Date de Commande). Puis un objet combobox permet de sélectionner le statut du suivi afin de changer le suivi de la commande. Enfin deux boutons, un de modification de suivi de commande et un de suppression de la commande ont été placé. Ces boutons sont aussi soumis à confirmation.


Finalement, pour les onglets Commandes de Livres et Commandes de Dvd nous avons l'affichage suivant :

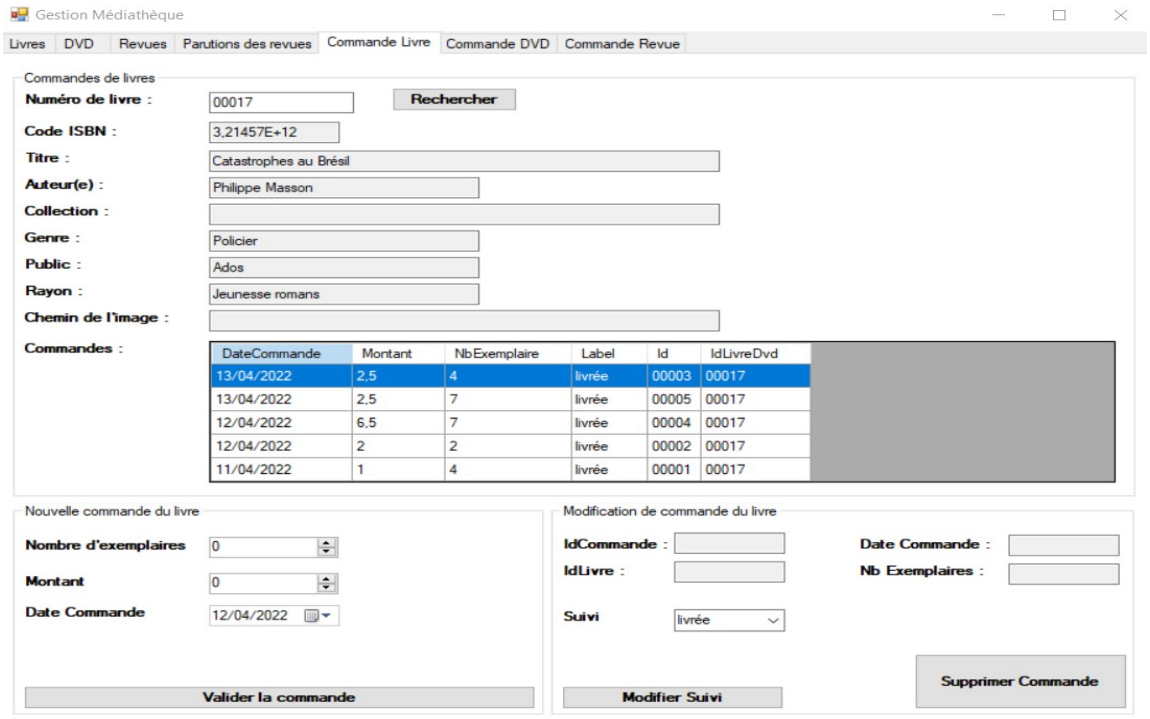
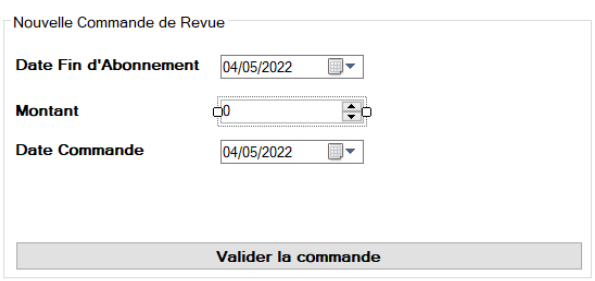


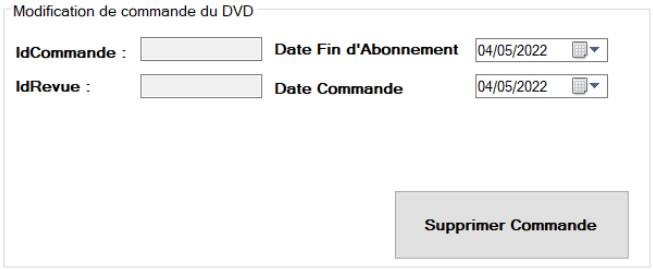

Figure 27: Vue de l'onglet Commande de livre après modifications

## Modifications pour la gestion de commande de Revue

Dans le troisième onglet créé, la mise en page principale est similaire a celle des Commandes de Dvd ou de Livres, à la différence que les informations dans les groubox récupérées pour les commandes sont différentes. De plus dans leur cas, on parle davantage d'Abonnement.

Ainsi, les objets graphiques pour la prise de commandes et sa modification sont différentes. Dans le cas de la prise de commande d'abonnement, deux DateTimePicker ont été placé afin de récupérer la date de commande et la date de fin d'abonnement, puis un NumericUpDown pour le montant et un bouton de validation. Pour le second groupbox, il s'agit d'un groupbox de suppression. De même que pour les onglets de commandes de livres et de Dvd, des textBox et des DatePicker de rappels des informations de l'abonnement sélectionné en lecture seul ont été placé. Enfin, un bouton de suppression d'abonnement a été mis en place avec confirmation.




## Création des méthodes pour l'affichage : Classe FrmMediatek

Le code complet de cette Classe est accessible à ce lien

A l'effigie du mode opératoire du code déjà présent, de nouvelles régions ont été ajouté dans le fichier **FrmMediatek.cs** afin de faciliter la lecture du code . Ces région sont nommées **LivresCommandes, DvdCommandes, RevuesCommandes.**

```
#region LivresCommandes

//------------------------------------------------------------
// ONGLET "COMMANDES D'UN LIVRE"
//------------------------------------------------------------
```

Figure 28: Exemple d'ajout de région

### BtnLivresCmdNumRecherche_Click()

La méthode de gestion de l'événement sur le clique du bouton de recherche permet de vérifier

Son code est le suivant :

```
/// <summary>
/// Evenement de clic sur le bouton de recherche
/// Permet l'affichage du livre dont le numéro est saisi.
/// Si non trouvé, affichage d'un MessageBox.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void BtnCmdLivresNumRecherche_Click(object sender, EventArgs e)
{
    if (!txbCmdLivresNumero.Text.Equals(""))
    {

        Livre livre = lesLivres.Find(x => x.Id.Equals(txbCmdLivresNumero.Text));

        if (livre != null)
        {
            AfficheCmdLivresInfos(livre);
        }
        else
        {
            MessageBox.Show("numéro introuvable");
            txbCmdLivresNumero.Text = "";
        }
```

Cette méthode appelle la méthode qui permet l'affichage des informations du livre recherché **AfficheCmdLivresInfos()** qui est la suivante :

```
/// <summary>
/// Methode d'affichage des informations du livre sélectionné dan
/// </summary>
/// <param name="livre"></param>
1 référence
private void AfficheCmdLivresInfos(Livre livre)
{
    txbCmdLivresAuteur.Text = livre.Auteur;
    txbCmdLivresCollection.Text = livre.Collection;
    pcbCmdLivresImage.Text = livre.Image;
    txbCmdLivresIsbn.Text = livre.Isbn;
    txbCmdLivresNumero.Text = livre.Id;
    txbCmdLivresGenre.Text = livre.Genre;
    txbCmdLivresPublic.Text = livre.Public;
    txbCmdLivresRayon.Text = livre.Rayon;
    txbCmdLivresTitre.Text = livre.Titre;
    string image = livre.Image;
    try
    {
        pcbCmdLivresImage.Image = Image.FromFile(image);
    }
    catch
    {
        pcbCmdLivresImage.Image = null;
    }
    string idDocument = txbCmdLivresNumero.Text;
    lesCmdLivre = controle.GetAllCommandesDocument(idDocument);
    RemplirCmdLivresListe(lesCmdLivre);
    AccesNewCmdLivresGroupBox(true);
}
```

En fin de méthode, un appel vers le controleur est effectué afin de récupérer les informations depuis la base de données par l'appel GetAllCommandesDocument() où l'identifiant du livre, idDocument, est injecté afin de servir de filtre dans la requête SQL de sélection. Puis, un appel vers la méthode **RemplirCmdLivresListe()** permet l'affichage

des informations du livre recherché dans le **DataGridView** de l'onglet de commande de livre qui est la suivante :

```
/// <summary>
/// Methode qui remplit le datagridview avec la liste des commandes de livre reçue en paramètre
/// </summary>
/// <param name="commandes">Liste des commandes du document</param>
6 références
private void RemplirCmdLivresListe(List<CommandeDocument> commandes)
{
    bdgLivreCmdListe.DataSource = commandes;
    dgvCmdLivresListe.DataSource = bdgLivreCmdListe;
    dgvCmdLivresListe.Columns["idCommande"].Visible = false;
    dgvCmdLivresListe.Columns["idSuivi"].Visible = false;
    dgvCmdLivresListe.Columns["idLivreDvd"].Visible = false;

    dgvCmdLivresListe.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
    dgvCmdLivresListe.Columns["dateCommande"].DisplayIndex = 0;
    dgvCmdLivresListe.Columns["montant"].DisplayIndex = 1;
    dgvCmdLivresListe.Columns["nbExemplaire"].DisplayIndex = 2;
    dgvCmdLivresListe.Columns["label"].DisplayIndex = 3;

}
```

Cette liste est alimentée par la propriété **DataSource** qui ici a la valeur de la liste des commandes du livre concerné. Afin de lier la liste des commandes nommée commandes et celle de l'objet graphique DataGridView nommé **dgvLivresCmdListe,** il est nécessaire de mettre en place un objet **BindingSource,** ici nommée **bdgLivresCmdListe**, comme le montre le code suivant placé dans les déclarations de variables globales.

```
private readonly BindingSource bdgLivresCmdListe = new BindingSource();
private List<Commande> lesCmdLivre = new List<Commande>();
```

### BtnNewCmdLivresValider_Click

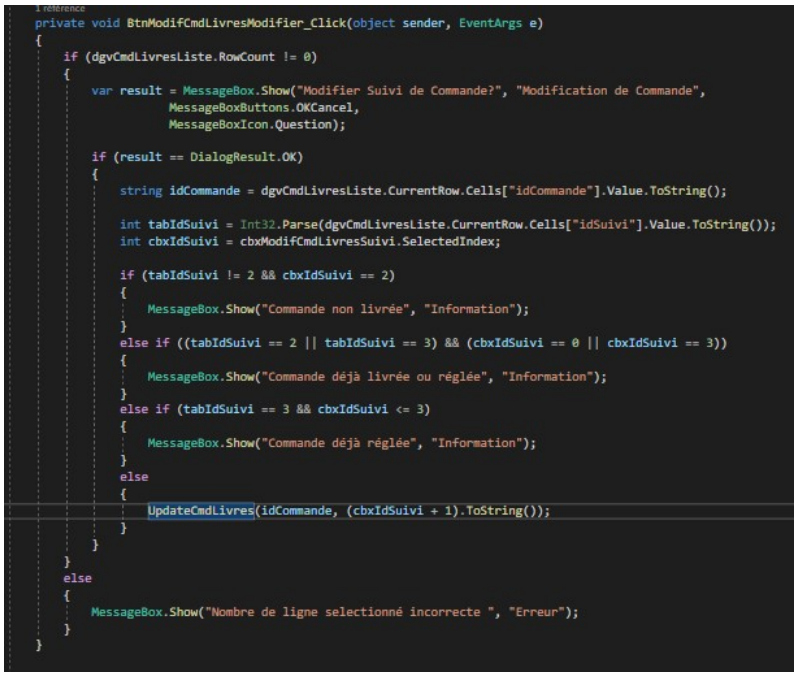
Cette méthode se déclenche sur le clic du bouton Valider Commande. Elle récupère les informations de la vue dans les onglets de commande de livre. Au clic une première vérification se fait sur les valeurs du nombre d'exemplaire et du montant qui doivent être différents de 0. Puis, un message de confirmation de type OKCANCEL de commande apparaît, permettant de cliquer sur ok pour confirmer la commande.

Ensuite, les informations des objets de commandes sont récupérés afin d'instancier puis de valoriser un objet **CommandeDocument**. L'identifiant de la nouvelle commande est valorisé depuis une méthode appelant la méthode **GetLastIdCommande** qui demande à la base de données l'identifiant Max de la table commande puis lui ajoute 1. Puis, la méthode Controle.CréerCommande() est appellée afin de lancer une requete d'insert dans la base de donnée, lorsque celle ci réussie, la liste des commandes de livres est remise à jour avec la méthode contrôle.GetAllCommandesDocument et les valeurs des objets graphiques sont vidés pour être rempli avec les nouvelles informations.

