COMPTE RENDU ATELIER 2

Table des matières

Contexte

Le réseau MediaTek86 regroupe plusieurs médiathèques dans le département de la Vienne. Afin de faciliter la gestion quotidienne des documents (livres, DVD, revues) et des commandes, une application de bureau a été développée pour les employés des services administratif et prêts.

Existant

La base de code existante conprenait :

- Une API REST en PHP permettant l'accès à la base de données MySQL, avec les fonctionnalités de consultation des livres, DVD, revues et exemplaires déjà implémentées
- Une application C# de bureau permettant la consultation des documents, avec les onglets Livres, DVD, Revues et Parutions des revues déjà fonctionnels

Objectifs du projet

Le projet consistait à compléter ces deux applications en ajoutant les fonctionnalités suivantes :

- Gestion complète des documents (ajout, modification, suppression)
- Gestion des commandes de livres, DVD et revues
- Suivi de l'état des exemplaires
- Mise en place d'un système d'authentification
- Sécurisation, qualité du code et logs
- Tests unitaires et fonctionnels
- Documentation technique et utilisateur
- Déploiement en ligne et sauvegarde automatisée

Langages et technologies utilisées

Côté application C#, le développement a été réalisé en C# sous Visual Studio, en utilisant le framework .NET Framework 4.6. Les bibliothèques Newtonsoft.Json et Serilog ont été utilisées respectivement pour la sérialisation JSON et la gestion des logs. Les tests unitaires ont été écrits avec le framework MSTest.

Côté API REST, le développement a été réalisé en PHP 8.3 sous NetBeans, en suivant une architecture REST avec une authentification Basic Auth. Le serveur local utilisé pour les tests est WampServer.

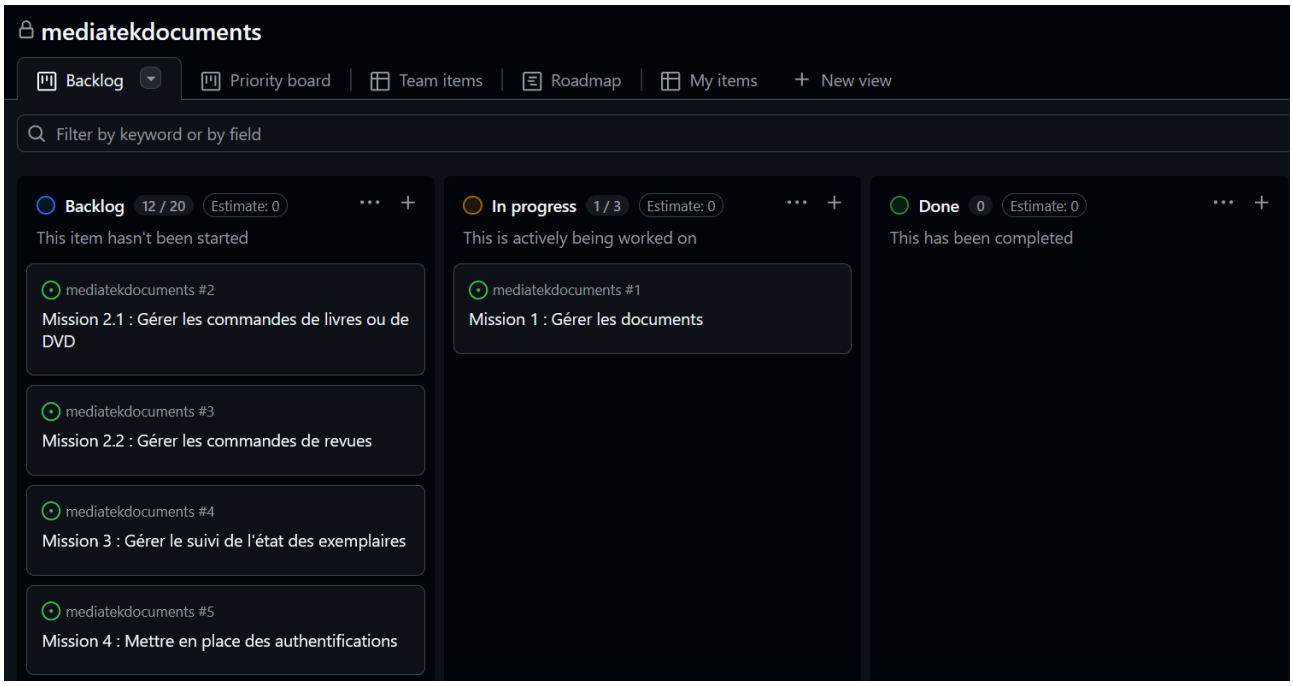
Pour la base de données, le SGBD utilisé est MySQL, géré via phpMyAdmin.

Concernant les outils et méthodes, le versioning a été géré avec GitHub, en utilisant les issues, les branches et les pull requests. La gestion de projet a été réalisée avec un Kanban GitHub. Les tests fonctionnels ont été réalisés avec Postman. La documentation technique a été générée avec Sandcastle pour le C# et phpDocumentor pour le PHP. Le déploiement a été réalisé chez OVH via FileZilla.

Mission 1 : Gérer les documents

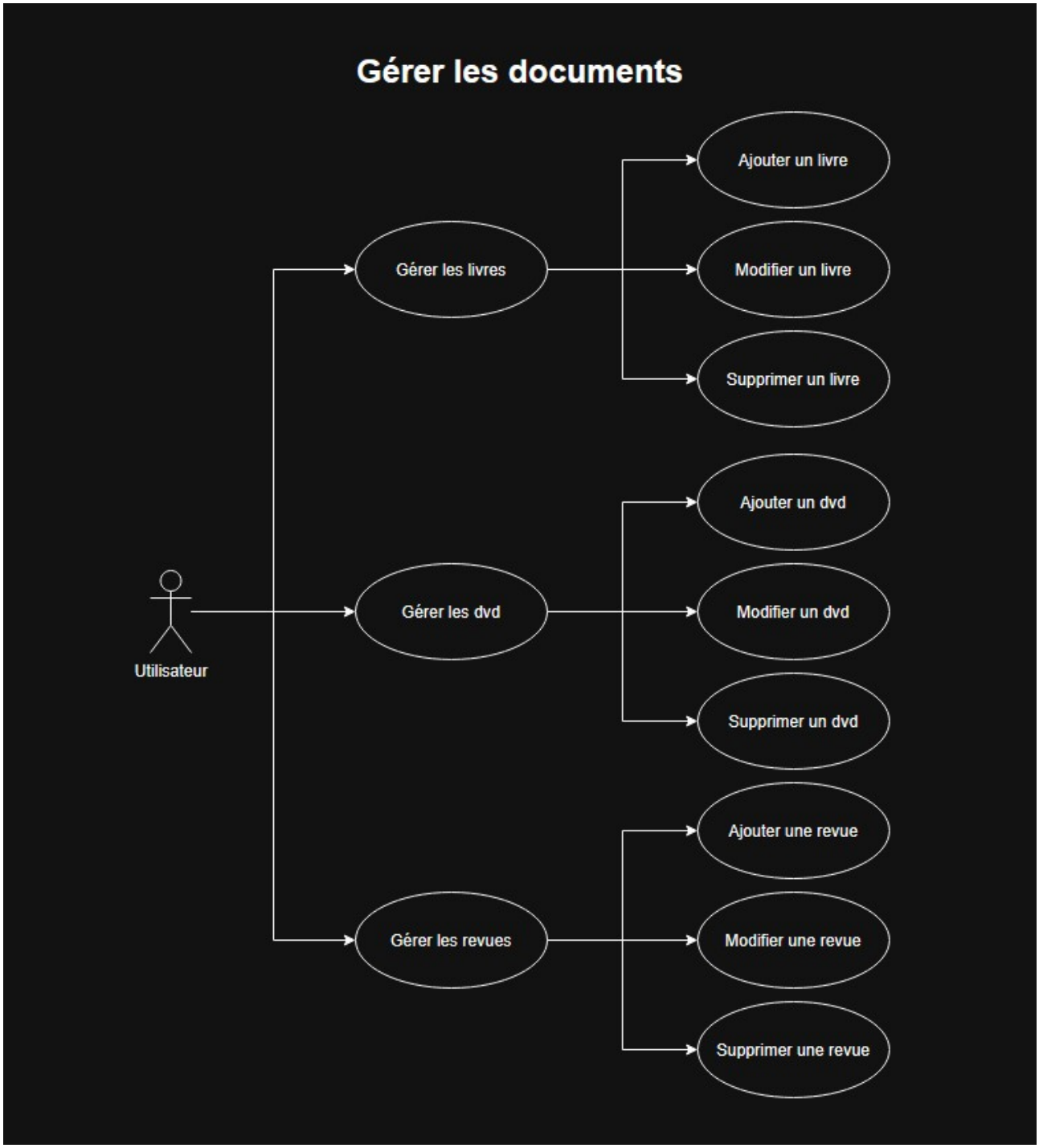
Demandes de la mission

Dans les onglets actuels (Livres, Dvd, Revues), ajouter les fonctionnalités (boutons) qui permettent d'ajouter, de modifier ou de supprimer un document. Un document ne peut être supprimé que s'il n'a pas d'exemplaire rattaché, ni de commandes. La modification d'un document ne peut pas porter sur son id. Toutes les sécurités seront mises en place pour éviter des erreurs de manipulation.