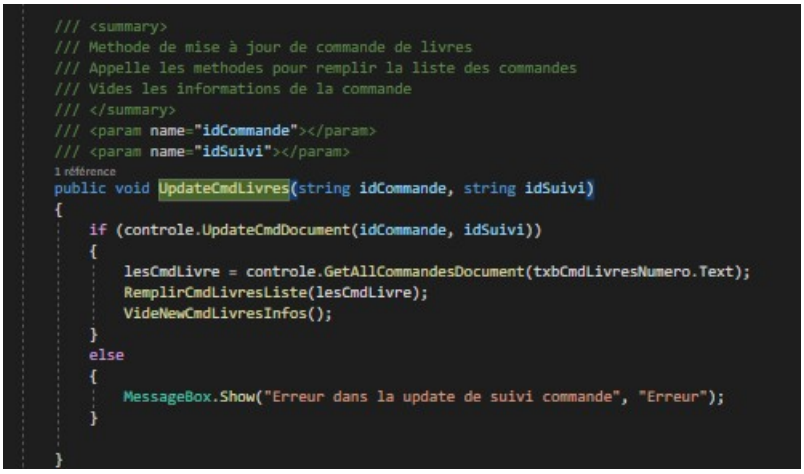
```
private void BtnNewCmdLivresValider_Click(object sender, EventArgs e)
{
    if (numNewCmdLivresMontant.Value != 0 && numNewCmdLivresNbExemplaire.Value != 0)
    {
        var result = MessageBox.Show("Valider Commande?", "Confirmation de Commande",
                 MessageBoxButtons.OKCancel,
                 MessageBoxIcon.Question);

        if (result == DialogResult.OK)
        {
            try
            {
                string strId = "00000";
                string strMaxId = "";
                if (controle.GetLastIdCommande().ToString() == null)
                {
                    strMaxId = "1";
                }
                else
                {
                    strMaxId = controle.GetLastIdCommande().ToString();
                }


                int lenMaxId = strMaxId.Length;

                string idCommande = strId.Remove(0, lenMaxId) + strMaxId;
                string idLivre = txbCmdLivresNumero.Text;

                DateTime dateCommande = dtpNewCmdLivresDate.Value;
                int nbExemplaire = (int)numNewCmdLivresNbExemplaire.Value;
                double montant = (double)numNewCmdLivresMontant.Value;

                string idDocument = txbCmdLivresNumero.Text;
                CommandeDocument commande = new CommandeDocument(idCommande, dateCommande, montant, "1", "en cours", idLivre, nbExemplaire);

                if (controle.CreerCommande(commande))
                {
                    lesCmdLivre = controle.GetAllCommandesDocument(idDocument);
                    RemplirCmdLivresListe(lesCmdLivre);
                    VideNewCmdLivresInfos();
                }
                else
                {
                    MessageBox.Show("Erreur dans la creation de commande", "Erreur");
                }
            }
            catch (Exception ex)
```

### BtnModifCmdLivresModifier_Click

Cette méthode se déclenche au clic du bouton modifier commande. Après vérification qu'une commande est bien sélectionnées, un message de confirmation OKCANCEL apparaît. Enfin, le suivi peut être modifié en fonction de son statut actuel. Une commande livre et réglée ne peuvent retourner au statut en cours ou relancé . Selon cela, le suivi de commande est mis à jour via l'appel de la méthode **UpdatecmdLivres** qui appelle la méthode **contrôle.UpdateCmdDocument** permettant de lancer une requête **update** dans la base de données. En fonction de son succès, les objets graphiques sont remis à jours puis valorisé en conséquence.



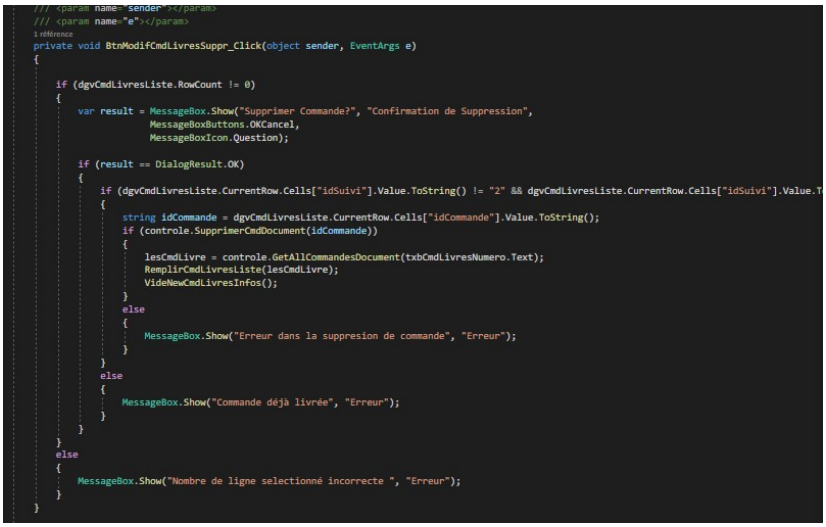
```
1 référence
private void BtnModifCmdLivresModifier_Click(object sender, EventArgs e)
{
    if (dgvCmdLivresListe.RowCount != 0)
    {
        var result = MessageBox.Show("Modifier Suivi de Commande?", "Modification de Commande",
                  MessageBoxButtons.OKCancel,
                  MessageBoxIcon.Question);

        if (result == DialogResult.OK)
        {
            string idCommande = dgvCmdLivresListe.CurrentRow.Cells["idCommande"].Value.ToString();

            int tabIdSuivi = Int32.Parse(dgvCmdLivresListe.CurrentRow.Cells["idSuivi"].Value.ToString());
            int cbxIdSuivi = cbxModifCmdLivresSuivi.SelectedIndex;

            if (tabIdSuivi != 2 && cbxIdSuivi == 2)
            {
                MessageBox.Show("Commande non livrée", "Information");
            }
            else if ((tabIdSuivi == 2 || tabIdSuivi == 3) && (cbxIdSuivi == 0 || cbxIdSuivi == 3))
            {
                MessageBox.Show("Commande déjà livrée ou réglée", "Information");
            }
            else if (tabIdSuivi == 3 && cbxIdSuivi <= 3)
            {
                MessageBox.Show("Commande déjà réglée", "Information");
            }
            else
            {
                UpdateCmdLivres(idCommande, (cbxIdSuivi + 1).ToString());
            }
        }
    }
    else
    {
        MessageBox.Show("Nombre de ligne selectionné incorrecte ", "Erreur");
    }
}
```



```
/// <summary>
/// Methode de mise à jour de commande de livres
/// Appelle les methodes pour remplir la liste des commandes
/// Vides les informations de la commande
/// </summary>
/// <param name="idCommande"></param>
/// <param name="idSuivi"></param>
1 référence
public void UpdateCmdLivres(string idCommande, string idSuivi)
{
    if (controle.UpdateCmdDocument(idCommande, idSuivi))
    {
        lesCmdLivre = controle.GetAllCommandesDocument(txbCmdLivresNumero.Text);
        RemplirCmdLivresListe(lesCmdLivre);
        VideNewCmdLivresInfos();
    }
    else
    {
        MessageBox.Show("Erreur dans la update de suivi commande", "Erreur");
    }

}
```

### BtnModifCmdLivresSuppr_Click

Cette méthode se déclenche sur le clic du bouton supprimer une commande. Après un message de confirmation et vérification qu'une ligne de commande est bien sélectionnée, les informations des objets graphiques de la commande en question sont récupérés, notamment son identifiant de commande qui permettra son identification dans la base. Les autres valeurs récupérées sont le statut du suivi et l'identifiant de suivi qui permettent de vérifier si la commande est déjà livrée, réglée ou non. En fonction de cela, la commande ne peut être supprimée.



```
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void BtnModifCmdLivresSuppr_Click(object sender, EventArgs e)
{

    if (dgvCmdLivresListe.RowCount != 0)
    {
        var result = MessageBox.Show("Supprimer Commande?", "Confirmation de Suppression",
                     MessageBoxButtons.OKCancel,
                     MessageBoxIcon.Question);

        if (result == DialogResult.OK)
        {
            if (dgvCmdLivresListe.CurrentRow.Cells["idSuivi"].Value.ToString() != "2" && dgvCmdLivresListe.CurrentRow.Cells["idSuivi"].Value.T
            {
                string idCommande = dgvCmdLivresListe.CurrentRow.Cells["idCommande"].Value.ToString();
                if (controle.SupprimerCmdDocument(idCommande))
                {
                    lesCmdLivre = controle.GetAllCommandesDocument(txbCmdLivresNumero.Text);
                    RemplirCmdLivresListe(lesCmdLivre);
                    VideNewCmdLivresInfos();
                }
                else
                {
                    MessageBox.Show("Erreur dans la suppresion de commande", "Erreur");
                }
            }
            else
            {
                MessageBox.Show("Commande déjà livrée", "Erreur");
            }
        }
    }
    else
    {
        MessageBox.Show("Nombre de ligne selectionné incorrecte ", "Erreur");
    }
}
```

## Autres méthodes utilitaires

D'autres méthodes ont été ajouté. Il s'agit de méthode de gestion de l'affichage notamment afin de mettre les informations des textbox, combobox et datagridview à jour suite aux modifications.

```
/// <summary>
/// Methode qui permet ou bloque l'accès à la gestion de prise de commandes de livres
/// et vide les objets graphiques
/// </summary>
/// <param name="acces"></param>
3 références
private void AccesNewCmdLivresGroupBox(bool acces)
{
    VideNewCmdLivresInfos();
    gpbNewCmdLivres.Enabled = acces;
    gpbModifCmdLivres.Enabled = acces;
}


/// <summary>
/// Methode qui vide les zones d'affichage des informations du bloc de commandes de livres
/// </summary>
8 références
private void VideNewCmdLivresInfos()
{
    numNewCmdLivresNbExemplaire.Value = 0;
    numNewCmdLivresMontant.Value = 0;
    dtpNewCmdLivresDate.Value = DateTime.Now;
}
```

```
/// <summary>
/// Methode qui vide les zones d'affichage des informations du bloc de livre à commander
/// Vide le datagridview associé
/// Bloque l'accès aux prises de commandes.
/// </summary>
2 références
private void VideCmdLivresInfos()
{
    txbCmdLivresNumero.Text = "";
    txbCmdLivresIsbn.Text = "";
    txbCmdLivresTitre.Text = "";
    txbCmdLivresAuteur.Text = "";
    txbCmdLivresCollection.Text = "";
    txbCmdLivresGenre.Text = "";
    txbCmdLivresPublic.Text = "";
    txbCmdLivresRayon.Text = "";
    txbCmdLivresImage.Text = "";
    pcbCmdLivresImage.Image = null;

    // Vide le datagridview
    lesCmdLivre = new List<CommandeDocument>();
    RemplirCmdLivresListe(lesCmdLivre);
}
```

Aussi, une méthode pour le tri des colonnes dans les différents datagridview a été ajouté

```
/// <summary>
/// Evenement sur le clic d'une colonne de la liste
/// - Tri la liste selon l'entete de colonne cliquée
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void DgvCmdLivresListe_ColumnHeaderMouseClick(object sender, DataGridViewCellMouseEventArgs e)
{

    VideNewCmdLivresInfos();
    string titreColonne = dgvCmdLivresListe.Columns[e.ColumnIndex].HeaderText;
    List<CommandeDocument> sortedList = new List<CommandeDocument>();
    switch (titreColonne)
    {
        case "Id":
            sortedList = lesCmdLivre.OrderBy(o => o.IdCommande).ToList();
            break;
        case "DateCommande":
            sortedList = lesCmdLivre.OrderBy(o => o.DateCommande).ToList();
            break;
        case "Montant":
            sortedList = lesCmdLivre.OrderBy(o => o.Montant).ToList();
            break;
        case "NbExemplaire":
            sortedList = lesCmdLivre.OrderBy(o => o.NbExemplaire).ToList();
            break;
        case "Label":
            sortedList = lesCmdLivre.OrderBy(o => o.Label).ToList();
            break;

    }
    RemplirCmdLivresListe(sortedList);

}
```

**Il est à noter que pour l'ensemble de ces méthodes concernant les objets Livres, les mêmes méthode ont été créé pour la gestion des objets Dvd et de Revue.**

## Création de Classes dans le package metier

Création des classes **Suivi** qui hérite de la Classe **Catégorie,** puis des Classes **Abonnement** et **CommandesDocuments** qui hérite de la nouvelle classe **Commande**

### Dans Commande.cs

```
public class Commande
{

    /// <summary>
    /// Constructeur de la Classe mere Commande
    /// </summary>
    /// <param name="idCommande"></param>
    /// <param name="dateCommande"></param>
    /// <param name="montant"></param>
    /// <param name="idSuivi"></param>
    /// <param name="label"></param>
    2 références
    public Commande(string idCommande, DateTime dateCommande, double montant, string idSuivi, string label)
    {
        this.IdCommande = idCommande;
        this.DateCommande = dateCommande;
        this.Montant = montant;
        this.IdSuivi = idSuivi;
        this.Label = label;
    }

    /// <summary>
    /// Getter et Setter de la propriété IdCommande autogénérés
    /// </summary>
    20 références
    public string IdCommande { get; set; }

    /// <summary>
    /// Getter et Setter de la propriété DateCommande autogénérés
    /// </summary>
    7 références
    public DateTime DateCommande { get; set; }

    /// <summary>
    /// Getter et Setter de la propriété Montant autogénérés
    /// </summary>
    11 références
    public double Montant { get; set; }
```

Son code source complet est visible a ce lien

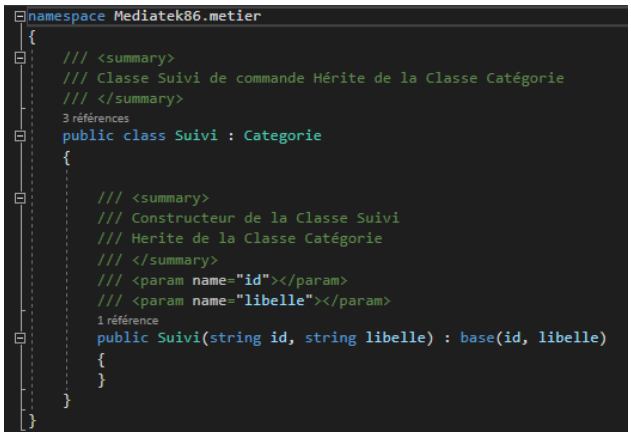
### Dans CommandeDocument.cs

Cette Classe hérite de la Classe **Commande**, ainsi elle récupère ses propriétés filles partagées, ainsi cette Classe Fille possède un identifiant de document, **IdLivreDvd** et un nombre d'exemplaire, **NbExemplaire**. Son code source complet est visible a ce lien

```
public class CommandeDocument : Commande
{

    /// <summary>
    /// Constructeur de la classe CommandeDocument
    /// Herite des propriétés de la classe Commande
    /// </summary>
    /// <param name="idCommande"></param>
    /// <param name="dateCommande"></param>
    /// <param name="montant"></param>
    /// <param name="idSuivi"></param>
    /// <param name="label"></param>
    /// <param name="idLivreDvd"></param>
    /// <param name="nbExemplaire"></param>
    4 références
    public CommandeDocument(string idCommande, DateTime dateCommande, double montant,
        string idSuivi, string label, string idLivreDvd, int nbExemplaire)
        : base(idCommande, dateCommande, montant, idSuivi, label)
    {
        this.IdLivreDvd = idLivreDvd;
        this.NbExemplaire = nbExemplaire;
    }
    /// <summary>
    /// Getter et Setter de la propriété IdLivreDvd autogénérés
    /// </summary>
    5 références
    public string IdLivreDvd { get; set; }

    /// <summary>
    /// Getter et Setter de la propriété NbExemplaire autogénérés
    /// </summary>
    7 références
    public int NbExemplaire { get; set; }
```