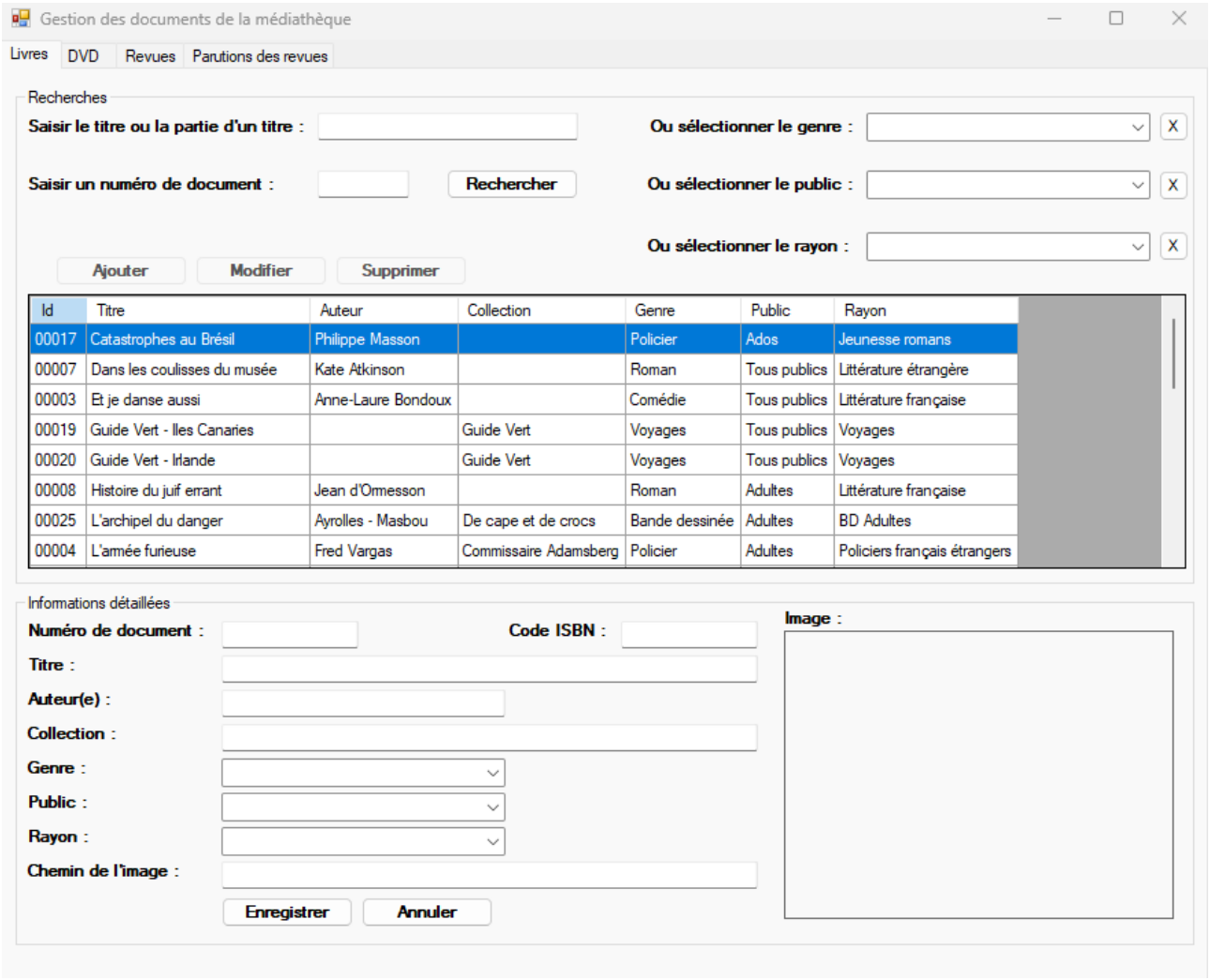
Temps de réalisation estimé : 8h
Temps de réalisation réel : 9h

Diagramme de cas d’utilisation



Aperçu des pages

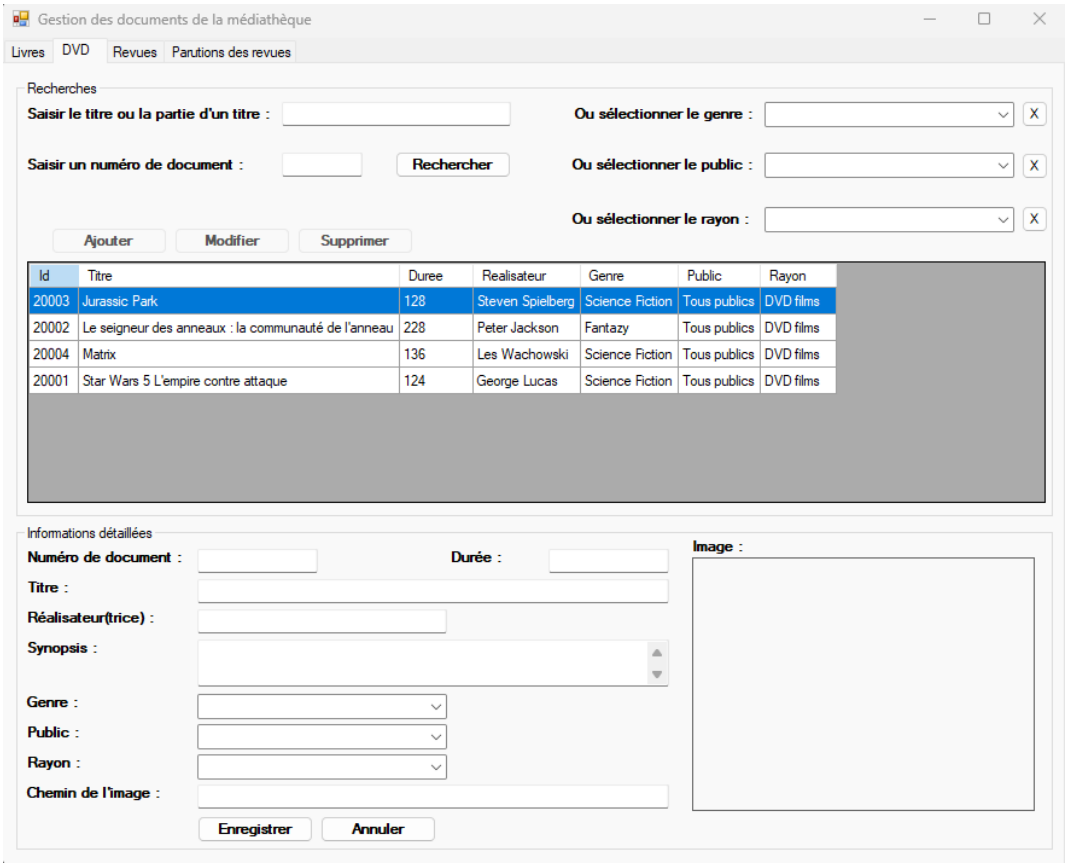
Voici à quoi ressemble la page Livres :

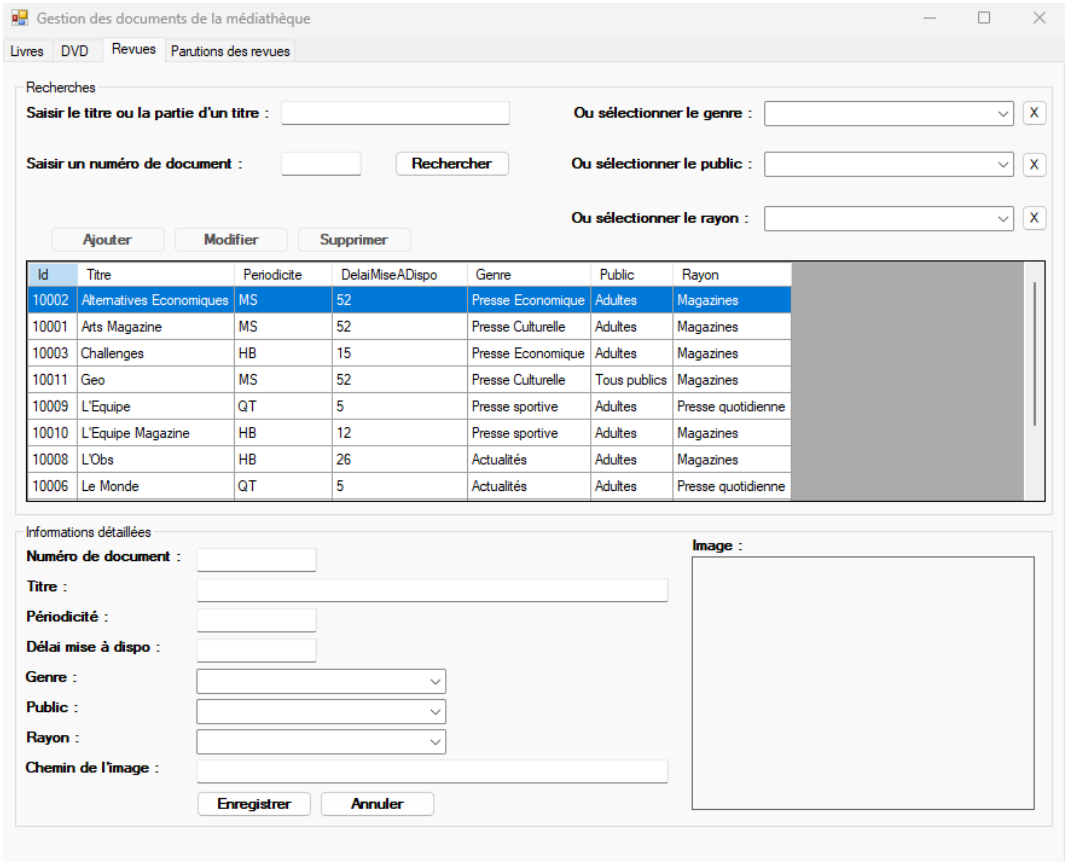


J’ai ajouté les boutons Ajouter, Modifier et Supprimer au dessus de la liste, et les boutons Enregistrer et Supprimer qui apparaissent sous le formulaire quand on clique sur Ajouter ou Modifier.

Le formulaire est vidé et passe en saisie quand on clique sur Ajouter, et il est remplis avec les informations de la ligne sélectionnée dans la liste et passe en saisie quand on clique sur Modifier. En cliquant sur Enregistrer, le livre / le DVD / la revue est ajouté à la base de données si on est en mode Ajout, et le livre / le DVD / la revue sélectionné est modifié en base de données si on est en mode Modification. Si on clique sur Annuler le formulaire est vidé et repasse en ReadOnly.

Les pages DVD et Revues :





Explication de code

MyAccessBDD.php :

Dans cette classe qui contient toutes les requêtes SQL, les méthodes suivantes ont été ajoutées :

```
private function insertLivre(?array $champs) : ?int{
```

Insère un livre dans les tables document, livres_dvd et livre en utilisant une transaction pour respecter la règle « tout ou rien ».

```
private function updateLivre(?string $id, ?array $champs) : ?int{
```

Modifie un livre dans les tables dpcument et livre en utilisant une transaction.

```
private function deleteLivre(?array $champs) : ?int{
```

Supprime un livre des tables livre, livres_dvd et document après avoir vérifié qu'il n'a pas d'exemplaire ni de commandes.

```
private function insertDvd(?array $champs) : ?int{
```

Insère un DVD dans les tables document, livres_dvd et dvd en utilisant une transaction.

```
private function updateDvd(?string $id, ?array $champs) : ?int{
```

Modifie un DVD dans les tables document et dvd en utilisant une transaction.

```
private function deleteDvd(?array $champs) : ?int{
```

Supprime un DVD dans les tables document et dvd en utilisant une transaction.

```
private function insertRevue(?array $champs) : ?int{
```

Insère une revue dans les tables document et revue en utilisant une transaction.

```
private function updateRevue(?string $id, ?array $champs) : ?int{
```

Modifie une revue dans les tables document et revue en utilisant une transaction.

```
private function deleteRevue(?array $champs) : ?int{
```

Supprime une revue des tables revue et document après avoir vérifié qu'elle n'a pas d'exemplaires ni d'abonnements.

Connexion.php :

```
public function beginTransaction() : void{
```

```
public function commit() : void{
```

```
public function rollBack() : void{
```

J’ai ajouté des trois méthodes pour permettre la gestion des transactions depuis MyAccessBDD.

Access.cs

J’ai ajouté ces méthodes qui envoient les requêtes POST, PUT ou DELETE à l’API et retournent true si l’opération a réussi et false si elle a échoué :

```
public bool CreerLivre(Livre livre)
public bool ModifierLivre(Livre livre)
public bool SupprimerLivre(Livre livre)
public bool CreerDvd(Dvd dvd)
public bool ModifierDvd(Dvd dvd)
public bool SupprimerDvd(Dvd dvd)
public bool CreerRevue(Revue revue)
public bool ModifierRevue(Revue revue)
public bool SupprimerRevue(Revue revue)
```

FrmMediatekController.cs :

J’ai ajouté ces méthodes qui font le lien entre la vue et la classe Access. Chaque méthode appelle simplement la méthode correspondante dans Access.

```
public bool CreerLivre(Livre livre)
public bool ModifierLivre(Livre livre)
public bool SupprimerLivre(Livre livre)
public bool CreerDvd(Dvd dvd)
public bool ModifierDvd(Dvd dvd)
public bool SupprimerDvd(Dvd dvd)
public bool CreerRevue(Revue revue)
public bool ModifierRevue(Revue revue)
public bool SupprimerRevue(Revue revue)
```

FrmMediatek.cs :

Les méthodes suivantes activent ou désactivent les champs de saisie selon si on est en mode lecture ou en mode saisie :

```
private void LivresChampsModifiables(bool actif)
private void DvdChampsModifiables(bool actif)
private void RevuesChampsModifiables(bool actif)
```

Les méthodes suivantes vident le formulaire et passent les champs en mode saisie quand on entre en mode Ajout :

```
private void btnLivresAjouter_Click(object sender, EventArgs e)
private void btnDvdAjouter_Click(object sender, EventArgs e)
private void btnRevuesAjouter_Click(object sender, EventArgs e)
```

Les méthodes suivantes activent les champs pour modifier le document sélectionné, en gardant le numéro en lecture seule :

```
private void btnLivresModifier_Click(object sender, EventArgs e)
private void btnDvdModifier_Click(object sender, EventArgs e)
private void btnRevuesModifier_Click(object sender, EventArgs e)
```

Les méthodes suivantes vérifient les champs obligatoires puis appellent le contrôleur pour créer ou modifier le document selon le mode en cours :

```
private void btnLivresEnregistrer_Click(object sender, EventArgs
e)
private void btnDvdEnregistrer_Click(object sender, EventArgs e)
private void btnRevuesEnregistrer_Click(object sender, EventArgs
e)
```

Les méthodes suivantes vident les champs et les repassent en lecture seule :

```
private void btnLivresAnnuler_Click(object sender, EventArgs e)
private void btnDvdAnnuler_Click(object sender, EventArgs e)
private void btnRevuesAnnuler_Click(object sender, EventArgs e)
```

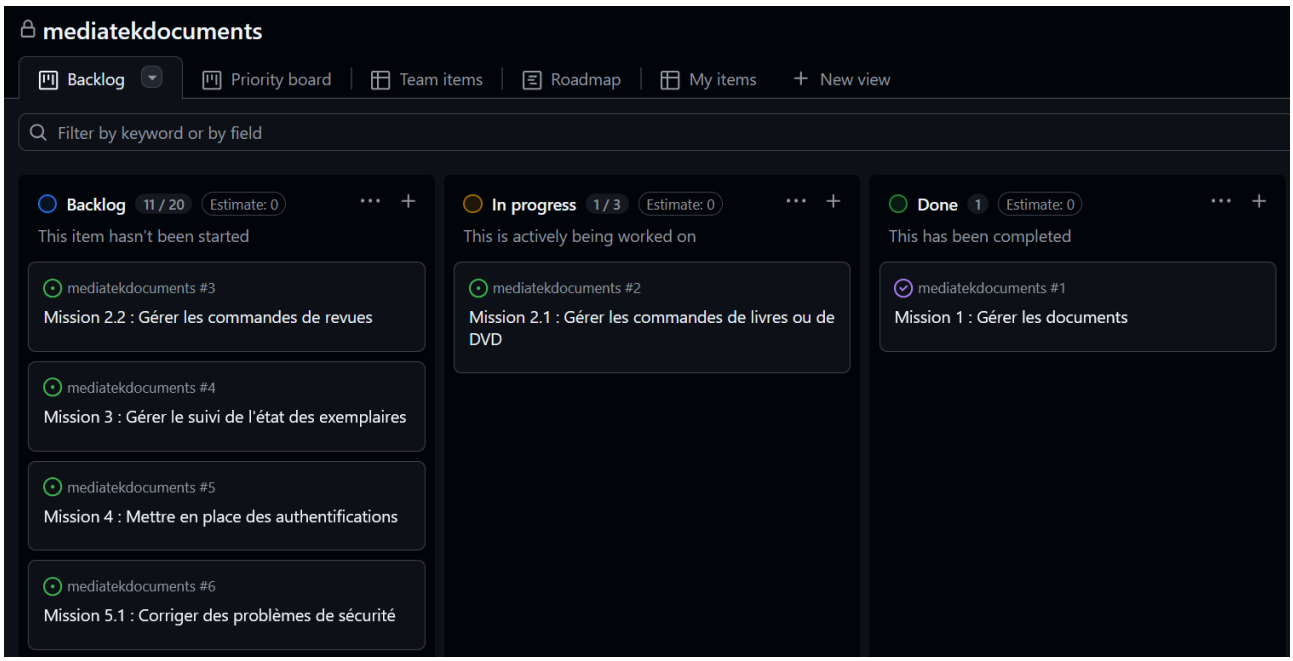
Les méthodes suivantes demandent confirmation puis appellent le contrôleur pour supprimer le document sélectionné :

```
private void btnLivresSupprimer_Click(object sender, EventArgs
e)
private void btnDvdSupprimer_Click(object sender, EventArgs e)
private void btnRevuesSupprimer_Click(object sender, EventArgs
e)
```

Mission 2.1 : Gérer les commandes de livres ou de DVD

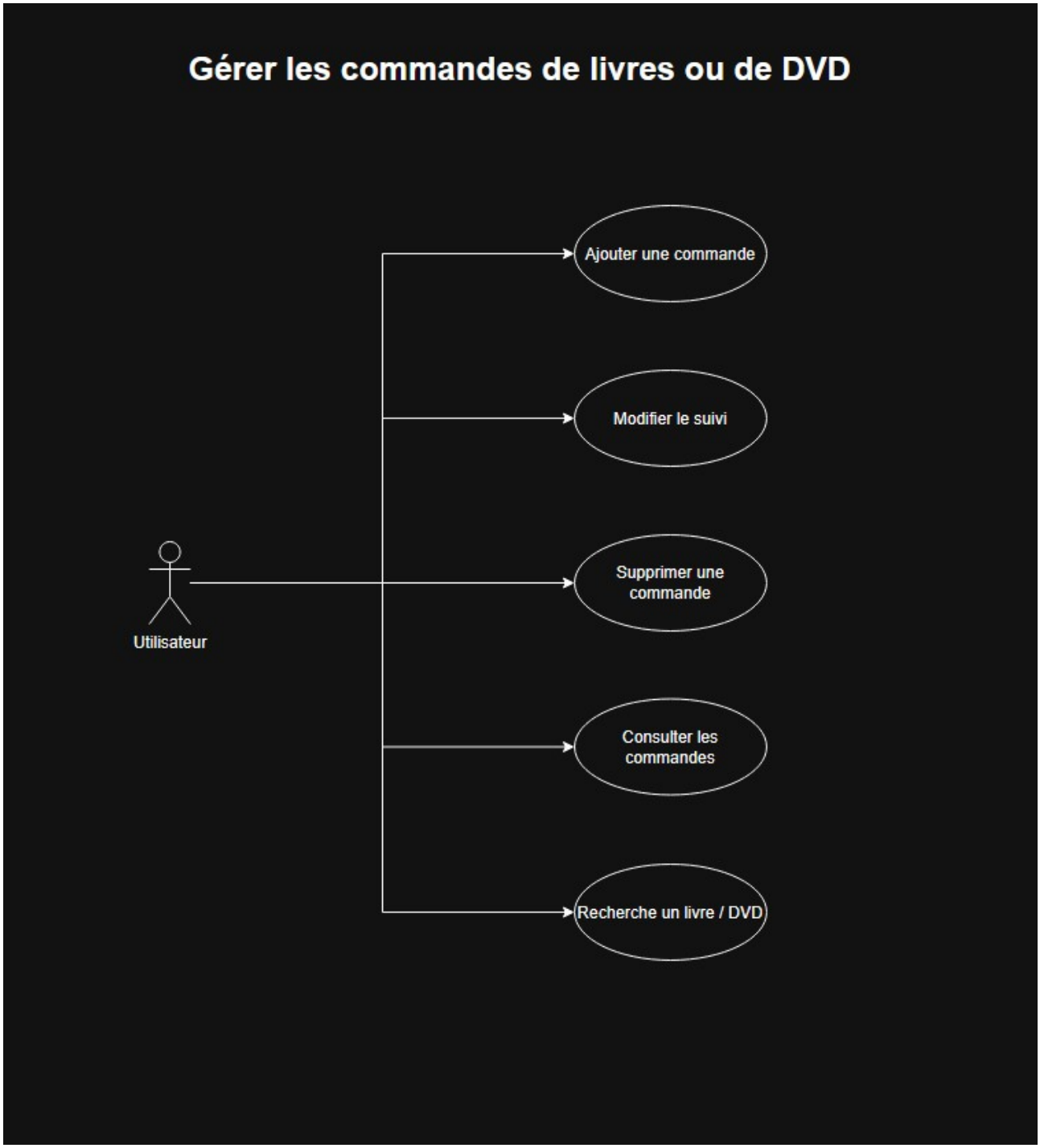
Demandes de la mission

- 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.
- Créer un onglet (ou une nouvelle fenêtre) pour gérer les commandes de livres.
- La charte graphique doit correspondre à l'existant.
- Dans toutes les listes, permettre le tri sur les colonnes.
- Dans l'onglet (ou la fenêtre), 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. Faire un trigger qui réalise aussi la suppression dans la classe fille.
- 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 en suivant la même logique que pour les commandes de livres.
- Toutes les sécurités seront mises en place pour éviter des erreurs de manipulation.