### Suivi.cs

Cette Classe hérite de la Classe **Catégorie**, ainsi elle récupère ses propriétés filles partagées qui sont un identifiant et son libelle. Son code source complet est visible a ce lien.



```
namespace Mediatek86.metier
{
    /// <summary>
    /// Classe Suivi de commande Hérite de la Classe Catégorie
    /// </summary>
    3 références
    public class Suivi : Categorie
    {

        /// <summary>
        /// Constructeur de la Classe Suivi
        /// Herite de la Classe Catégorie
        /// </summary>
        /// <param name="id"></param>
        /// <param name="libelle"></param>
        1 référence
        public Suivi(string id, string libelle) : base(id, libelle)
        {
        }
    }
}
```

### Abonnement.cs

Cette Classe hérite de la Classe **Commande**, ainsi elle récupère ses propriétés filles partagées. Aussi, cette Classe Fille possède un identifiant de document, **IdRevue** et une date de fin d'abonnement, **DateFinAbonnement**. Son code source complet est visible a ce lien.

```
public class Abonnement : Commande
{

    /// <summary>
    /// Constructeur de la classe Abonnement
    /// </summary>
    /// <param name="idCommande"></param>
    /// <param name="dateCommande"></param>
    /// <param name="montant"></param>
    /// <param name="idSuivi"></param>
    /// <param name="label"></param>
    /// <param name="idRevue"></param>
    /// <param name="dateFinAbonnement"></param>


    public Abonnement(string idCommande, DateTime dateCommande, double montant, string idSuivi, string label, string idRevue, DateTime dateFinAbonnement) :
        base(idCommande, dateCommande, montant, idSuivi, label)
    {
        this.IdRevue = idRevue;
        this.DateFinAbonnement = dateFinAbonnement;
    }


    /// <summary>
    /// Getter et Setter de la propriété idRevue autogénérés
    /// </summary>

    public string IdRevue { get; set; }

    /// <summary>
    /// Getter et Setter de la propriété dateFinAbonnement autogénérés
    /// </summary>

    public DateTime DateFinAbonnement { get; set; }
}
```

## Ajout des méthodes dans les Classes Dao et Controle

De la même manière que dans la Classe **FrmMediatek**, des régions ont été ajouté afin de facilité la lecture du code dans la Classe **Dao**. Aussi, les méthodes ajoutées dans cette Classe qui traitent les objets Livre et Dvd sont gérés par les mêmes méthodes. Concernant celles des Abonnements, les méthodes sont différentes mais possède la même trame, c'est pourquoi l'ajout de ces méthodes seront éclipsés dans ce rapport de par leur similarité. Les codes sources de ces deux classes sont accessibles aux liens suivants :

Dao et Controle.

### GetAllCommandes____

```
public static List<CommandeDocument> GetAllCommandesDocument(string idDocument)
{

    List<CommandeDocument> lesCmdDoc = new List<CommandeDocument>();

    string req = "Select c.id, c.dateCommande, c.montant, c.idSuivi, cd.nbExemplaire,
    req += " from commande c join suivi s on c.idSuivi = s.idSuivi";
    req += " join commandedocument cd on c.id = cd.id";
    req += " where cd.idLivreDvd = @iddoc";
    req += " order by c.dateCommande DESC;";


    Dictionary<string, object> parameters = new Dictionary<string, object>
     {
        { "@iddoc", idDocument}
     };

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


    while (curs.Read())
    {
        string idCommande = (string)curs.Field("id");
        DateTime dateCommande = (DateTime)curs.Field("dateCommande");
        double montant = (double)curs.Field("montant");

        int idSuivi = (int)curs.Field("idSuivi");
        string label = (string)curs.Field("label");
        string idLivreDvd = (string)curs.Field("idLivreDvd");
        int nbExemplaire = (int)curs.Field("nbExemplaire");

        CommandeDocument commande = new CommandeDocument(idCommande, dateCommande, mon
        lesCmdDoc.Add(commande);
    }
    curs.Close();
```

```
public static List<Abonnement> GetAllAbonnementRevue(string idRevue)
{
    List<Abonnement> lesAboRevue = new List<Abonnement>();
    string req = "Select c.id, c.dateCommande, c.montant, c.idSuivi, a.dateFinAbonnement, a.idRevue, s.label";
    req += " from commande c join suivi s on c.idSuivi = s.idSuivi";
    req += " join abonnement a on c.id = a.id";
    req += " where a.idRevue = @idRevue";
    req += " order by c.dateCommande DESC;";


    Dictionary<string, object> parameters = new Dictionary<string, object>
     {
        { "@idRevue", idRevue}
     };

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


    while (curs.Read())
    {
        string idCommande = (string)curs.Field("id");
        DateTime dateCommande = (DateTime)curs.Field("dateCommande");
        double montant = (double)curs.Field("montant");
        int idSuivi = (int)curs.Field("idSuivi");
        string label = (string)curs.Field("label");

        string idrevue = (string)curs.Field("idLivreDvd");
        DateTime dateFinAbonnement = (DateTime)curs.Field("dateFinAbonnement");

        Abonnement abonnement = new Abonnement(idCommande, dateCommande, montant, idSuivi.ToString(), label, idrevue, dateFinAbon
        lesAboRevue.Add(abonnement);
    }
    curs.Close();

    return lesAboRevue;
}
```

Cette Classe permet de créer une requete Select puis de lancer cette requête SQL exécutées par la Classe BddMySql. Ainsi ces nouvelles méthodes permettent de récupérer toute les commandes selon l'identifiant d'un document ou d'une revue. Aussi, la méthode de gauche est fonctionnelle autant pour les commandes de livre que de Dvd puisque leur Classe métier respective hérite de la Classe LivreDvd.

La méthode suivante permet de faire le lien entre les méthodes appelées dans la vue et celles de la classe Dao qui gère les requêtes SQL

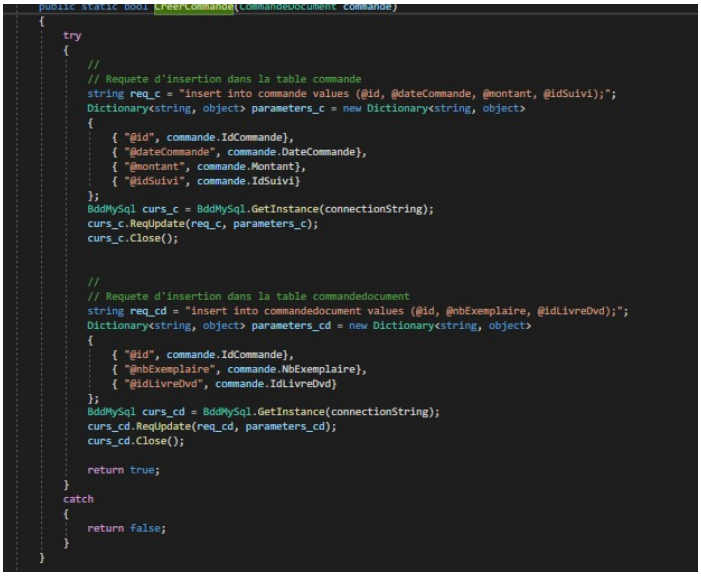
```
/// <summary>
/// Methode du controleur accedant à la méthode GetAllCommandesDocument
/// Permet la récupération de la liste de commande d'un livre ou Dvd
/// </summary>
/// <returns>Collection d'objets de Livre ou de Dvd</returns>
9 références
public List<CommandeDocument> GetAllCommandesDocument(string idDocument)
{
    return Dao.GetAllCommandesDocument(idDocument);
}
```

```
/// <summary>
/// Methode du controleur accédant à la méthode GetAllCommandesRevue
/// Permet la récupération de la liste des abonnements d'une revue
/// </summary>
/// <param name="idRevue">Identifiant d'une revue</param>
/// <returns>La liste des abonnements d'une revue</returns>
3 références
public List<Abonnement> GetAllAbonnemmentRevue(string idRevue)
{
    return Dao.GetAllAbonnemmentRevue(idRevue);
}
```

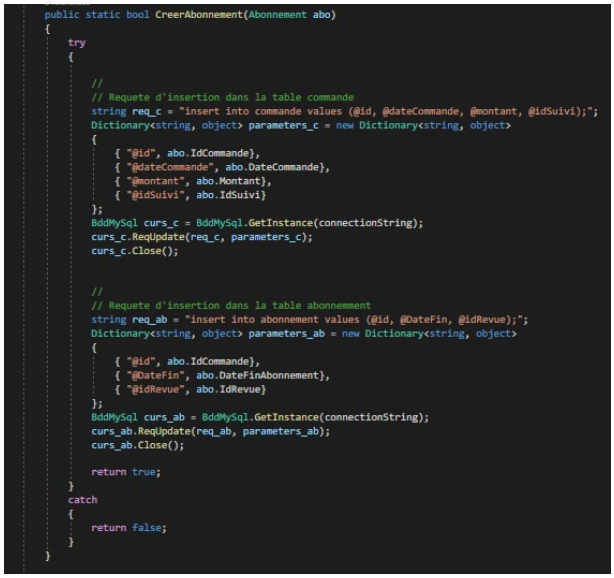
## CreerCommande & CreerAbonnement

*Création des méthodes*

Ces méthodes permettent de lancer une requête SQL afin d'insérer dans la base de données une nouvelle ligne de commande d'un Livre, d'un Dvd, ou d'une revue. Par le biais d'un objet de type Dictionnaire, les différentes valeurs de la commande sont insérés en paramètres(l'identifiant de la commande, sa date, son montant et son suivi), puis un appel vers la Classe BddMySql permet d'initialiser un curseur qui recevra l'instance de cette Classe, puis un appel avec sa méthode ReqUpdate permet l'insertion dans la table commande. De la même manière, un second insert doit s'effectuer dans la table commandedocument qui recevra les valeurs d'id de la commande, le nombre d'exemplaire, l'identifiant du document physique, ou alors vers la table abonnement dans le cas d'une revue afin d'enregistrer ces identifiants de commandes, sa date de fin d'abonnement et l'identifiant de la revue

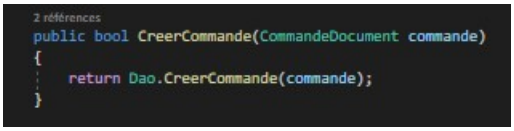


```
public static bool CreerCommande(CommandeDocument commande)
{
    try
    {
        //
        // Requete d'insertion dans la table commande
        string req_c = "insert into commande values (@id, @dateCommande, @montant, @idSuivi);";
        Dictionary<string, object> parameters_c = new Dictionary<string, object>
        {
            { "@id", commande.IdCommande},
            { "@dateCommande", commande.DateCommande},
            { "@montant", commande.Montant},
            { "@idSuivi", commande.IdSuivi}
        };
        BddMySql curs_c = BddMySql.GetInstance(connectionString);
        curs_c.ReqUpdate(req_c, parameters_c);
        curs_c.Close();


        //
        // Requete d'insertion dans la table commandedocument
        string req_cd = "insert into commandedocument values (@id, @nbExemplaire, @idLivreDvd);";
        Dictionary<string, object> parameters_cd = new Dictionary<string, object>
        {
            { "@id", commande.IdCommande},
            { "@nbExemplaire", commande.NbExemplaire},
            { "@idLivreDvd", commande.IdLivreDvd}
        };
        BddMySql curs_cd = BddMySql.GetInstance(connectionString);
        curs_cd.ReqUpdate(req_cd, parameters_cd);
        curs_cd.Close();

        return true;
    }
    catch
    {
        return false;
    }
}
```

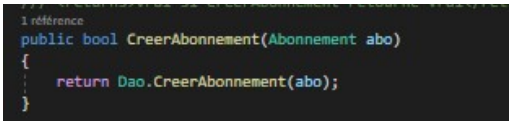


```
public static bool CreerAbonnement(Abonnement abo)
{
    try
    {

        //
        // Requete d'insertion dans la table commande
        string req_c = "insert into commande values (@id, @dateCommande, @montant, @idSuivi);";
        Dictionary<string, object> parameters_c = new Dictionary<string, object>
        {
            { "@id", abo.IdCommande},
            { "@dateCommande", abo.DateCommande},
            { "@montant", abo.Montant},
            { "@idSuivi", abo.IdSuivi}
        };
        BddMySql curs_c = BddMySql.GetInstance(connectionString);
        curs_c.ReqUpdate(req_c, parameters_c);
        curs_c.Close();


        //
        // Requete d'insertion dans la table abonnemment
        string req_ab = "insert into abonnement values (@id, @DateFin, @idRevue);";
        Dictionary<string, object> parameters_ab = new Dictionary<string, object>
        {
            { "@id", abo.IdCommande},
            { "@DateFin", abo.DateFinAbonnement},
            { "@idRevue", abo.IdRevue}
        };
        BddMySql curs_ab = BddMySql.GetInstance(connectionString);
        curs_ab.ReqUpdate(req_ab, parameters_ab);
        curs_ab.Close();

        return true;
    }
    catch
    {
        return false;
    }
}
```

De la même manière, cette méthode peut être appelée depuis la classe Controle.



```
2 références
public bool CreerCommande(CommandeDocument commande)
{
    return Dao.CreerCommande(commande);
}
```



```
1 référence
public bool CreerAbonnement(Abonnement abo)
{
    return Dao.CreerAbonnement(abo);
}
```

*Création d'un trigger de contrôle de partition sur héritage de commande*

La partition sur héritage de commande implique qu'une commande est, soit une commandedocument soit un abonnement, pas les deux. Ainsi le code suivant est le trigger créé afin de mettre en place cette vérification depuis l'onglet de requête SQL de phpMyAdmin. Celui-ci se déclenche à chaque insert dans la table commande, puis vérifie le nombre de fois que l'identifiant dans l'insert apparaît dans les tables commandedocument et abonnement. Si cet identifiant apparaît au moins une fois dans la ces tables alors l'insert est rejetée.

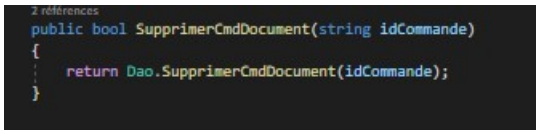
```
CREATE TRIGGER `trigInsertCommandes`
BEFORE INSERT ON `commande` FOR EACH ROW
BEGIN
  DECLARE idCmdDoc INT;
  DECLARE idCmdRevue INT;

  SELECT Count(id) into idCmdDoc FROM commandedocument WHERE id =
NEW.id;
  SELECT Count(id) into idCmdRevue FROM abonnement WHERE id = NEW.id;

  IF(idCmdDoc > 0  || idCmdDoc > 0) THEN
    SIGNAL SQLSTATE "45000"
      SET MESSAGE_TEXT = "opération impossible doublon idCmd" ;
  END IF;

END ;
```

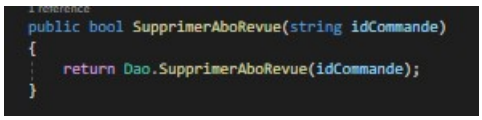
**SupprimerCmd___**

Cette méthode permet de lancer une requête SQL afin de supprimer dans la base de données une ligne de commande définie par son identifiant de commande d'un Livre ou d'un Dvd. Par le biais d'un objet de type Dictionnaire, les différentes valeurs de la commande sont insérés en paramètres, soit ici l'identifiant de la commande, puis un appel vers la Classe BddMySql permet d'initialiser un curseur qui recevra l'instance de cette Classe, puis un appel avec sa méthode ReqUpdate permet l'insertion dans la table commandedocument. De la même manière, un second delete doit s'effectuer dans la table commande.



De la même manière, ces méthodes peuvent être appelées depuis la classe Controle.




### UpdateCmdDocument

Cette méthode permet de lancer une requete SQL afin de modifier dans la base de données une ligne de commande définie par son identifiant de commande d'un Livre ou de Dvd. Par le biais d'un objet de type Dictionnaire, les différentes valeurs de la commande sont insérés en paramètres, soit ici l'identifiant de la commande et l'id du Suivi, puis un appel vers la Classe BddMySql permet d'initialiser un curseur qui recevra l'instance de cette Classe, puis un appel avec sa méthode ReqUpdate permet la mise à jours dans la table commande. Il n'y a pas d'équivalent de cette méthode pour les abonnements

```
public static bool UpdateCmdDocument(string idCommande, string idSuivi)
{
    try
    {
        string req = "UPDATE commande SET idSuivi = @idSuivi WHERE id = @id";

        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@id", idCommande},
            { "@idSuivi", Int32.Parse(idSuivi)}
        };
        BddMySql curs = BddMySql.GetInstance(connectionString);
        curs.ReqUpdate(req, parameters);
        curs.Close();
        return true;
    }
    catch
    {
        return false;
    }
}
```

```
3 références
public bool UpdateCmdDocument(string idCommande, string idSuivi)
{
    return Dao.UpdateCmdDocument(idCommande, idSuivi);
}
```

De la même manière, ces méthodes peuvent être appelées depuis la classe Controle.

## Méthodes supplémentaires de gestion des abonnements

### ParutionDansAbonnement

Un abonnement ne peut être supprimé que si la date de parution de la revue en question se situe entre la date d'abonnement (ou ici de commande) et la date de fin d'abonnement, ainsi une méthode a été ajouté dans la classe Dao afin de pouvoir gérer cette situation.

```
/// <summary>
/// Methode qui vérifie si la date d'achat de l'exemplaire est comprise entre la date de parution la commande et la date de fin d'abonnement
/// </summary>
/// <param name="dateCommande">Date de la commande</param>
/// <param name="dateFin">Date de fin d'abonnement</param>
/// <param name="dateParution">Date de parution d'un numero de revue</param>
/// <returns>Vrai si dateParution est entre les 2 autres dates</returns>
4 références
public static bool ParutionDansAbonnement(DateTime dateCommande, DateTime dateFin, DateTime dateParution)
{

    if ((dateCommande < dateParution && dateParution < dateFin) || dateParution == DateTime.MinValue)
    {
        return true;
    }

    return false;
}
```

```
1 référence
public bool ParutionDansAbonnement(DateTime dateCommande, DateTime dateFin, DateTime dateParution)
{
    return Dao.ParutionDansAbonnement(dateCommande, dateFin, dateParution);
}
```

Cette méthode est appelée depuis la vue lors d'un événement sur le clic du bouton supprimer de l'onglet abonnement.

*Création d'une fenêtre d'alerte des revues fin d'abonnement.*

Pour cela plusieurs modifications ont été effectué. Dans un premier temps une fonction stockée a été ajouté depuis PhpMyAdmin, nommée fctEndingAbo. Cette fonction permet de récupérer sous forme de chaine concaténée, via la méthode GROUP_CONCAT, la liste des idRevue dont la fin d'abonnement est dans moins de 30jours.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` FUNCTION `fctEndingAbo`()
RETURNS varchar(255) CHARSET utf8 COLLATE utf8_unicode_ci
  READS SQL DATA
BEGIN
  DECLARE idRevues VARCHAR(5);
  DECLARE grpStr VARCHAR(255);
```

```
  SELECT DISTINCT idRevue,
   GROUP_CONCAT( DISTINCT idRevue)
   INTO idRevues,  grpStr
   FROM abonnement
   WHERE dateFinAbonnement <= DATE_ADD(NOW(), INTERVAL 1  MONTH);

 IF(grpStr = null) THEN
  SET grpStr = "x";
 END IF;

RETURN grpStr;
END$$
DELIMITER ;
```

*Ajout dans Dao*

Cette fonction stockée est appelée depuis la Classe Dao via la méthode GetEndingAbonnement par une requête SQL de type Select, permettant ainsi de récupérer les valeurs qu'elle retourne.

```
/// <summary>
/// Methode permettant de lancer une requete SQL SELECT d'appel de fonction stockée
/// </summary>
/// <returns>Chaine d'idRevues des revues en fin d'abonnement </returns>
1 référence
public static String GetEndingAbonnement()
{
    string strFinAbo = "";

    string req = "SELECT fctEndingAbo() as id;";

    BddMySql curs = BddMySql.GetInstance(connectionString);
    curs.ReqSelect(req, null);


    while (curs.Read())
    {

        strFinAbo += (string)curs.Field("id");
    }
    curs.Close();

    return strFinAbo;
}
```

En parallèle, une autre méthode est créée, nommée GetEndingTitleDate, dans cette classe permettant de récupérer la liste des titres et des fins d'abonnements concernés par ces revues via une seconde requête Select, elle permet de retourner un dictionnaire contenant ces informations.

```
2 références
public static Dictionary<string, string> GetEndingTitleDate(string strIdRevues)
{
    string[] lstlesRevues = strIdRevues.Split(',');
    DateTime dateFinAbonnement;
    Dictionary<string, string> dictFinAbo = new Dictionary<string, string>();

    foreach (var item in lstlesRevues)
    {
        string req = "SELECT Distinct titre, dateFinAbonnement FROM abonnement a JOIN document d ON (d.id = a.idRevue) ";
        req += " WHERE idRevue = @liste GROUP BY titre ";

        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@liste", item}
        };

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

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

            dictFinAbo.Add(titre, dateFinAbonnement.ToShortDateString());
        }
        curs.Close();
    }
    return dictFinAbo;
}
```

*Ajout dans le controleur*

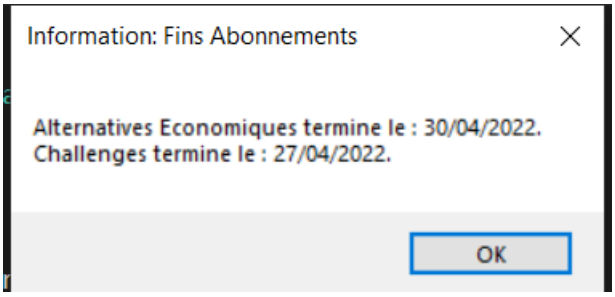
Dans le contrôleur, deux méthodes gèrent ces méthodes la première permet de récupérer la liste des idRevue dans une liste lesFinAbo pour la stocker dans une variable globale. La seconde, GetEndingTitleDate permet d'appeler la méthode de même nom dans la Classe Dao afin de récupérer le dictionnaire contenant les titres et dates de fin d'abonnement pour les concaténer dans une chaîne qui servira à l'affichage.

```
/// <summary>
/// Methode du controleur d'appel de la methode GetAllRevues
/// Getter sur la chaine lesFinAbo
/// </summary>
/// <returns>Chaine de liste d'idRevue</returns>
0 références
public String GetEndingAbonnement()
{
    return lesFinAbo;
}

/// <summary>
/// Methode du controleur accédant à la méthode GetEndingTitleDate
/// Permet la récupération de la liste des revues en fin d'abonnements
/// </summary>
/// <returns>La chaine d'alerte à afficher</returns>
1 référence
public String GetEndingTitleDate()
{
    Dictionary<String, String> dictFinAbo = Dao.GetEndingTitleDate(lesFinAbo);
    string strList = "";

    foreach (var item in dictFinAbo)
    {
        strList += item.Key + " termine le : " + item.Value + ".\n";
    }
    return strList;
}
```

*Ajout dans la vue*

Une dernière méthode dans la vue FrmMediatek a été ajouté permettant d'afficher la fenêtre d'alerte des abonnements au lancement de l'application. Elle est ainsi lancée depuis le constructeur de la vue.

```
/// <summary>
/// Constructeur de la Classe
/// Récupére l'instance du controleur de la Classe Controle
/// </summary>
/// <param name="controle"></param>
1 référence
internal FrmMediatek(Controle controle)
{
    InitializeComponent();

    this.controle = controle;
    MessageBox.Show(controle.GetEndingTitleDate(), "Information: Fins Abonnements");

}
```




### Dépot github

A l'issue de ce code, un commit a été effectué sur la plateforme GitHub dans le dépôt du projet.

## Mission 3 : gérer le suivi de l'état des documents

(Non implémentée)

## Mission 4 : mettre en place des authentifications

### Objectifs

Cette mission a pour objectif la mise en place des restrictions d'accès à certaines fonctionnalités de l'application, suivant l'habilitation de l'utilisateur de l'application.

- **Dans la base de donnée**
  - Ajouts des table Utilisateur et une table Service sachant que chaque utilisateur ne fait partie que d'un service.

Figure 29: Suivi Trello Mission 4

- Remplir les tables d'exemples pour contrôler des tests

- **Dans la vue**
  - Fenêtre d'authentification que le contrôleur ouvre en première
  - Service Culture qui n'a accès à rien : afficher un message précisant que les droits ne sont pas suffisants pour accéder à cette application, puis fermer l'application
- **Dans le contrôleur**
  - suivant le type de personne authentifiée, empêcher certains accès en rendant invisibles ou inactifs certains onglets ou objets graphiques
  - L'alerte de fin d'abonnement n'apparaisse que pour les personnes concernées (qui gèrent les commandes).

## Modifications dans la base de données

Dans un premier temps des ajouts de deux nouvelles tables ont été mis en place, soit la table service qui énumère les différents types de services de l'entreprise (Culture, Prêt, Administration et Administratif), puis la table utilisateurs qui contient la liste des utilisateurs avec leur login et mot de passe, ainsi que l'identifiant de leur service. Ces ajouts ont été effectué depuis l'onglet d'exécution des requêtes SQL de phpMyAdmin

### Création et Insert de la table service

```sql
DROP TABLE IF EXISTS service;
CREATE TABLE service (
  idService int PRIMARY KEY NOT NULL AUTO_INCREMENT,
  label varchar(15) NOT NULL
)
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```sql
INSERT INTO service(label)
VALUES
('administratif'),
('prêts'),
('culture'),
('administrateur');
```

### Création et Insert de la table utilisateur

```sql
DROP TABLE IF EXISTS `utilisateur`;
CREATE TABLE IF NOT EXISTS `utilisateur` (
  `idUser` int(11) NOT NULL AUTO_INCREMENT,
```

```
 `idService` int(3) NOT NULL,
 `pwd` varchar(255) NOT NULL,
 `login` varchar(255) NOT NULL,
 PRIMARY KEY (`idUser`),
 KEY ` commande_ibfk_1` (`idService`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

### Sa clé étrangère vers idService

```
ALTER TABLE `utilisateur`
  ADD CONSTRAINT ` commande_ibfk_1` FOREIGN KEY (`idService`) REFERENCES `service` (`idService`);
COMMIT;
```

### Exemples d'entrées test

```
INSERT INTO utilisateur(idService, pwd, login)
VALUES (1,'ppfdsfhtrts3','adflog1'),
(2,'rts3fdsfhtrt','prtlog1'),
(3,'pZpfgsfhtrts3','cltlog1'),
(4,'ppfdsfhtrts3','adminlog1'),
(1,'3Eh63Gfe4htC','adflog2'),
(2,'5gsfhdemGT5G','prtlog2'),
(3,'r4gfpfdsfht','cltlog2'),
(4,'slogr4gfdgdA','adminlog2');
```

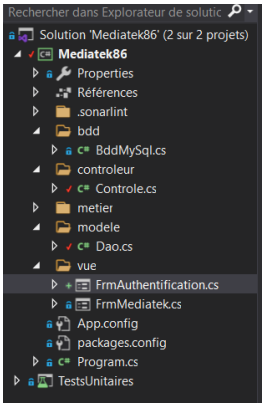
### Authentifications

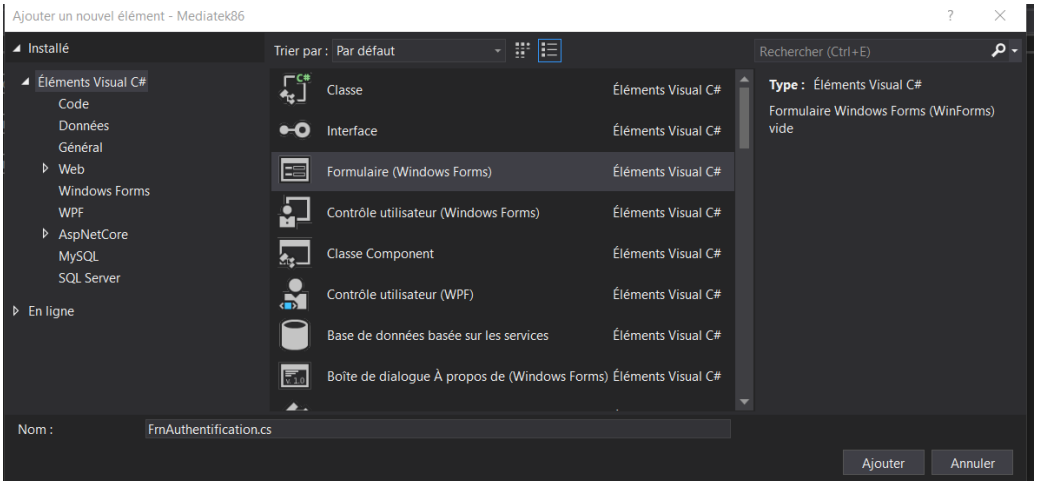
```
admf_pwd   || admf_log  || Administratif
prt_pwd    || prt_log   || Prets
clt_pwd    || clt_log   || Culture
admin_pwd   || admin_log ||     Administrateur
```

## Mise en place fenêtre d'authentification

### Modification du package vue

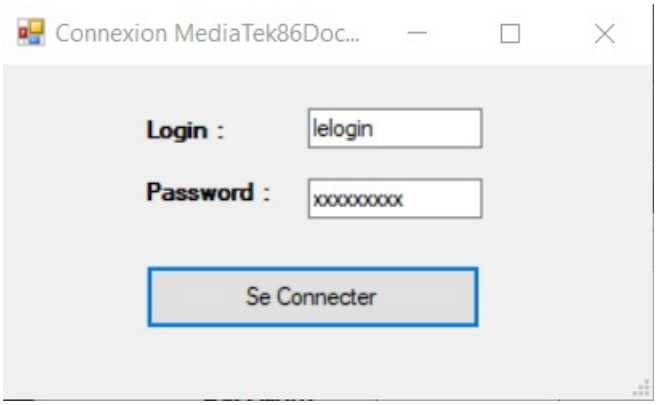
Dans l'IDE, dans le package vue, sur un clique droit sélectionner ajouter un nouvel élément, choisir formulaire Windows Forms. Un nouveau formulaire, nommé FrmAuthentification est créé dans le package vue.