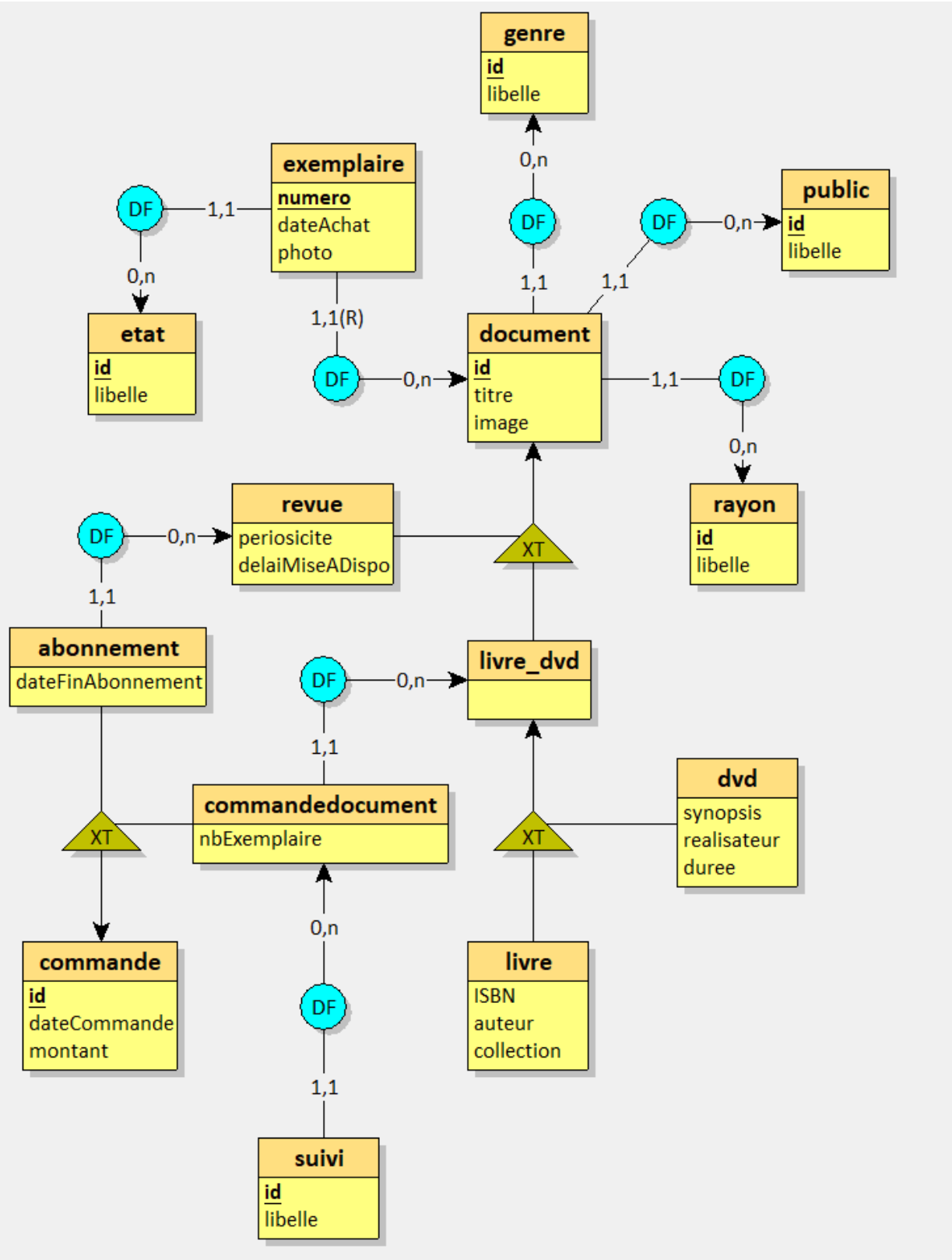
Temps de réalisation estimé : 8h
Temps de réalisation réel : 7h

Diagramme de cas d’utilisation



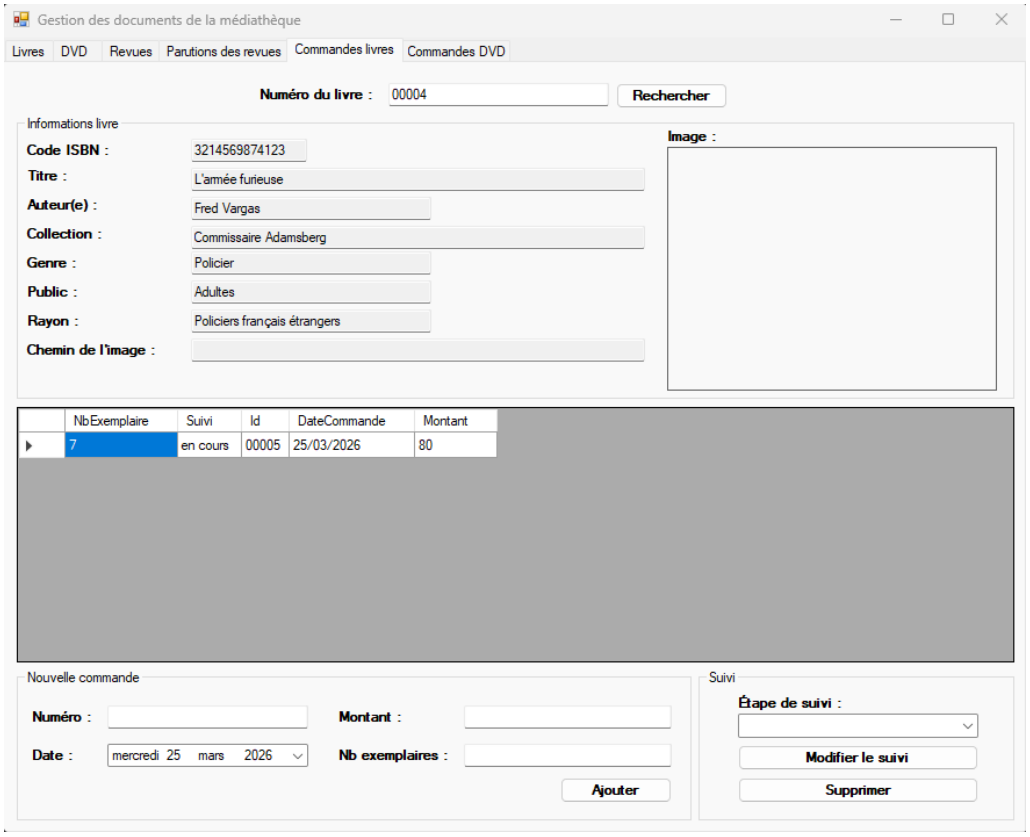
MCD

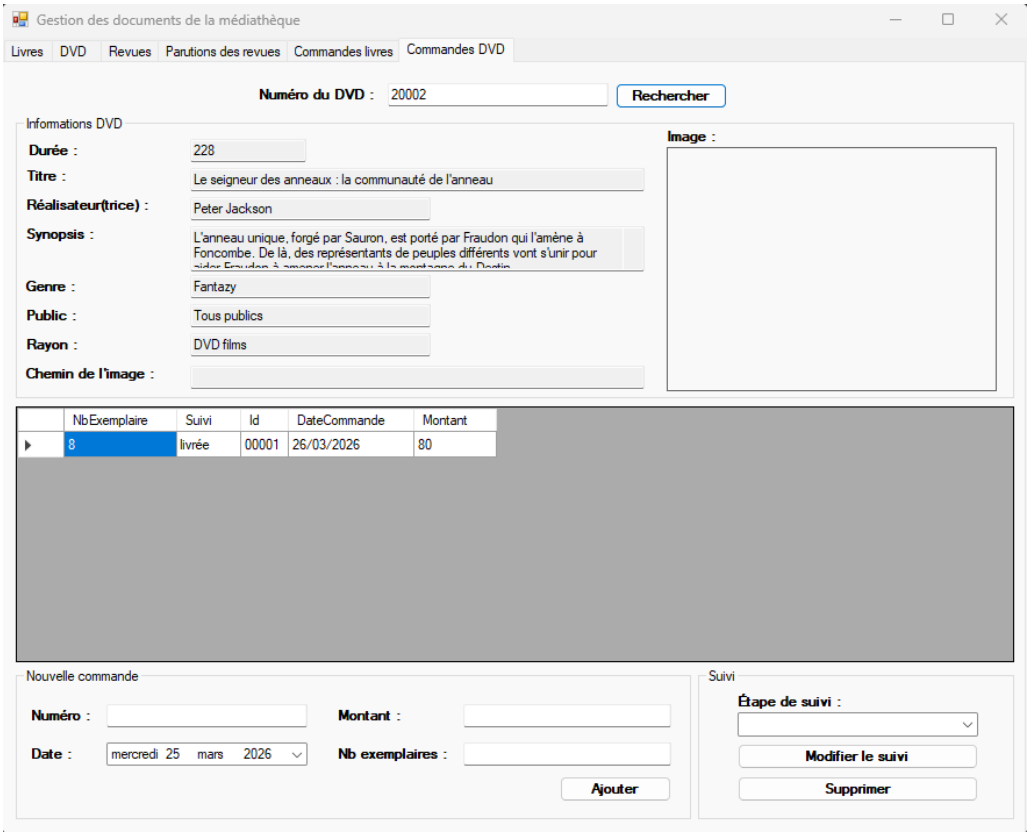
Pour ce qui est du MCD, il faut juste ajouter l'entité suivi, reliée à l'entité commandedocument (en bas) :



Nouvelles pages

Voici à quoi ressemble les pages Commandes livres et Commandes DVD :





L’utilisateur commence par chercher un livre / DVD en saisissant son numéro dans la barre de recherche en haut et clique sur le bouton Rechercher. Les informations du livre / DVD trouvé s’affichent, ainsi que les commandes existantes concernant ce livre / DVD. Il peut modifier le statut d’une commande ou bien la supprimer en bas à droite. Enfin, il peut créer une nouvelle commande en remplissant le formulaire du bas et en cliquant sur Ajouter.

Explication de code

Liste des classes impactées dans l’API :

MyAccessBDD.php :

```
private function selectCommandesDocument(?array $champs) : ?
array
```

Cette méthode récupère toutes les commandes d’un document.

```
private function insertCommandeDocument(?array $champs) : ?int
```

Cette méthode insère une commande dans les tables commande et commandedocument en utilisant une transaction. Le suivi est automatiquement mis à « en cours ».

```
private function updateSuiviCommande(?string $id, ?array
$champs) : ?int
```

Cette méthode modifie uniquement le champ idSuivi dans la table commandedocument.

```
private function deleteCommandeDocument(?array $champs) : ?int
```

Cette méthode supprime une commande dans les tables commande et commandedocument après avoir vérifié que son suivi n’est pas « livrée » ou « réglée ».

Liste des classe impactées dans l’application C# :

Du côté de l’application C#, j’ai créé 3 nouvelles classes : Commande.cs qui contient les informations communes à toutes les commandes, CommandeDocument.cs qui contient les informations spécifiques aux commandes de documents, et Suivi.cs qui représente une étape de suivi d’une commande.

```
using System;

namespace MediaTekDocuments.model
{
    /// <summary>
    /// Classe métier Commande : contient les informations d'une commande
    /// </summary>
    3 références | 0 modification | 0 auteur, 0 modification
    public class Commande
    {
        7 références | 0 modification | 0 auteur, 0 modification
        public string Id { get; }
        5 références | 0 modification | 0 auteur, 0 modification
        public DateTime DateCommande { get; }
        5 références | 0 modification | 0 auteur, 0 modification
        public double Montant { get; }

        1 référence | 0 modification | 0 auteur, 0 modification
        public Commande(string id, DateTime dateCommande, double montant)
        {
            Id = id;
            DateCommande = dateCommande;
            Montant = montant;
        }
    }
}
```

```
using System;

namespace MediaTekDocuments.model
{
    /// <summary>
    /// Classe métier CommandeDocument hérite de Commande :
    /// contient les informations d'une commande de document (livre ou dvd)
    /// </summary>
    43 références | 0 modification | 0 auteur, 0 modification
    public class CommandeDocument : Commande
    {
        5 références | 0 modification | 0 auteur, 0 modification
        public int NbExemplaire { get; }
        7 références | 0 modification | 0 auteur, 0 modification
        public string IdLivreDvd { get; }
        7 références | 0 modification | 0 auteur, 0 modification
        public string IdSuivi { get; }
        3 références | 0 modification | 0 auteur, 0 modification
        public string Suivi { get; }

        4 références | 0 modification | 0 auteur, 0 modification
        public CommandeDocument(string id, DateTime dateCommande, double montant,
            int nbExemplaire, string idLivreDvd, string idSuivi, string suivi)
            : base(id, dateCommande, montant)
        {
            NbExemplaire = nbExemplaire;
            IdLivreDvd = idLivreDvd;
            IdSuivi = idSuivi;
            Suivi = suivi;
        }
    }
}
```

```
namespace MediaTekDocuments.model
{
    /// <summary>
    /// Classe métier Suivi : hérite de Categorie
    /// </summary>
    3 références | 0 modification | 0 auteur, 0 modification
    public class Suivi : Categorie
    {
        0 références | 0 modification | 0 auteur, 0 modification
        public Suivi(string id, string libelle) : base(id, libelle)
        {
        }
    }
}
```