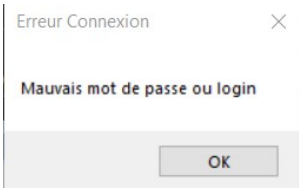


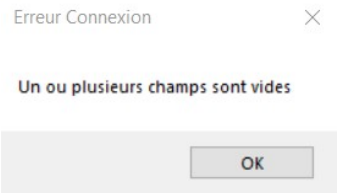



### Création dans la vue

Dans la vue, deux textBox récupèrent le login et un mot de passe où les longueurs maximales ont été mise à 15 caractères, aussi un bouton de validation a été mis en place.

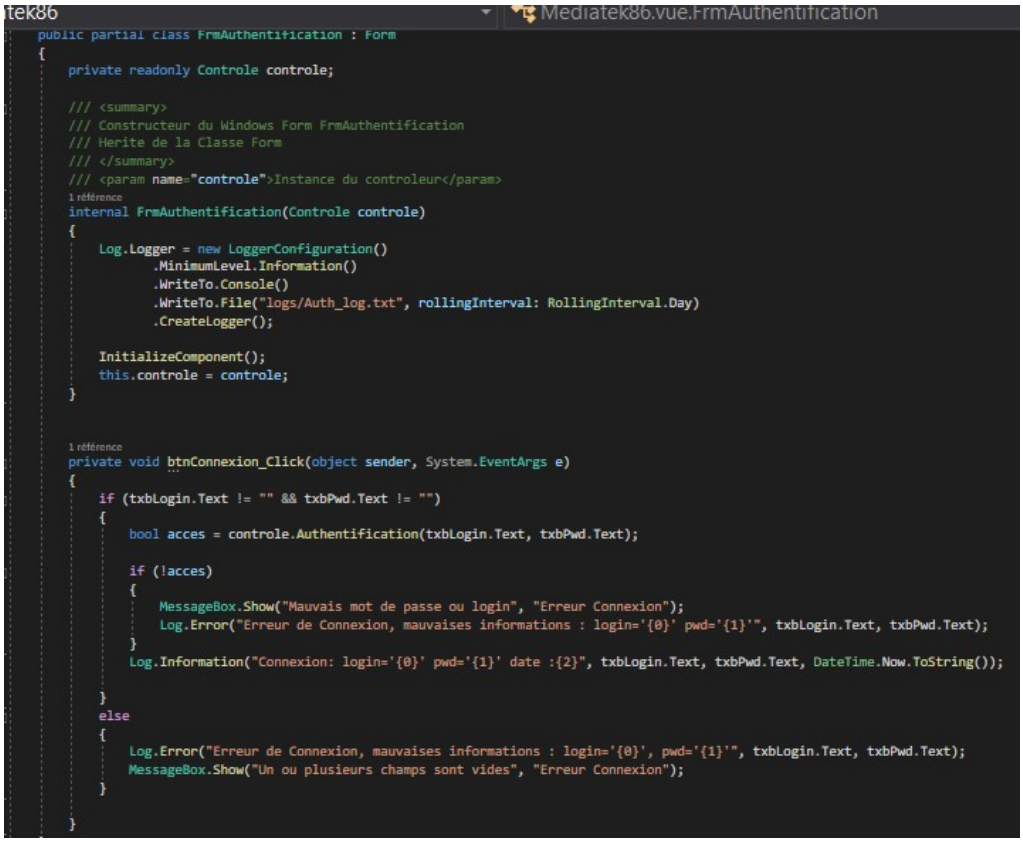



Aussi, des fenêtres d'erreurs de logs ont été mise en place afin d'informer qu'un champs est vide ou que les informations de connexion sont erronées.

La Classe FrmAuthentification permet ainsi d'initialiser ces composants graphiques et appeler le controleur. Aussi, une méthode événement permet la gestion du clic sur le bouton de connexion, elle est nommée btnConnexion_click. Elle permet aussi de gérer les fenêtres d'erreurs de connexion.




### Création des méthodes dans Dao et Controle

Ces méthodes permettent de lancer une requête SQL afin de récupérer dans la base de données de les informations de l'utilisateur demandée sur le formulaire de connexion. Par le biais d'un objet de type Dictionnaire, les différentes valeurs d'authentifications sont insérés en paramètres(login et mot de passe), puis un appel vers la Classe BddMySql permet d'initialiser un curseur qui recevra l'instance de cette Classe, enfin un appel avec sa méthode ReqSelect permet la récupération des informations dans la table utilisateur, si les identifiants insérés sont trouvés dans la base. Aussi, une fonction de haschage permet de récupérer les informations de login et mots de passes insérés en cryptés dans la base.

```
4 références
public static Utilisateur Authentification(string login, string pwd)
{
    try
    {
        string label = "";
        int idUser = 0;
        int idSuivi = 0;

        string req = "Select u.idUser, u.idService, s.label";
        req += " from utilisateur u join service s on u.idService = s.idService";
        req += " where u.pwd = SHA2(@pwd, 256) AND u.login = SHA2(@login, 256) ;";

        Dictionary<string, object> parameters = new Dictionary<string, object>() {
            { "@pwd", pwd},
            { "@login", login}
       };



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

        while (curs.Read())
        {
            idUser = curs.Field("idUser") is DBNull ? 0 : (int)curs.Field("idUser");

            idSuivi = curs.Field("idService") is DBNull ? 0 : (int)curs.Field("idService");

            label = curs.Field("label") is DBNull ? "" : (string)curs.Field("label");

        }
        curs.Close();

        if (idUser != 0 && idSuivi != 0 && label != "")
        {
            Utilisateur user = new Utilisateur(idUser, idSuivi, label);
            return user;

        }
        return null;
```

Dans la Classe Controle des modifications ont été effectué afin d'appeler cette fenêtre en première depuis un appel de la Classe Program.cs. Lorsque les informations saisies correspondent a un utilisateur de la base, alors la méthode retourne vrai ce qui permet de lancer la suite de l'application en appelant la Classe FrmMediatek.

```
1 référence
public Controle()
{

    lesLivres = Dao.GetAllLivres();
    lesDvd = Dao.GetAllDvd();
    lesRevues = Dao.GetAllRevues();
    lesGenres = Dao.GetAllGenres();
    lesRayons = Dao.GetAllRayons();
    lesPublics = Dao.GetAllPublics();
    lesSuivis = Dao.GetAllSuivis();
    lesFinAbo = Dao.GetEndingAbonnement();

    frmAuth = new FrmAuthentification(this);
    frmAuth.ShowDialog();

}
```

```
1 référence
public Boolean Authentification(string login, string pwd)
{
    user = Dao.Authentification(login, pwd);

    if (user != null)
    {
        frmAuth.Hide();

        frmMediatek = new FrmMediatek(this);
        AccesServices();

        return true;
    }

    return false;

}
```

## Gestion de la restriction des accès des services

Les accès à certaines fonctionnalités doivent être restreinte selon le service de l'utilisateur affecté, cette information est récupérée lors de la connexion de l'utilisateur. Une fois l'utilisateur trouvé et son service récupéré, la méthode AccesServices est appelée depuis la méthode Authentification du controleur. Cette méthode permet de déterminer ce qui affiché ou non dans la vue selon l'identifiant du service d'affectation via un switch.

```
private void AccesServices()
{
    frmMediatek.Text = "Gestion Médiathèque: " + user.Label;
    switch (user.IdService)
    {
        case 4: //Admin
        case 1: //Administratif
            frmMediatek.AfficheFinAbo();
            frmMediatek.ShowDialog();

            break;

        case 2: //Prêts
            frmMediatek.AccesServicePrets();
            frmMediatek.ShowDialog();
            break;

        case 3: //Culture
            frmMediatek.AfficheServiceCulture();
            frmMediatek.Close();
            break;

        default: //None
            frmMediatek.AfficheServiceNone();
            frmMediatek.Close();
            break;
    }

}
#endregion
```

Les services Admin et administratif ont actuellement les mêmes priviléges et l'alerte de fin d'abonnement doit s'afficher pour eux. Pour le service Prêt, il n'y pas d'alerte et les onglets de commandes sont supprimés via la méthode AcccesServicesPrêts.

```
#region Acces
/// <summary>
/// Methode de supression d'access aux onglets de commandes
/// </summary>
1 référence
public void AccesServicePrets()
{
    tabOngletsApplication.TabPages.Remove(tabCmdRevue);
    tabOngletsApplication.TabPages.Remove(tabCmdDvd);
    tabOngletsApplication.TabPages.Remove(tabCmdLivre);
    tabOngletsApplication.TabPages.Remove(tabReceptionRevue);
}

/// <summary>
/// Methode d'affichage des fin abonnements
/// </summary>
1 référence
public void AfficheFinAbo()
{
    MessageBox.Show(controle.GetEndingTitleDate(), "Information: Fins Abonnements");
}

1 référence
public void AfficheServiceCulture()
{
    MessageBox.Show("Application indisponible pour le service Culture", "Informations");
}

1 référence
public void AfficheServiceNone()
{
    MessageBox.Show("Application indisponible: service non détecté", "Erreur!");
}

#endregion
```

Enfin, pour le service Culture, une alerte leur indique qu'ils n'ont pas de droit d'accès à ce service puis l'application se ferme. Au cas, où le service n'est pas détecté une alerte leur indiquant que l'application est indisponible est affichée puis l'application se ferme aussi.

Le commit de l'ajout de ce code est trouvable à ce lien

# Mission 5 : Qualité du code

## Suivi de projet

La mission 4 terminée, ses étiquettes dans ce tableau modifiées pour être mises en violet pour signifier leurs clôtures, nous obtenons alors le tableau de suivi visible. La mission suivante a pour objectif de mettre en place la qualité du code produit, ainsi une liste de 4 quatre ont été établis.Ainsi les différentes étapes établies sont les suivantes :

- Contrôler la qualité du code avec SonarLint
- Création des tests unitaires avec MS Test
- Création de logs avec Serilog dans des fichiers
- Création de la documentation technique de l'application

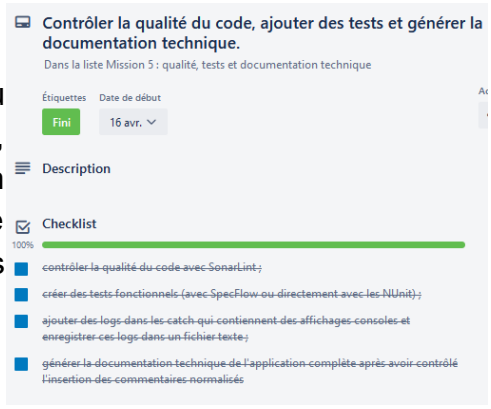
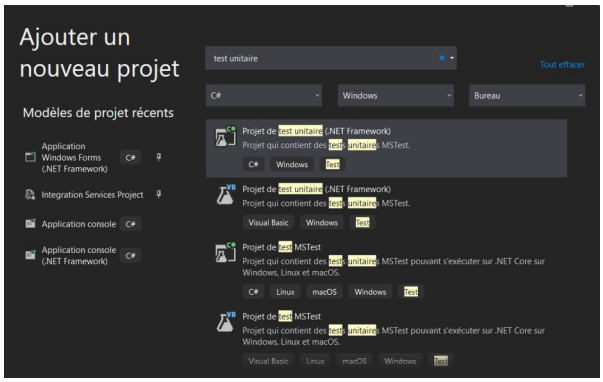
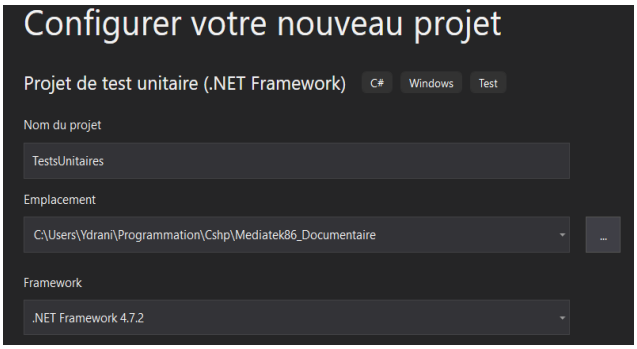



Figure 30: Carte de suivi de mission de la mission 5

## Tests unitaires

Trois Classes sont concernées par les tests unitaires effectués avec MS Test. Pour mettre les classes de test en place, il est nécessaire d'ajouter un nouveau projet à la solution puis de sélectionner Projet de test unitaire NET Framework en C#, puis de donner un nom au projet, ici TestMediatek




Pour créer une Classe à tester, il convient de se rendre dans la Classe concernée, d'effectuer un clic droit sur le nom de la Classe puis de cliquer sur créer des tests unitaires. Ainsi, une classe de test est automatiquement générée avec l'ensemble des méthodes de la Classe concernée, après avoir sélectionné le nom de projet de test avec lequel associé ces tests ainsi que le code de la méthode de test sur Corps vide.