Access.cs :

J’ai créé les méthodes suivantes qui envoient les requêtes GET, POST, PUT ou DELETE à l’API et retournent les résultats correspondants :

```
public List<Categorie> GetAllSuivis()
public List<CommandeDocument> GetCommandesDocument(string
idLivreDvd)
public bool CreerCommandeDocument(CommandeDocument commande)
public bool ModifierSuiviCommande(CommandeDocument commande)
public bool SupprimerCommandeDocument(CommandeDocument commande)
```

FrmMediatekController.cs :

J’ai créé les méthodes suivantes pour faire le lien entre la vue et la classe Access :

```
public List<Categorie> GetAllSuivis()
public List<CommandeDocument> GetCommandesDocument(string
idLivreDvd)
public bool CreerCommandeDocument(CommandeDocument commande)
public bool ModifierSuiviCommande(CommandeDocument commande)
public bool SupprimerCommandeDocument(CommandeDocument commande)
```

FrmMediatek.cs :

J’ai ajouté deux nouvelles régions : #region Onglet Commandes Livres et #region Onglet Commanes DVD. Les méthodes suivantes ont été ajoutées pour chaque onglet :

```
private void tabCommandesLivres_Enter(object sender, EventArgs
e)
private void tabCommandesDvd_Enter(object sender, EventArgs e)
```

Ces méthodes remplissent le combobox de suivi et vident les champs à l’ouverture de l’onglet.

```
private void VideCommandesLivresInfos()
private void VideCommandesDvdInfos()
```

Ces méthodes vident toutes les zones d’affichage et le datagrid.

```
private void AfficheCommandesLivresInfos(Livre livre)
private void AfficheCommandesDvdInfos(Dvd dvd)
```

Ces méthodes affichent les informations du document sélectionné.

```
private void RemplirCommandesLivresListe(List<CommandeDocument>
commandes)
private void RemplirCommandesDvdListe(List<CommandeDocument>
commandes)
```

Ces méthodes recherchent un document par son numéro et affichent ses commandes.

```
private void btnCommandesLivresAjouter_Click(object sender,
EventArgs e)
private void btnCommandesDvdAjouter_Click(object sender,
EventArgs e)
```

Ces méthodes vérifient les champs obligatoires puis créent une nouvelle commande avec le suivi « en cours ».

```
private void btnCommandesLivresSuivi_Click(object sender,
EventArgs e)
private void btnCommandesDvdSuivi_Click(object sender, EventArgs
e)
```

Ces méthodes vérifient les règles de suivi puis modifient l’étape de la commande sélectionnée.

```
private void btnCommandesLivresSupprimer_Click(object sender,
EventArgs e)
private void btnCommandesDvdSupprimer_Click(object sender,
EventArgs e)
```

Ces méthodes demandent confirmation puis suppriment la commande sélectionnée si elle n’est pas livrée ou réglée.

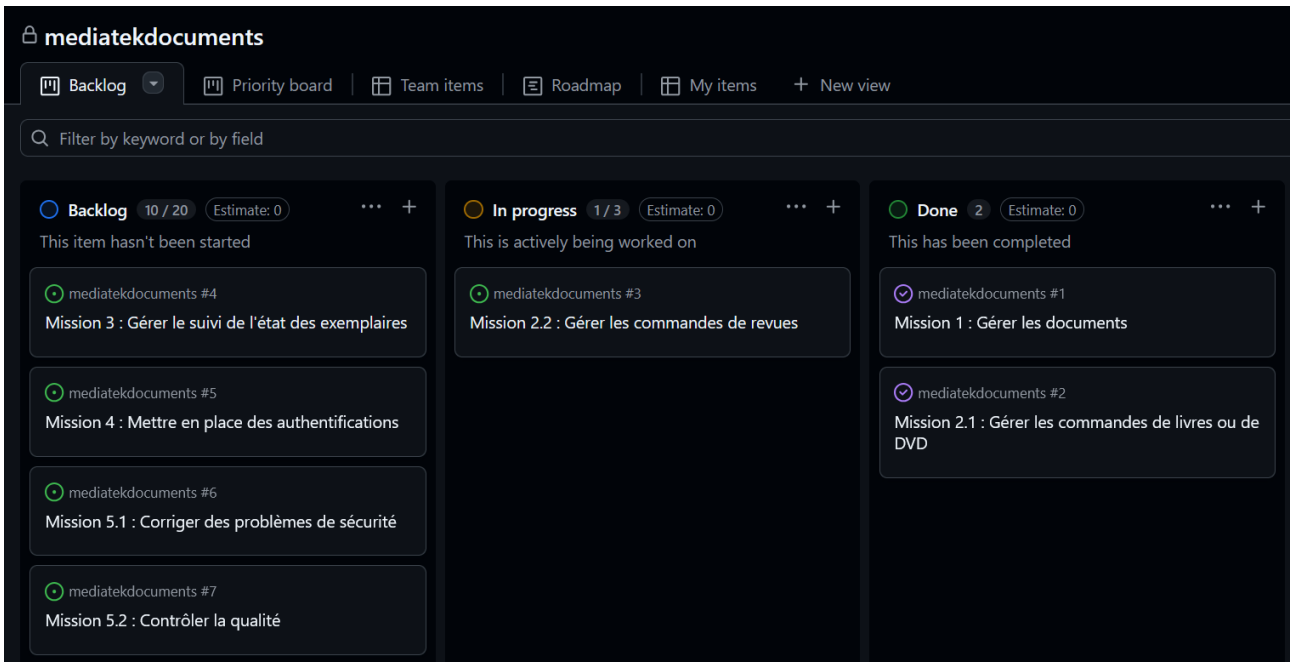
```
private void dgvCommandesLivres_ColumnHeaderMouseClick(object
sender, DataGridViewCellMouseEventArgs e)
private void dgvCommandesDvd_ColumnHeaderMouseClick(object
sender, DataGridViewCellMouseEventArgs e)
```

Ces méthodes permettent le tri sur les colonnes du datagrid.

Mission 2.2 : Gérer les commandes de revues

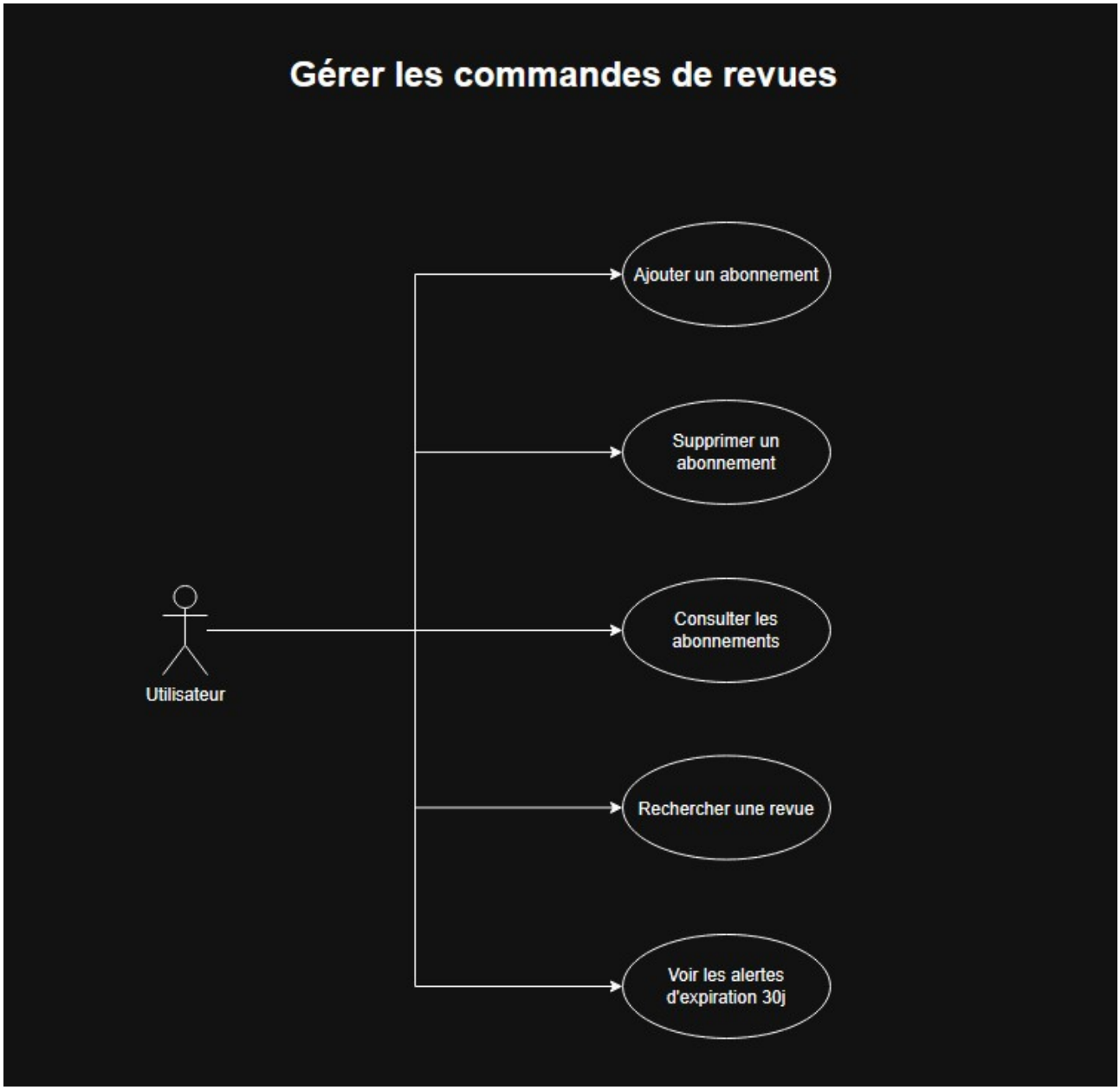
Demandes de la mission

- Créer un onglet (ou une fenêtre) 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.
- La charte graphique doit correspondre à l'existant.
- Dans toutes les listes, permettre le tri sur les colonnes.
- 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.
- Permettre de supprimer une commande de revue uniquement 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.
- Toutes les sécurités seront mises en place pour éviter des erreurs de manipulation.
- Créer une méthode 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.