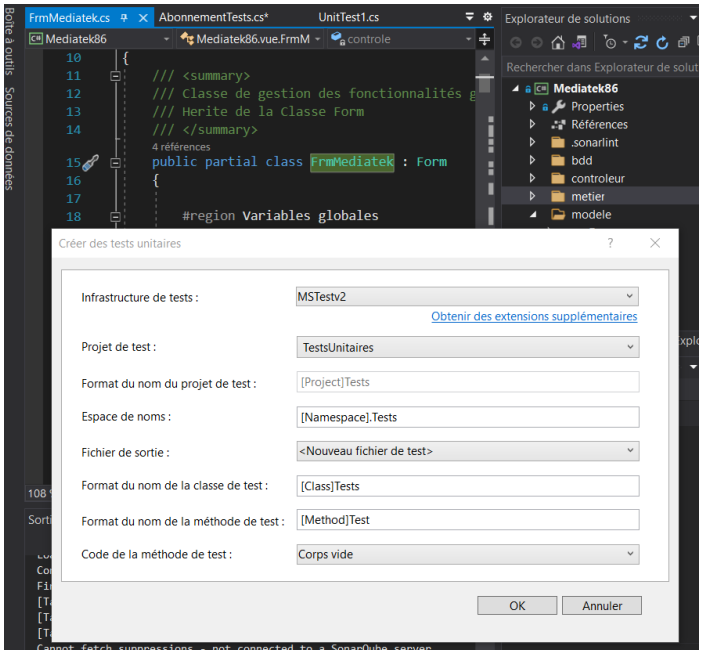



Figure 31: Exemple de fenêtre de gestion de création de tests unitaires

### Classe AbonnementTest

Dans cette classe de test pour la Classe Abonnement, l'ensemble des getters sur les différentes propriétés sont testés avec une valeur test prédéfinie et celle appelée par l'application

```
namespace Mediatek86.metier.Tests
{
    [TestClass()]
    0 références
    public class AbonnementTests
    {
        private const string idCommande = "00002";
        private static DateTime dateCommande = DateTime.Now;
        private const double montant = 23;
        private const string idSuivi = "1";
        private const string label = "en cours";
        private const string idRevue = "00002";
        private static DateTime dateFinAbonnement = DateTime.Now;

        private readonly Abonnement abo = new Abonnement(idCommande, dateCommande, montant, idSuivi, label, idRevue, dateFinAbonnement);

        [TestMethod()]
        0 références
        public void AbonnementTest()
        {

            Assert.AreEqual(idCommande, abo.IdCommande, "devrait réussir : idCommande valorisé");
            Assert.AreEqual(idRevue, abo.IdRevue, "devrait réussir : idRevue valorisé");
            Assert.AreEqual(idSuivi, abo.IdSuivi, "devrait réussir : idSuivi valorisé");
            Assert.AreEqual(label, abo.Label, "devrait réussir : label valorisé");
            Assert.AreEqual(montant, abo.Montant, "devrait réussir : montant valorisé");
            Assert.AreEqual(dateCommande, abo.DateCommande, "devrait réussir : dateCommande valorisé");
            Assert.AreEqual(dateFinAbonnement, abo.DateFinAbonnement, "devrait réussir : dateFinAbonnement valorisé");

        }
    }
```

### Classe UtilisateurTest

De même, ici pour la classe Utilisateur

```
namespace Mediatek86.metier.Tests
{
    [TestClass()]
    0 références
    public class UtilisateurTests
    {
        private const int idUser = 2;
        private const int idService = 34;
        private const string label = "dev";

        private readonly Utilisateur user = new Utilisateur(idUser, idService, label);

        [TestMethod()]
        0 références
        public void UtilisateurTest()
        {
            Assert.AreEqual(idUser, user.IdUser, "devrait réussir : idUser valorisé");
            Assert.AreEqual(idService, user.IdService, "devrait réussir : idRevue valorisé");
            Assert.AreEqual(label, user.Label, "devrait réussir : idSuivi valorisé");

            Assert.AreNotEqual(23, user.IdUser, "devrait échouer");
            Assert.AreNotEqual(12, user.IdUser, "devrait échouer");
            Assert.AreNotEqual("gfsdgs", user.Label, "devrait échouer");
        }
    }
}
```

### Classe DaoTest

L'intérêt de cette classe étant de tester les retours de la base de données, il a été nécessaire de mettre en place plusieurs méthodes afin que les requêtes exécutées n'apportent pas de modification à la base de données. Aussi, une nouvelle instance de connexion à la base de données à été mise en place avec les informations de connexion à la base de test en locale, via la méthode GetInstance de la Classe BddMySql.

Les méthodes BeginTransaction et EndTransaction permettent réciproquement de mettre les requêtes exécutées en mode transaction puis de pouvoir revenir à l'état initial. Cela se produit grâce à des requêtes SQL envoyées à la base permettant de lui indiquer qu'il ne doit pas y avoir de commit dans la base et que le mode transaction est activé. Puis, après l'exécution des requêtes SQL à tester, la requête SQL RollBack permet de retourner à l'état initial.

```
/// Connection Distante
private static readonly string server = "localhost";
private static readonly string userid = "root";
private static readonly string password = "";
private static readonly string database = "mediatek86";
private static readonly string connectionString = "server=" + server + ";user id=" + userid + ";password=" + password + ";database=" + database + ";SslMode=none";


//Recupération de l'instance de la BDD
readonly BddMySql curs = BddMySql.GetInstance(connectionString);

/// <summary>
/// Methode qui envoie une requete de ne pas faire de modification dans la base
/// Met en place le mode Transaction
/// </summary>
6 références
private void BeginTransaction()
{
    curs.ReqUpdate("SET AUTOCOMMIT=0", null);
    curs.ReqUpdate("START TRANSACTION", null);
}

/// <summary>
/// Methode qui envoie une requete pour revenir en arrière
/// Stop le mode Transaction
/// </summary>
6 références
private void EndTransaction()
{
    curs.ReqUpdate("ROLLBACK", null);
}
```

De même, certaines tables étant soumises à des contraintes, dont par exemple de partition, il est nécessaire de désactiver ces contraintes pour les méthodes de test, ainsi les méthodes suivantes permettent cette désactivation, puis de les réactiver.

```
/// <summary>
/// Méthode qui envoie une requete permettant d'annuler les vérifications de contraintes
/// </summary>
6 références
private void AnnuleContraintes()
{
    curs.ReqUpdate("SET FOREIGN_KEY_CHECKS=0", null);
}
/// <summary>
/// Méthode qui envoie une requete permettant d'activer les vérifications de contraintes
/// </summary>
6 références
private void ActiveContraintes()
{
    curs.ReqUpdate("SET FOREIGN_KEY_CHECKS=1", null);
}
```

Ainsi, il est possible de prendre pour exemple la méthode de test de création de commande :

```
/// <summary>
/// Methode de test de Dao.CreerCommande
/// </summary>
[TestMethod()]
0 références
public void CreerCommandeTest()
{
    BeginTransaction();
    AnnuleContraintes();
    string id = "aaaaa";
    DateTime dateCommande = DateTime.Now;
    double montant = 23.3;
    string idSuivi = "1";
    string label = "en cours";
    string idLivreDvd = "00017";
    int nbExemplaire = 1;


    string idDoc = "00017";
    List<CommandeDocument> lstCmdDoc = Dao.GetAllCommandesDocument(idDoc);
    int nbAvantInsert = lstCmdDoc.Count;

    CommandeDocument cmdDoc = new CommandeDocument(id, dateCommande, montant, idSuivi, label, idLivreDvd, nbExemplaire);
    Dao.CreerCommande(cmdDoc);

    List<CommandeDocument> lstCmdDocApres = Dao.GetAllCommandesDocument(idDoc);
    int nbAprestInsert = lstCmdDocApres.Count;

    CommandeDocument cmdDocAdd = lstCmdDocApres.Find(doc => doc.IdCommande.Equals(id)
        && doc.Montant.Equals(montant)
        && doc.IdSuivi.Equals(idSuivi)
        && doc.Label.Equals(label)
        && doc.IdLivreDvd.Equals(idLivreDvd)
        && doc.NbExemplaire.Equals(nbExemplaire)
        );

    Assert.IsNotNull(cmdDocAdd, "devrait réussir : une commande ajoutée");
    Assert.AreEqual(nbAvantInsert + 1, nbAprestInsert, "devrait réussir : un développeur en plus");
    ActiveContraintes();
    EndTransaction();

}
```

A l'appel de la méthode de test, les méthodes BeginTransaction et AnnuleContrainte sont appelé afin de mettre en place le mode Transaction. Puis, le nombre de commande au numéro du document est récupérée dans une variable et la nouvelle commande est instanciée après avoir initialisé les propriétés à tester pour la nouvelle commande à envoyer.

Une fois la requête de commande envoyée, on récupère à nouveau le nombre de commande existant pour le document concerné. Il est alors possible de tester si le nombre

de commande avant l'exécution de la requête +1 est égale au nouveau nombre de commande. Aussi, il est possible d'essayer de récupérer la commande en filtrant sur les informations envoyées afin de pouvoir si l'instance récupérée est null ou non. Puis les méthodes de réactivation de contrainte et de RollBack sont appelées afin d'annuler l'envoi de cette nouvelle commande.

Parmi les autres méthodes testés, il est possible de prendre en compte la méthode de test sur la méthode ParutionDansAbonnement qui récupéré les valeurs de trois dates différentes et teste la méthode avec ces dates dans des ordres différents.

```
    /// Vrai si dateParution est entre les 2 autres dates
    /// </summary>
    [TestMethod()]
    0 références
    public void ParutionDansAbonnementTest()
    {
        // Varibles test ParutionDansAbonnement
        DateTime dateMin = DateTime.Parse("01/04/2022"); // data achat Exemplaire
        DateTime date = DateTime.Parse("03/04/2022"); // date Fin abo
        DateTime dateMax = DateTime.Parse("13/04/2022"); // date Parution

        //  ((dateCommande < dateParution && dateParution < dateFin) )

        Assert.AreEqual(true, Dao.ParutionDansAbonnement(dateMin, dateMax, date));
        Assert.AreEqual(false, Dao.ParutionDansAbonnement(dateMax, date, dateMin));
        Assert.AreEqual(false, Dao.ParutionDansAbonnement(dateMax, dateMin, date));
    }
}
```

L'ensemble des méthodes de tests de cette classe est visible à ce lien.

## Mise en place des logs

Les logs de journalisations sont effectués avec l'ensemble des extensions de Serilog permettant de recevoir les logs depuis la Console et depuis des journaux textes. Ces extensions sont installées depuis le gestionnaire d'extension de l'IDE. Afin de pouvoir utiliser cette extension, il est nécessaire de placer son import dans la déclaration des using. Ici, trois Classes sont concernées par cette journalisation. Ces logs sont récupérables depuis le dossier :

MediatekGest_Documentaire\Mediatek86gestion\bin\Release\logs

### Logs dans BddMySql

La première est celle de la Classe BddMySql, ainsi à l'ensemble des catch d'erreurs des logs sont récupérés vers les fichiers log.txt, Error_log.txt, Fatal_log.txt. Son instanciation est définie dans le constructeur de la Classe et permet de définir ce qui doit être écrit dans les logs, ici tout ce qui relève du niveau minimum Debug (soit Debug, Information, Error et Fatal)

Ainsi aux niveaux des catchs de la Classe des demandes de logs adaptés aux niveaux de l'erreur ont été ajouté dans le code.

```
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)
    {
        Console.WriteLine(e.Message);
        Log.Error("BddMySql.ReqSelect catch stringQuery={0} erreur={1}", stringQuery, e.Message);
    }
    catch (InvalidOperationException e)
    {
        ErreurGraveBddNonAccessible(e, "");
    }
}
```

```
public void ReqUpdate(string stringQuery, Dictionary<string, object> parameters)
{
    MySqlCommand command;
    try
    {
        command = new MySqlCommand(stringQuery, connection);
        if (!(parameters is null))
        {
            Log.Debug("BddMySql.ReqUpdate stringQuery={0}", stringQuery);
            foreach (KeyValuePair<string, object> parameter in parameters)
            {
                Log.Debug("BddMySql.ReqUpdate parameter.Key={0} parameter.value={1}", parameter.Key, parameter.Value);
                command.Parameters.Add(new MySqlParameter(parameter.Key, parameter.Value));
            }
        }
        command.Prepare();
        command.ExecuteNonQuery();
    }
    catch (MySqlException e)
    {

        Console.WriteLine(e.Message);
        Log.Error("BddMySql.ReqUpdate catch stringQuery={0} erreur={1}", stringQuery, e.Message);
    }
    catch (InvalidOperationException e)
    {
        ErreurGraveBddNonAccessible(e, "");
    }
}
```

```
/// <summary>
/// Pas d'accès à la BDD : arrêt de l'application.
/// </summary>
/// <param name="e">The e<see cref="Exception"/>.</param>
3 références
private void ErreurGraveBddNonAccessible(Exception e, String stringConnect)
{
    Log.Fatal("BddMySql.BddMySql catch stringConnect={0} erreur={1}", stringConnect, e.Message);
    MessageBox.Show("Base de données non accessibles", "Erreur grave");
    Console.WriteLine(e.Message);
    Environment.Exit(1);
}
```

De même dans la Classe FrmAuthentification, une instanciation de log a été mise en place afin de récupérer les informations et erreurs exécutées lors des tentatives de connexion.

```
1 référence
internal FrmAuthentification(Controle controle)
{
    Log.Logger = new LoggerConfiguration()
          .MinimumLevel.Information()
          .WriteTo.Console()
          .WriteTo.File("logs/Auth_log.txt", rollingInterval: RollingInterval.Day)
          .CreateLogger();

    InitializeComponent();
    this.controle = controle;
}


1 référence
private void btnConnexion_Click(object sender, System.EventArgs e)
{
    if (txbLogin.Text != "" && txbPwd.Text != "")
    {
        bool acces = controle.Authentification(txbLogin.Text, txbPwd.Text);

        if (!acces)
        {
            MessageBox.Show("Mauvais mot de passe ou login", "Erreur Connexion");
            Log.Error("Erreur de Connexion, mauvaises informations : login='{0}' pwd='{1}'", txbLogin.Text, txbPwd.Text);
        }
        Log.Information("Connexion: login='{0}' pwd='{1}' date :{2}", txbLogin.Text, txbPwd.Text, DateTime.Now.ToString());

    }
    else
    {
        Log.Error("Erreur de Connexion, mauvaises informations : login='{0}', pwd='{1}'", txbLogin.Text, txbPwd.Text);
        MessageBox.Show("Un ou plusieurs champs sont vides", "Erreur Connexion");
    }

}
```

Ces logs sont retrouvables dans le fichier Auth_log.txt.