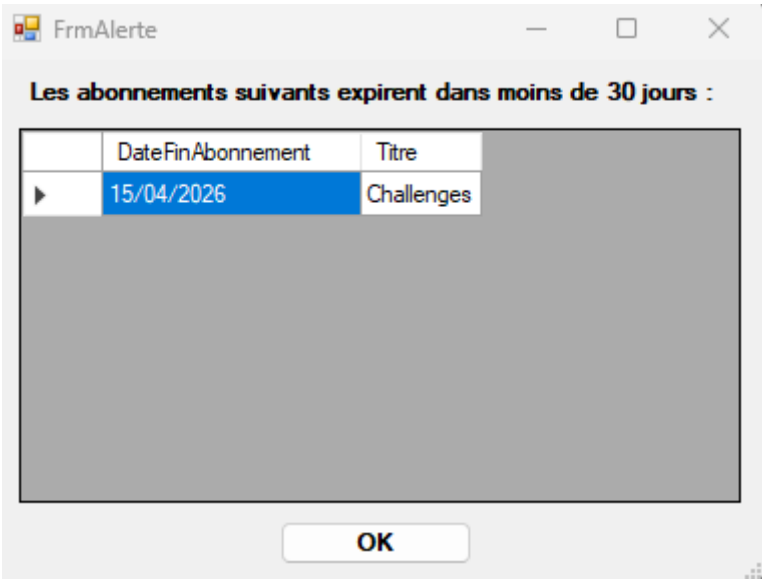
Temps de réalisation estimé : 4h
Temps de réalisation réel : 3h30

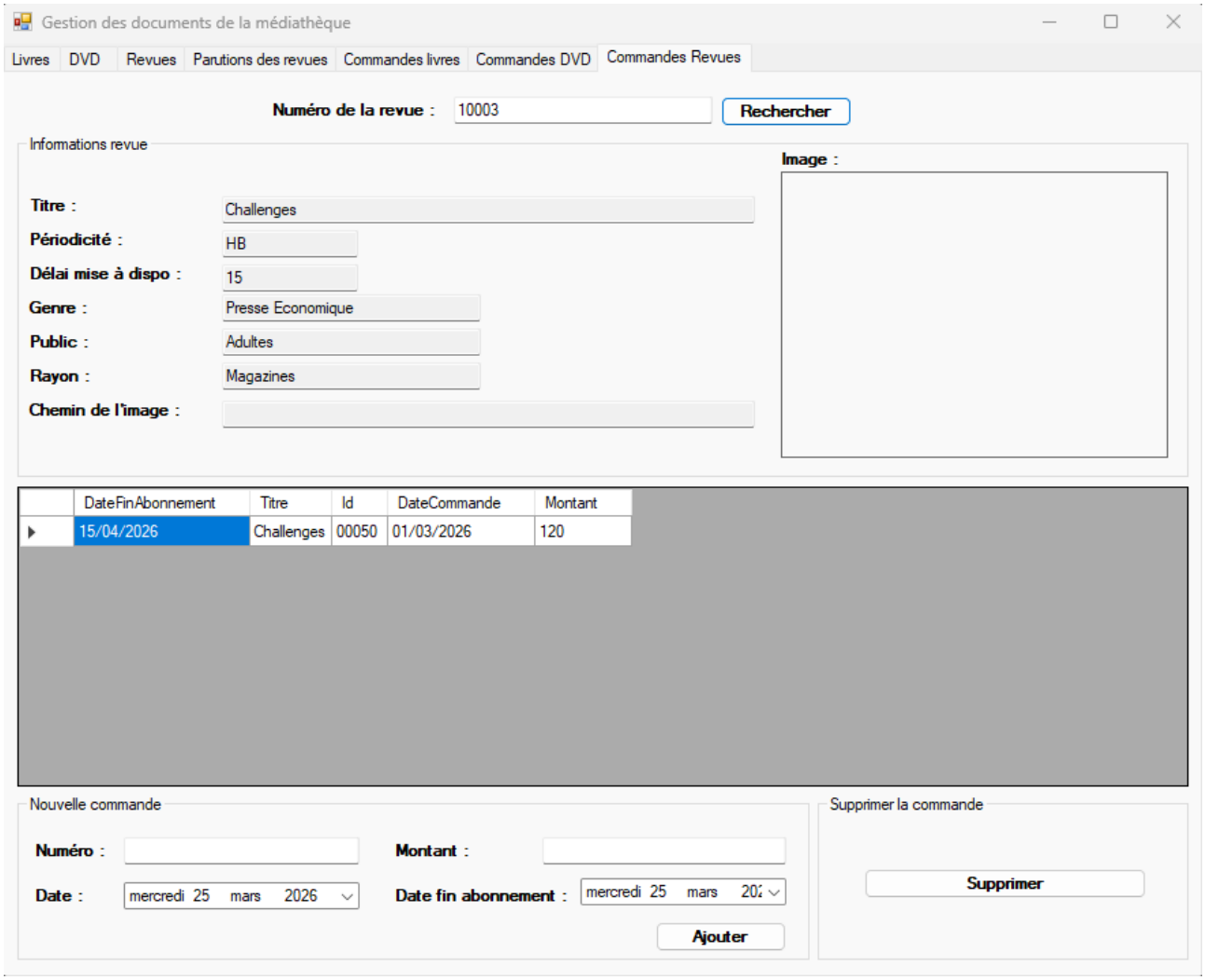
Diagramme de cas d’utilisation



Nouvelles pages

Voici à quoi ressemble la fenêtre d'alerte ainsi que la page Commandes revues :





L’utilisateur voit une fenêtre d’alerte au lancement qui lui montre les abonnements qui expirent dans moins de 30 jours. En cliquant sur OK il accède à l’application. Dans l’onglet Commandes revues il peut rechercher une revue en saisissant sons numéro. Les informations concernant cette revue s’affichent, ainsi que les abonnements existants. Il peut alors ajouter ou renouveler un abonnement en remplissant le formulaire en bas. Il peut également supprimer un abonnement en cliquant sur le bouton en bas à droite.

Explication de code

Liste des classes impactées dans l’API :

MyAccessBDD.php :

```
private function selectAbonnementsRevue(?array $champs) : ?array
```

Cette méthode récupère tous les abonnements d’une revue.

```
private function selectAbonnementsExpirantBientot() : ?array
```

Cette méthode récupère les revues dont l’abonnement se termine dans moins de 30 jours.

```
private function insertAbonnement(?array $champs) : ?int
```

Cette méthode insère un abonnement dans les tables commande et abonnement en utilisant une transaction.

```
private function deleteAbonnement(?array $champs) : ?int
```

Cette méthode supprime un abonnement dans les tables abonnement et commande en utilisant une transaction.

Liste des classes impactées dans l'application C# :

J'ai créé la classe Abonnement.cs qui contient les informations d'un abonnement :

```
using System;

namespace MediaTekDocuments.model
{
    /// <summary>
    /// Classe métier Abonnement hérite de Commande :
    /// contient les informations d'un abonnement à une revue
    /// </summary>
    29 références | 0 modification | 0 auteur, 0 modification
    public class Abonnement : Commande
    {
        3 références | 0 modification | 0 auteur, 0 modification
        public DateTime DateFinAbonnement { get; }
        3 références | 0 modification | 0 auteur, 0 modification
        public string IdRevue { get; }
        1 référence | 0 modification | 0 auteur, 0 modification
        public string Titre { get; }

        1 référence | 0 modification | 0 auteur, 0 modification
        public Abonnement(string id, DateTime dateCommande, double montant,
            DateTime dateFinAbonnement, string idRevue, string titre = "")
            : base(id, dateCommande, montant)
        {
            DateFinAbonnement = dateFinAbonnement;
            IdRevue = idRevue;
            Titre = titre;
        }
    }
}
```

Access.cs :

J'ai créé les méthodes suivantes qui envoient les requêtes GET, POST, PUT ou DELETE à l'API et retournent les résultats correspondants :

```
public List<Abonnement> GetAbonnementsRevue(string idRevue)
public bool CreerAbonnement(Abonnement abonnement)
public bool SupprimerAbonnement(Abonnement abonnement)
public List<Abonnement> GetAbonnementsExpirantBientot()
```

FrmMediatekController.cs :

J’ai créé les méthodes suivantes pour faire le lien entre la vue et la classe Access :

```
public List<Abonnement> GetAbonnementsRevue(string idRevue)
public bool CreerAbonnement(Abonnement abonnement)
public bool SupprimerAbonnement(Abonnement abonnement)
public List<Abonnement> GetAbonnementsExpirantBientot()
```

FrmAlerte.cs :

J’ai créé un nouveau formulaire FrmAlerte.cs qui s’ouvre au démarrage de l’application. Il contient les méthodes suivantes :

```
public FrmAlerte(List<Abonnement> lesAbonnements)
private void RemplirListe(List<Abonnement> lesAbonnements)
private void btnAlerteOK_Click(object sender, EventArgs e)
```

FrmMediatek.cs :

La méthode suivante est appelée dans le constructeur pour récupérer les abonnements qui expirent bientôt et ouvrir la fenêtre d’alerte si la liste n’est pas vide :

```
private void AfficheAlerteAbonnements()
```

J’ai ajouté une nouvelle région : #region Onglet Commandes Revues
Cette région contient les méthodes suivantes qui gèrent la recherche, l’affichage, l’ajout et la suppression d’abonnements.

```
private void tabCommandesRevues_Enter(object sender, EventArgs
e)
private void VideCommandesRevuesInfos()
private void VideCommandesRevuesSaisie()
private void AfficheCommandesRevuesInfos(Revue revue)
private void RemplirCommandesRevuesListe(List<Abonnement>
abonnements)
private void btnCommandesRevuesRechercher_Click(object sender,
EventArgs e)
private void btnCommandesRevuesAjouter_Click(object sender,
EventArgs e)
private void btnCommandesRevuesSupprimer_Click(object sender,
EventArgs e)
```

```
private void dgvCommandesRevues_ColumnHeaderMouseClick(object
sender, DataGridViewCellMouseEventArgs e)
```

J’ai créé une classe séparée GestionAbonnement.cs pour y mettre la méthode suivante car ça permet de la tester unitairement sans dépendance à l’interface graphique :

```
public bool ParutionDansAbonnement(DateTime dateCommande,
DateTime dateFinAbonnement, DateTime dateParution)
```

MediaTekDocuments.Tests :