## Contrôler la qualité du code avec SonarLint et SonarQube

### SonarLint

**SonarLint** étant un plugin pouvant être intégré sur Visual Studio, la grande majorité des morceaux de codes mal écrits ou d'erreurs de sécurité peuvent être rapportés directement dans un console dédiée.

Ainsi, plusieurs alertes comme celle illustrée en *figure 30* ont été directement gérés. Sur cette figure, l'alerte signale que :

- Le constructeur de Document a plus de 9 paramètres, ce qui ne correspond pas à la norme attendue.
- Une instanciation de Controle n'est pas nécessaire dans Program.cs
- Un morceau de code commenté peut être supprimé.

Or, ces trois alertes peuvent être marqué comme non corrigé puisque ces erreurs sont volontaire et nécessaire.

En effet, il n'y a pas d'autre choix que de laisser les 9 parametres du constructeur de Document. L'instanciation de Controle est nécessaire à l'ouverture et à la gestion des fenêtres de l'application. Enfin, le code commenté dans la Classe Dao est un rappel des propriétés pour la connexion en locale à la base de données.

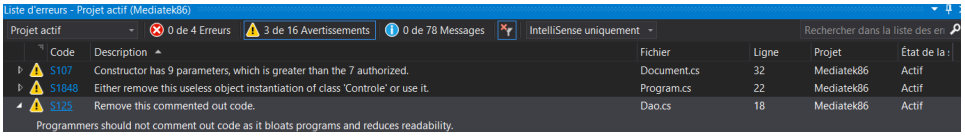



Figure 32: Exemples d'alertes traitées avec le plugin SonarLin

### Corrections de SonarQube

Il est alors possible de constater plusieurs alertes dont 1 notée comme bugs et 14 de mauvaises utilisation du code « Code Smells ».

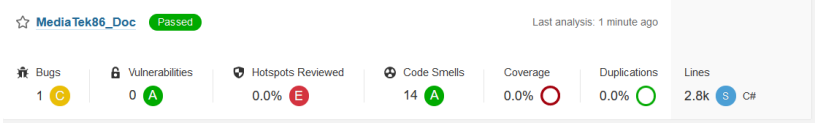



Figure 33: Relevé d'analyse par SonarQube après évolution.

La résolution de ces alertes se fait individuellement, en cliquant sur le message de l'erreur. Aussi, il est possible d'avoir une explication de l'alerte avant d'estimer s'il faut la réparer ou non. Ce choix ce fait par l'intermédiaire du bouton **Open** visible sur chacun des messages de l'analyse.

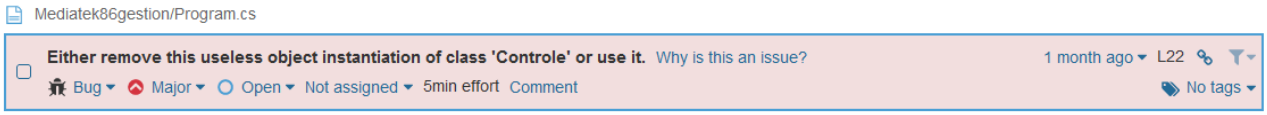
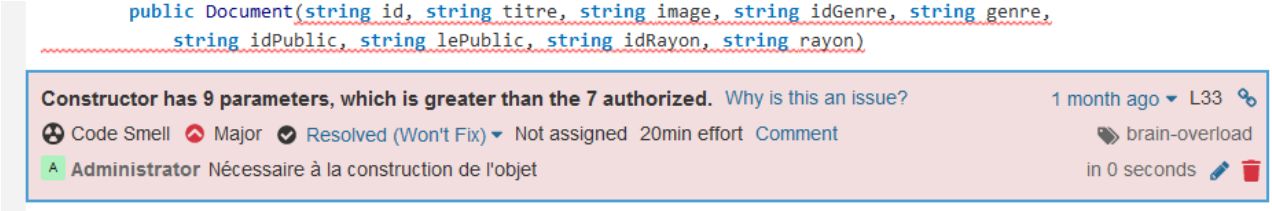



Figure 34: Exemple d'alerte de bugs

Ici, il est possible de constater une erreur relevant d'une instanciation de la Classe Controle dans le fichier Programs.cs qui n'est pas considéré comme nécessaire. Puisque cette instanciation est nécessaire à l'ouverture des fenêtres de l'application, étant donné que c'est dans la Classe Controle que s'effectue cette opération, alors cette alerte sera mutée et considérée comme résolue en tant que faux positif. Il n 'y a pas de modification à faire.


Figure 35: Résolution de l'alerte de bug

Concernant la correction des Code Smell, la plupart sont corrigeable rapidement




Pour les autres alertes non récupérées dans L'IDE, il a été détecté plusieurs erreurs du même typedans diverses Classes du package metier( Abonnement, CommandeDocument

la Classe. Cette erreur considère la mise en forme des Getters et Setters des propriétés IdLivreDvd et NbExemplaires non conformes.

```
/// <summary>
/// Getter et Setter de la propriété IdLivreDvd autogénérés
/// </summary>
public string IdLivreDvd { get => idLivreDvd; set => idLivreDvd = value; }
```

**Make this an auto-implemented property and remove its backing field.** Why is this an issue? 8 days ago L34
Code Smell Minor Resolved (Fixed) Not assigned 5min effort Comment clumsy
Administrator autogenere 2 minutes ago

```
/// <summary>
/// Getter et Setter de la propriété NbExemplaire autogénérés
/// </summary>
public int NbExemplaire { get => nbExemplaire; set => nbExemplaire = value; }
```

**Make this an auto-implemented property and remove its backing field.** Why is this an issue? 8 days ago L39
Code Smell Minor Resolved (Fixed) Not assigned 5min effort Comment clumsy
Administrator autogenere 2 minutes ago

Les modifications peuvent sont directement faites dans l'IDE via le menu d'action rapide sur le clic droit. Ainsi, le code est dorénavant le suivant :

```
/// <summary>
/// Getter et Setter de la propriété IdLivreDvd autogénérés
/// </summary>
2 références
public string IdLivreDvd { get; set; }

/// <summary>
/// Getter et Setter de la propriété NbExemplaire autogénérés
/// </summary>
4 références
public int NbExemplaire { get; set; }
```

Finalement, il a été possible d'obtenir les notes suivantes :

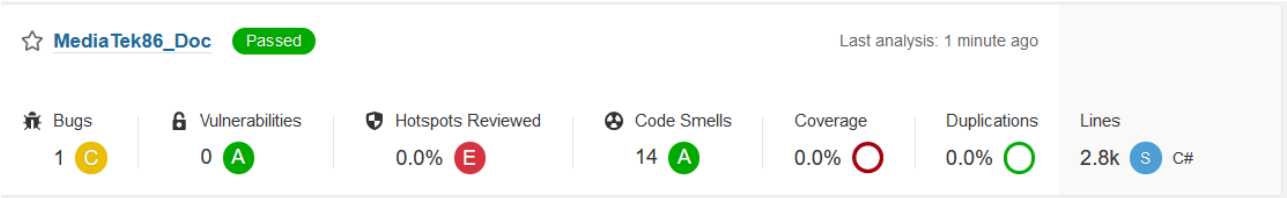
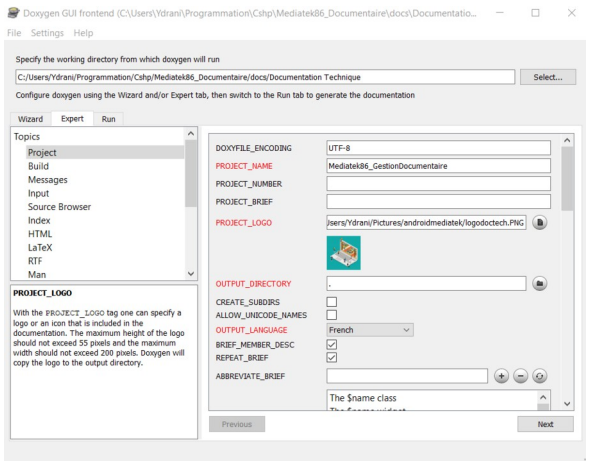



Figure 36: Mesures des différents points d'alerte de SonarQube

### Doctechnique

La documentation technique générée avec Doxyfile est visible à l'adresse suivante, en utilisant son utilitaire




# Déploiement

## Création de l'installateur

### Setup Wizard

La création de l'installeur de l'application a été effectué avec l'extension « Microsoft Visual Studio Installer Project » récupérable depuis le gestionnaire d'extension de l'IDE. Sa mise en place demande d'ajouter un nouveau projet type « Setup Wizard » dans la solution ici sous le nom de « Setup ».

Une fois le projet créé, un fenêtre de configuration « Choose a project type » s'ouvre à chaque étape, les options suivantes ont été sélectionné :

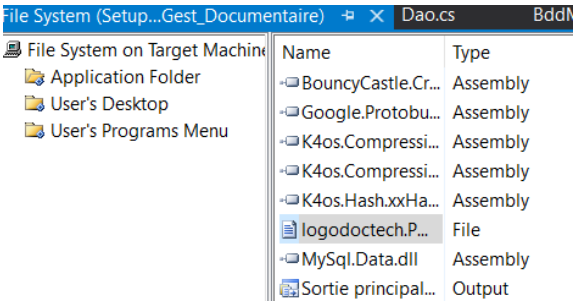
- Create a setup for a Windows application
- Sortie principale from Mediatek86


Figure 37: Etapes de configuration de seup wizard

Une fois le projet créé, il faut configurer le projet Setup_MediatekGest_Documentaire.

### Configurer Setup_MediatekGest_Documentaire

Dans le dossier Application Folder, un fichier logodoctech.ico a été ajouté, ce dernier permet de créer une icône personnalisée.




Enfin deux raccourcis sont créés depuis l'output **« Sortie principale from Mediatek »** qui ont été glissé respectivement dans les dossiers **« User's Desktop »** et **« User's Programs Menu »**.

Dans les propriétés de ces raccourcis, il est possible de les renommer afin de leur donner un nom correspondant au nom de l'application, **MediatekGest_Documentaire.**

Enfin dans la propriété **Icon**, il est possible de sélectionner le fichier ico préalablement déposer dans le dossier **Application Folder**.

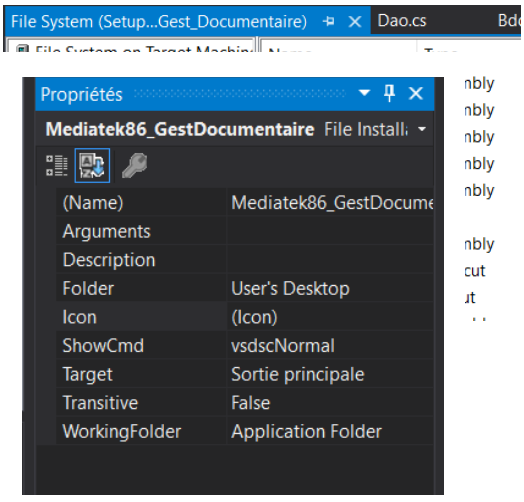
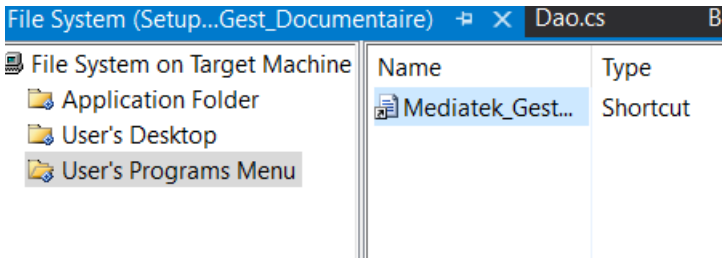



Figure 39: Mise en place des propriétés des raccourcis

Ainsi, nous avons dans les dossiers User's Desktop et User's Programs Menu le contenu suivant :



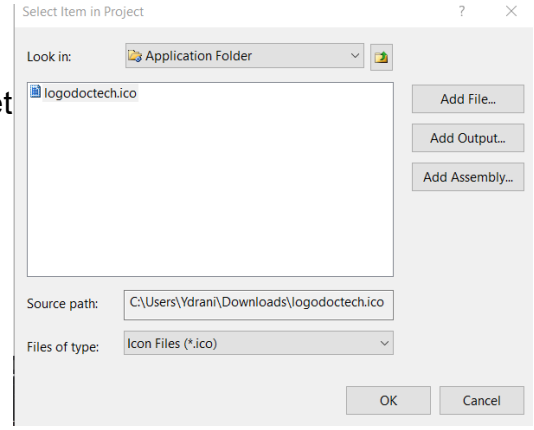




Figure 40: Exemple de contenu de setup

Enfin, il est possible de modifier les propriétés **Author** et **Manufacturer** du projet Setup. Cette dernière propriété permet de choisir le nom du dossier d'installation.

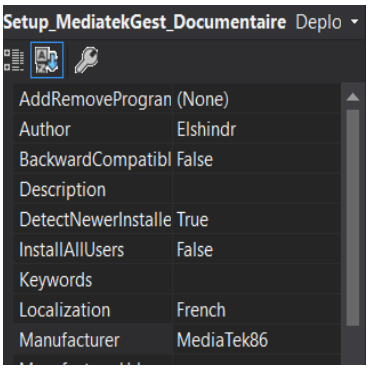



Figure 41: Propriétés du projet Setup_MediatekGest_Documentaire

Enfin dans l'onglet de l'IDE « générer », certaines modifications ont été effectués dans le gestionnaire de configurations. L'ensemble des checkbox de la colonne Générer ont été cochés pour les configurations de solutions actives Debug et Release.