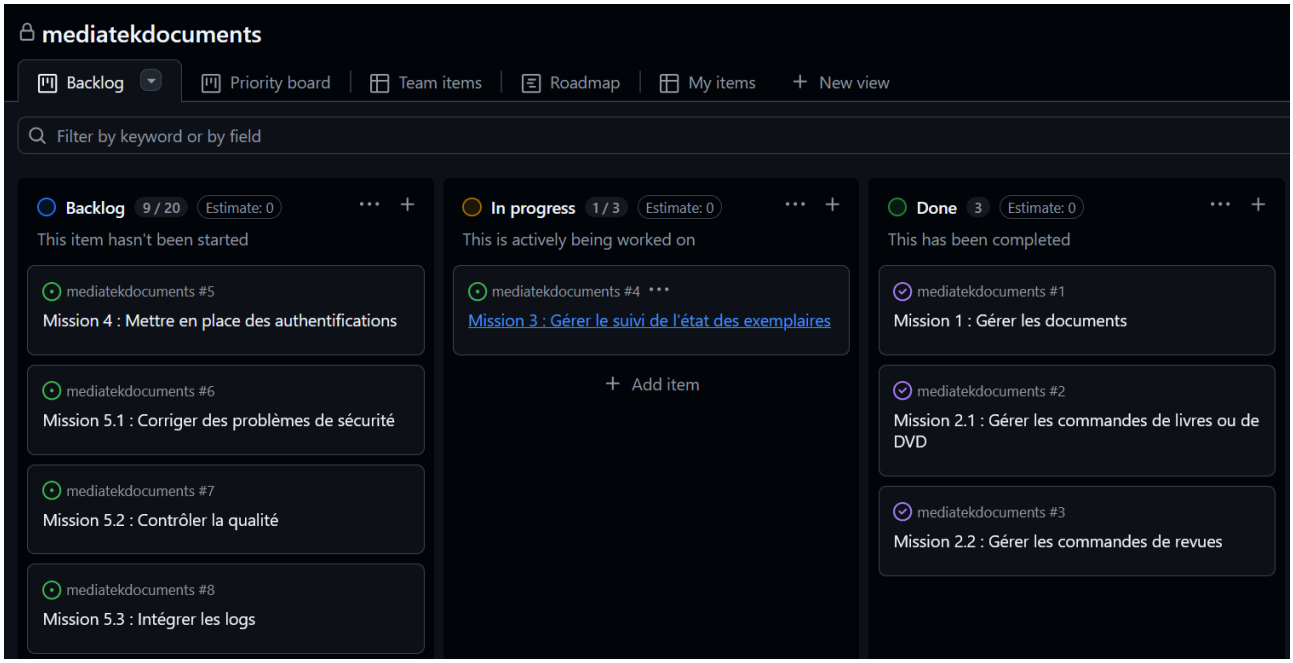
J’ai créé les tests unitaires suivants pour tester la méthode ParutionDansAbonnement() avec une date dans la période, une date avant la période et une date après la période :

```
public void ParutionDansAbonnement_DateDedans_RetourneTrue()
public void ParutionDansAbonnement_DateAvant_RetourneFalse()
public void ParutionDansAbonnement_DateApres_RetourneFalse()
```

Mission 3 : Gérer le suivi de l’état d’un exemplaires

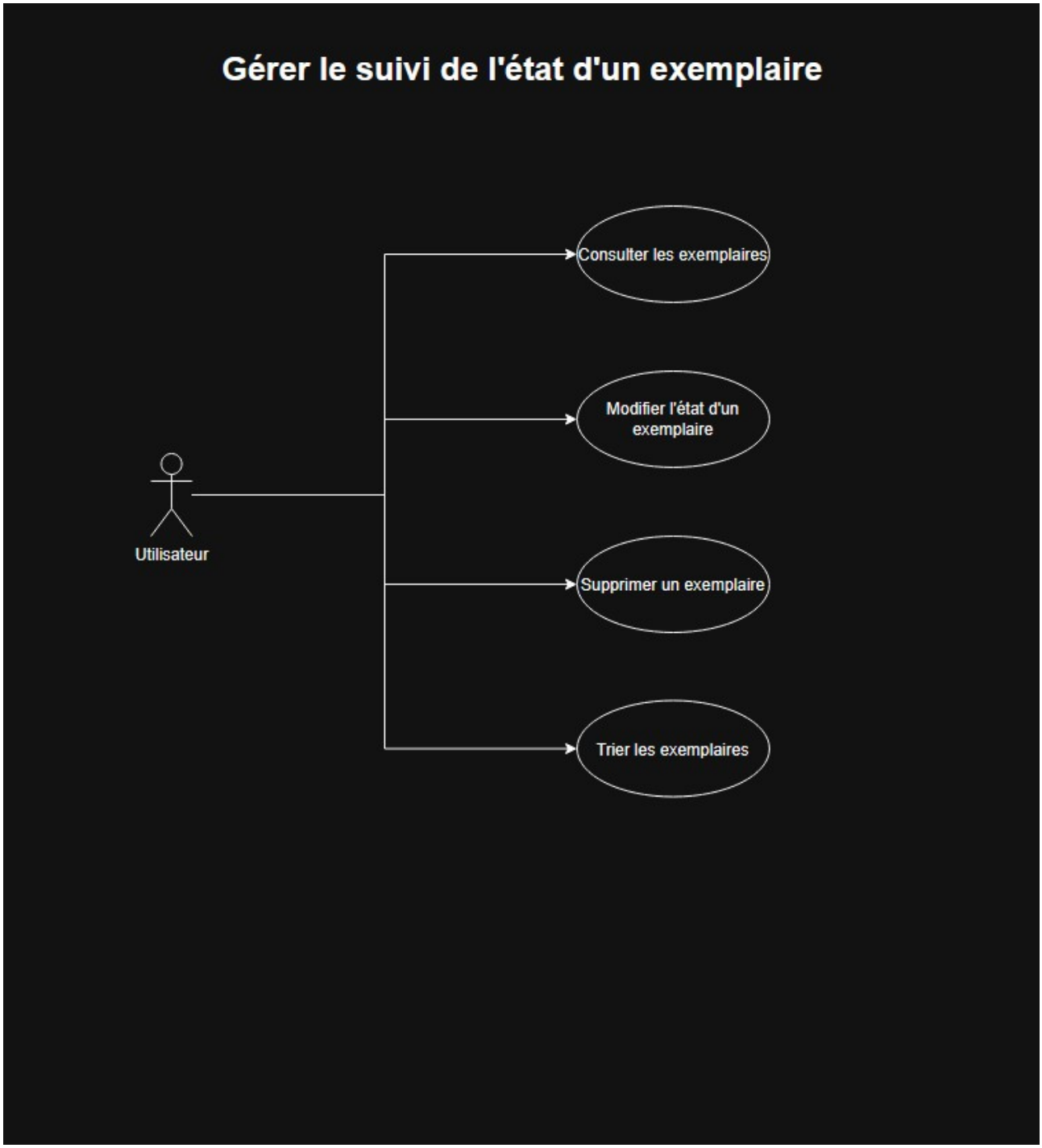
Demandes de la mission

- Agrandir la fenêtre en hauteur.
- Dans l'onglet Livres, partie basse, ajouter la liste des exemplaires du livre sélectionné. Cette liste d'exemplaires doit contenir les colonnes suivantes : 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.
- Dans l'onglet "Parutions des revues", liste des parutions, remplacer la colonne "Photo" par "Etat". Permettre aussi le changement d'état.
- Permettre de supprimer un exemplaire.



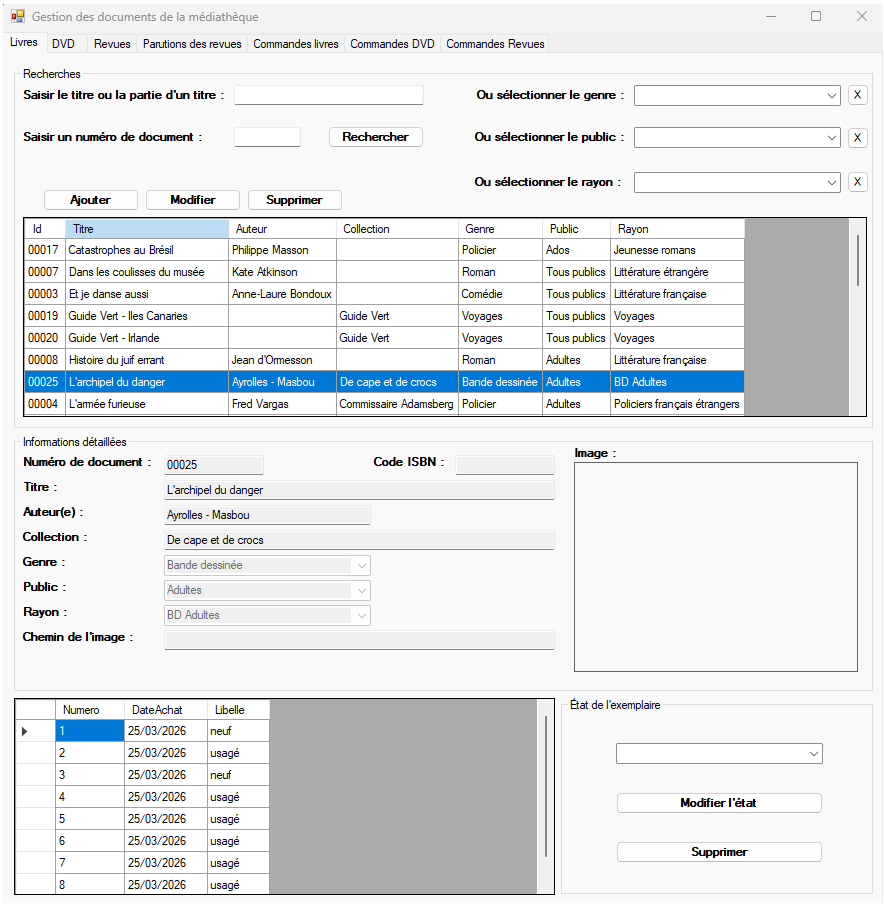
Temps de réalisation estimé : 5h
Temps de réalisation réel : 4h