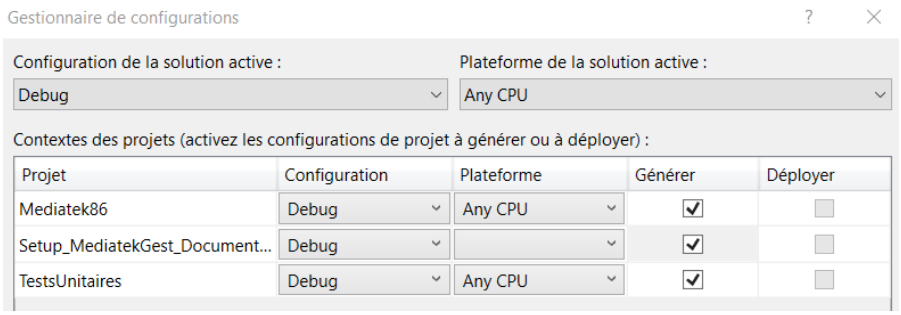



Figure 42: Gestionnaire de configuration de génération de projet

Enfin, il suffit de cliquer sur la solution du projet, puis de sélectionner « générer la solution ». Ainsi, l'installateur est créé dans le dossier du projet de la solution. Il convient de cliquer sur le fichier en **.msi** pour lancer l'installation.

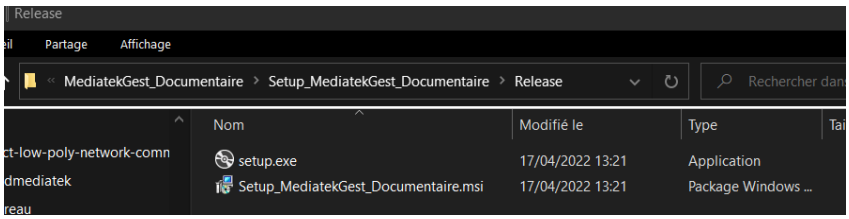



Figure 43: Chemin du dossier vers l'installateur de l'application

## Mise en ligne de la base de données

### Heroku

La plateforme de déploiement choisie est Heroku qui permet la mise en ligne de plateforme web. Pour mettre la base de donnée locale en ligne, il suffit de créer gratuitement un compte, de créer une nouvelle application et de le lier au compte du dépôt GitHub.

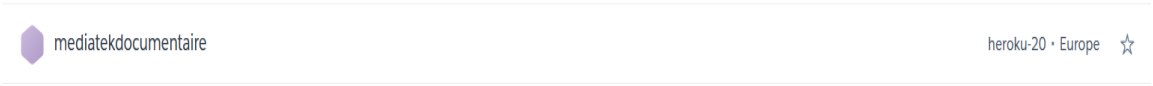



Figure 44: Fenêtre de liaison au dépot GitHib pour le déploiement sur Heroku

### JawsDB MySQL

La dernière étape consiste à mettre en place la base de donnée en ligne, pour cela la plateforme **Heroku** dispose de plusieurs add-ons dont **JawsDB MySQL** qui sera utilisé à cette intention. Pour cela, il suffit d'aller dans la liste des add-ons afin de trouver l'add-on, puis de choisir la formule concordant aux besoins de l'application ici «**Kitefin Shared** » qui est gratuite et suffisante pour les besoins actuels de l'application.

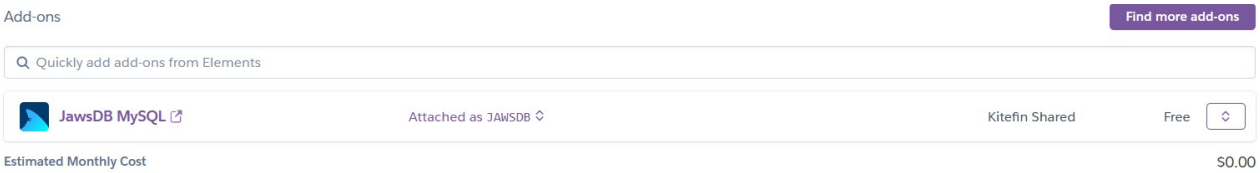



Figure 45: Fenêtre Heroku des addons

En cliquant sur «**JawsDB MySQL»**, une nouvelle s'ouvre dans laquelle il est possible de trouver les différentes informations de connexion à la base de donnée en cliquant sur le nom de la base mise à disposition, cf *figure*.

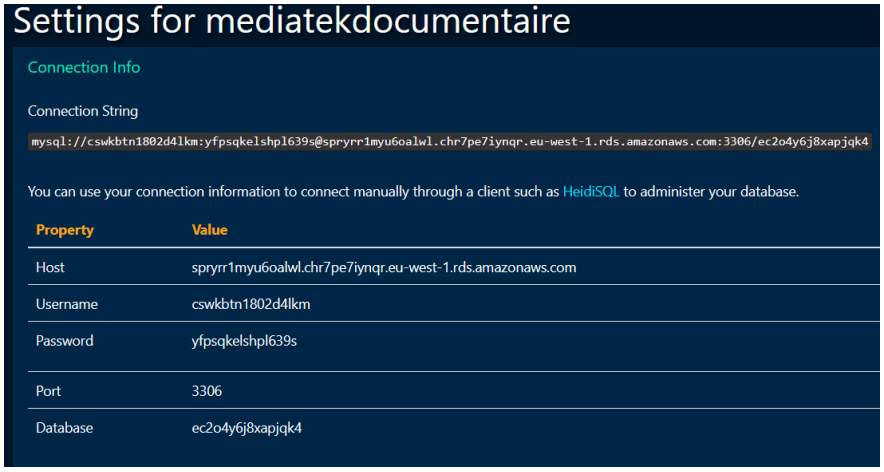



Figure 46: Fenêtre de ClearDB contenant les informations de connexion

Ces informations sont à remplacer avec celles qui définissaient l'accès local dans **Doa.cs,** cf figure

```
private static readonly string server = "spryrr1myu6oalwl.chr7pe7iynqr.eu-west-1.rds.amazonaws.com";
private static readonly string userid = "cswkbtn1802d4lkm";
private static readonly string password = "yfpsqkelshpl639s";
private static readonly string database = "ec2o4y6j8xapjqk4";
private static readonly string connectionString = "server=" + server + ";user id=" + userid + ";password=" + password + ";database=" + database + ";SslMode=no
```

Figure 47: Classe Doa.cs informations de connexion à la base de donnée distante

Enfin, il faut remplir la nouvelle base de donnée. Afin d'y accéder avec phpMyAdmin il est nécessaire de modifier son fichier config.inc.php, puis d'aller en bas du fichier afin de rajouter les données de connexion afin que le nouveau serveur soit disponible lors de la connexion à phpMyAdmin.

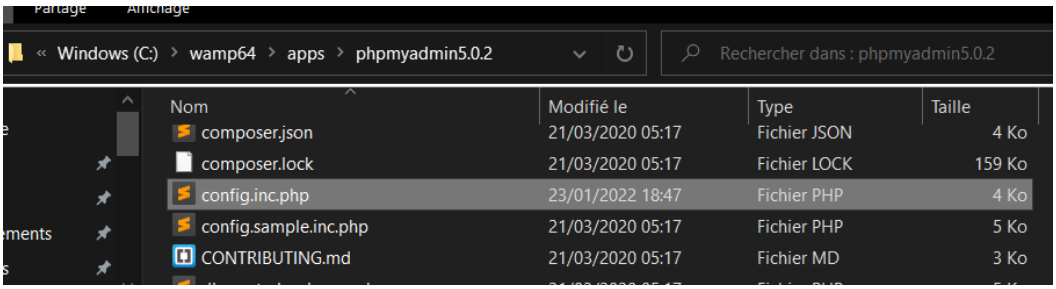



Figure 48: Localisation du fichier config.inc.php

Ainsi, il faut rajouter le fichier config.inc.php le contenu suivant :

```
$i++;
$cfg['Servers'][$i]['host'] = 'spryrr1myu6oalwl.chr7pe7iynqr.eu-west-
1.rds.amazonaws.com'; //hostname and port
$cfg['Servers'][$i]['user'] = 'cswkbtn1802d4lkm';   //user name
$cfg['Servers'][$i]['password'] = 'yfpsqkelshpl639s'; //password
$cfg['Servers'][$i]['auth_type'] = 'config';       //config
```

Après avoir sauvegardé ce fichier, il est alors possible d'accéder au serveur de la base de donnée via l'interface de connexion de phpMyAdmin. Enfin, la base de donnée peut être rempli comme précédemment via le menu d'importation de script SQL après avoir exporté la base modifiée, ici trouvable à ce lien.

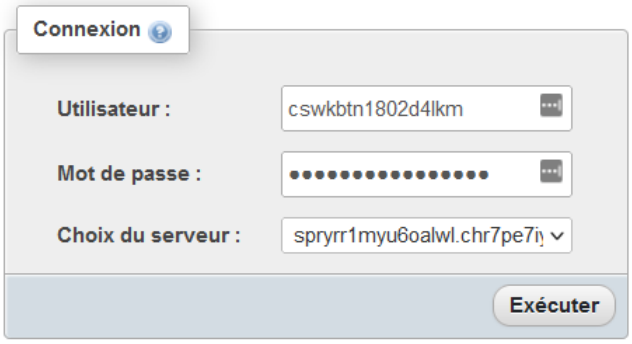



Figure 49: Interface de connexion à phpMyAdmin sur le nouveau serveur

# Bilan sur les objectifs atteints

## Finalités

Actuellement, l'évolution de l'application permet la gestion des commandes des différents types de documents : livres, dvd et revue, ainsi que l'apparition d'une alerte énumérant les fins d'abonnements. Aussi, la mise en place d'une authentification par le biais d'une fenêtre de connexion permet aux utilisateurs de se connecter à l'application et d'avoir uniquement accès aux éléments nécessaires à leur service d'affection Ces évolutions semblent fonctionnelles et respectueuses de l'expression des besoins. Aussi, des tests unitaires afin de pouvoir tester le suivi des futurs évolutions et des logs ont été mis en places afin de suivre les événements de l'application.

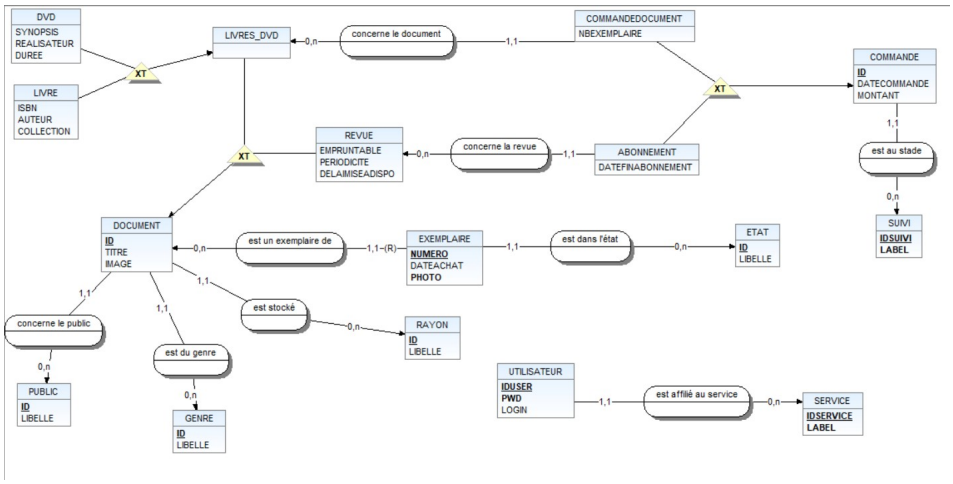



Figure 50: Schéma UML de la base de donnée finale

## Liste des compétences couvertes (B1, B2, B3)

### B1 Gérer le patrimoine informatique

- Recenser et identifier les ressources numériques
- Exploiter des référentiels, normes et standards adoptés par le prestataire informatique
- Vérifier les conditions de la continuité d'un service informatique

### B1 Répondre aux incidents et aux demandes d'assistance et d'évolution

- Traiter des demandes concernant les services réseau et système, applicatifs
- Traiter des demandes concernant les applications

### B1 Travailler en mode projet

- Analyser les objectifs et les modalités d'organisation d'un projet
- Planifier les activités
- Évaluer les indicateurs de suivi d'un projet et analyser les écarts

### B1 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 serv

### B1 Organiser son développement professionnel

- Mettre en place son environnement d'apprentissage personnel
- Mettre en œuvre des outils et stratégies de veille informationnelle
- Gérer son identité professionnelle
- Développer son projet professionnel

### B2 Concevoir et développer une solution applicative

- Participer à la conception de l'architecture d'une solution applicative
- Modéliser une solution applicative
- Exploiter les ressources du cadre applicatif (framework)
- Identifier, développer, utiliser ou adapter des composants logiciels
- Exploiter les technologies Web pour mettre en œuvre les échanges entre applications, y compris de mobilité
- Utiliser des composants d'accès aux données
- Intégrer en continu les versions d'une solution applicative
- Réaliser les tests nécessaires à la validation ou à la mise en production d'éléments adaptés ou développés
- Rédiger des documentations technique et d'utilisation d'une solution applicative

- Exploiter les fonctionnalités d' un environnement de développement et de tests

### B2 Assurer la maintenance corrective ou évolutive d'une solution applicative

- Recueillir, analyser et mettre à jour les informations sur une version d'une solution applicative
- Évaluer la qualité d'une solution applicative
- Analyser et corriger un dysfonctionnement
- Mettre à jour des documentations technique et d'utilisation de solution applicative

### B2 Gérer les données

- Développer des fonctionnalités applicatives au sein d'un système de gestion de base de données (relationnel ou non)
- Concevoir ou adapter une base de données
- Administrer et déployer une base de données

### B3 Assurer la cybersécurité d'une solution applicative et de son développement

- Participer à la vérification des éléments contribuant à la qualité d'un développement informatique
- Prendre en compte la sécurité dans un projet de développement d'une solution applicative
- Mettre en œuvre et vérifier la conformité d'une solution applicative et de son développement à un référentiel, une norme ou un standard de sécurité