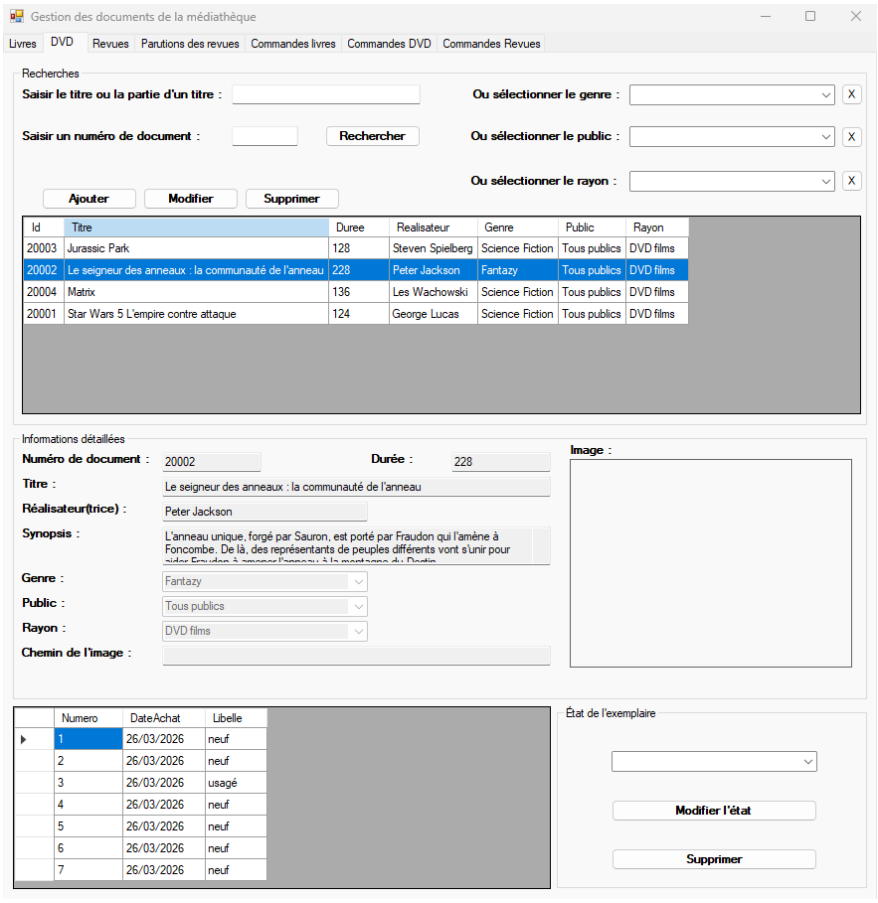
Diagramme de cas d’utilisation

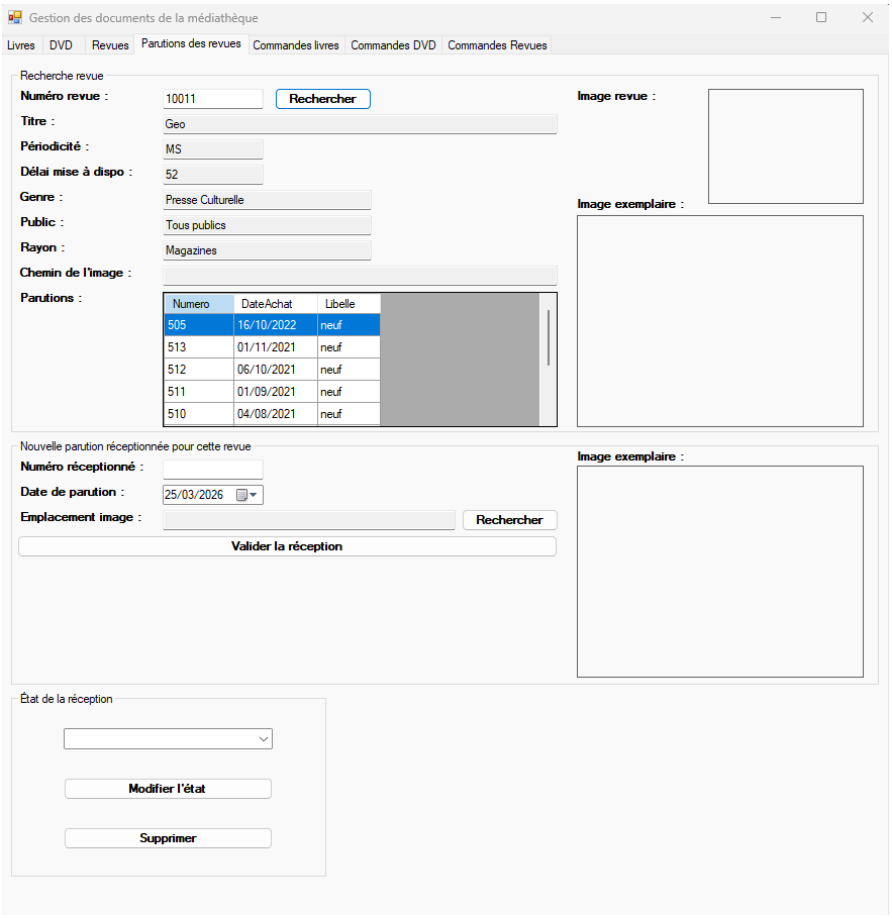


Aperçu

Voici à quoi ressemblent les onglets Livres, DVD et Parutions des revues :







Dans les onglets Livres et DVD j'ai ajouté un DataGridView qui affiche les exemplaires du document sélectionné. En bas à droite j'ai ajouté un GroupBox qui contient un ComboBox et un bouton qui permettent de modifier l'état de l'exemplaire sélectionné, et un bouton qui permet de supprimer l'exemplaire sélectionné.
Dans l'onglet Parutions des revues, j'ai juste ajouté le même GroupBox car les exemplaires sont déjà affichés.

Explication de code

Liste des classes impactées dans l'API :

MyAccessBDD.php :

```
private function selectExemplairesRevue(?array $champs) : ?array
```

J'ai modifié cette méthode pour retourner aussi le libellé de l'état.

```
private function updateEtatExemplaire(?string $id, ?array
$champs) : ?int
```

Cette méthode modifie l'état d'un exemplaire.

```
private function deleteExemplaire(?array $champs) : ?int
```

Cette méthode supprime un exemplaire.

Liste des classes impactées dans l'application C# :

J'ai modifié la classe Exemplaire.cs pour ajouter la propriété Libelle pour afficher le libellé de l'état dans les DataGrid :

```
using System;

namespace MediaTekDocuments.model
{
    /// <summary>
    /// Classe métier Exemplaire (exemplaire d'une revue)
    /// </summary>
    48 références | Nathan Boudier, Il y a 1 jour | 1 auteur, 1 modification
    public class Exemplaire
    {
        5 références | Nathan Boudier, Il y a 1 jour | 1 auteur, 1 modification
        public int Numero { get; set; }
        2 références | Nathan Boudier, Il y a 1 jour | 1 auteur, 1 modification
        public string Photo { get; set; }
        5 références | Nathan Boudier, Il y a 1 jour | 1 auteur, 1 modification
        public DateTime DateAchat { get; set; }
        4 références | Nathan Boudier, Il y a 1 jour | 1 auteur, 1 modification
        public string IdEtat { get; set; }
        4 références | 0 modification | 0 auteur, 0 modification
        public string Libelle { get; set; }
        3 références | Nathan Boudier, Il y a 1 jour | 1 auteur, 1 modification
        public string Id { get; set; }

        1 référence | 0 modification | 0 auteur, 0 modification
        public Exemplaire(int numero, DateTime dateAchat, string photo, string idEtat, string idDocument, string libelle = "")
        {
            this.Numero = numero;
            this.DateAchat = dateAchat;
            this.Photo = photo;
            this.IdEtat = idEtat;
            this.Id = idDocument;
            this.Libelle = libelle;
        }

    }
}
```

Access.cs :

J'ai créé les méthodes suivantes qui envoient les requêtes GET, POST, PUT ou DELETE à l'API et retournent les résultats correspondants :

```
public List<Etat> GetAllEtats()
public bool ModifierEtatExemplaire(Exemplaire exemplaire)
public bool SupprimerExemplaire(Exemplaire exemplaire)
```

FrmMediatekController.cs :

J'ai créé les méthodes suivantes pour faire le lien entre la vue et la classe Access :

```
public List<Etat> GetAllEtats()
public bool ModifierEtatExemplaire(Exemplaire exemplaire)
public bool SupprimerExemplaire(Exemplaire exemplaire)
```

FrmMediatek.cs

Dans la région : #region Onglet Livres :

```
private void RemplirLivresExemplaires(string idLivre)
```

Cette méthode récupère et affiche les exemplaires du livre sélectionné.

```
private void DgvLivresListe_SelectionChanged(object sender,
EventArgs e)
```

J'ai modifié cette méthode pour appeler RemplirLivresExemplaires à chaque sélection d'un livre.

J'ai ensuite ajouté ces méthodes pour le tri des colonnes, la modification de l'état d'un exemplaire, et la suppression d'un exemplaire :

```
private void dgvLivresExemplaires_ColumnHeaderMouseClick(object
sender, DataGridViewCellMouseEventArgs e)
private void btnLivresModifierEtat_Click(object sender,
EventArgs e)
private void btnLivresSupprimerExemplaire_Click(object sender,
EventArgs e)
```

Dans la région #region onglet Dvd :

```
private void RemplirDvdExemplaires(string idDvd)
```

Cette méthode récupère et affiche les exemplaires du livre sélectionné.

```
private void dgvDvdListe_SelectionChanged(object sender,
EventArgs e)
```

J'ai modifié cette méthode pour appeler RemplirLivresExemplaires à chaque sélection d'un dvd.

J'ai ensuite ajouté ces méthodes pour le tri des colonnes, la modification de l'état d'un exemplaire, et la suppression d'un exemplaire :

```
private void dgvDvdExemplaires_ColumnHeaderMouseClick(object
sender, DataGridViewCellMouseEventArgs e)
private void btnDvdModiferEtat_Click(object sender, EventArgs e)
```

```
private void btnDvdSupprimerExemplaire_Click(object sender,
EventArgs e)
```

Dans la région #region onglet Parutions :

```
private void RemplirReceptionExemplairesListe(List<Exemplaire>
exemplaires)
```

J’ai modifié cette méthode pour cacher la colonne Photo et afficher la colonne Libelle à la place.

```
private void dgvExemplairesListe_ColumnHeaderMouseClick(object
sender, DataGridViewCellMouseEventArgs e)
```

J’ai modifié cette méthode pour remplacer le tri su Photo par un tri sur Libelle.

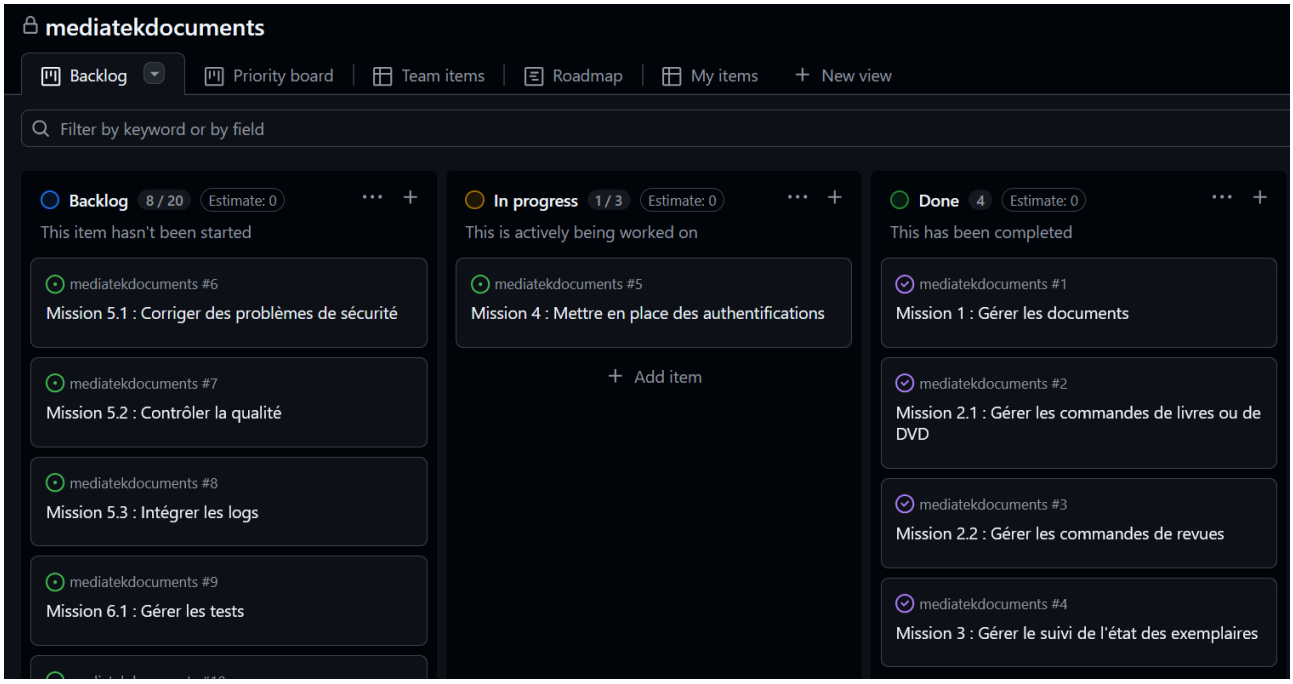
J’ai ensuite ajouté ces méthodes pour la modification de l’état d’un exemplaire et la suppression d’un exemplaire :

```
private void btnReceptionModifierEtat_Click(object sender,
EventArgs e)
private void btnReceptionSupprimerExemplaire_Click(object
sender, EventArgs e)
```

Mission 4 : Mettre en place des authentifications

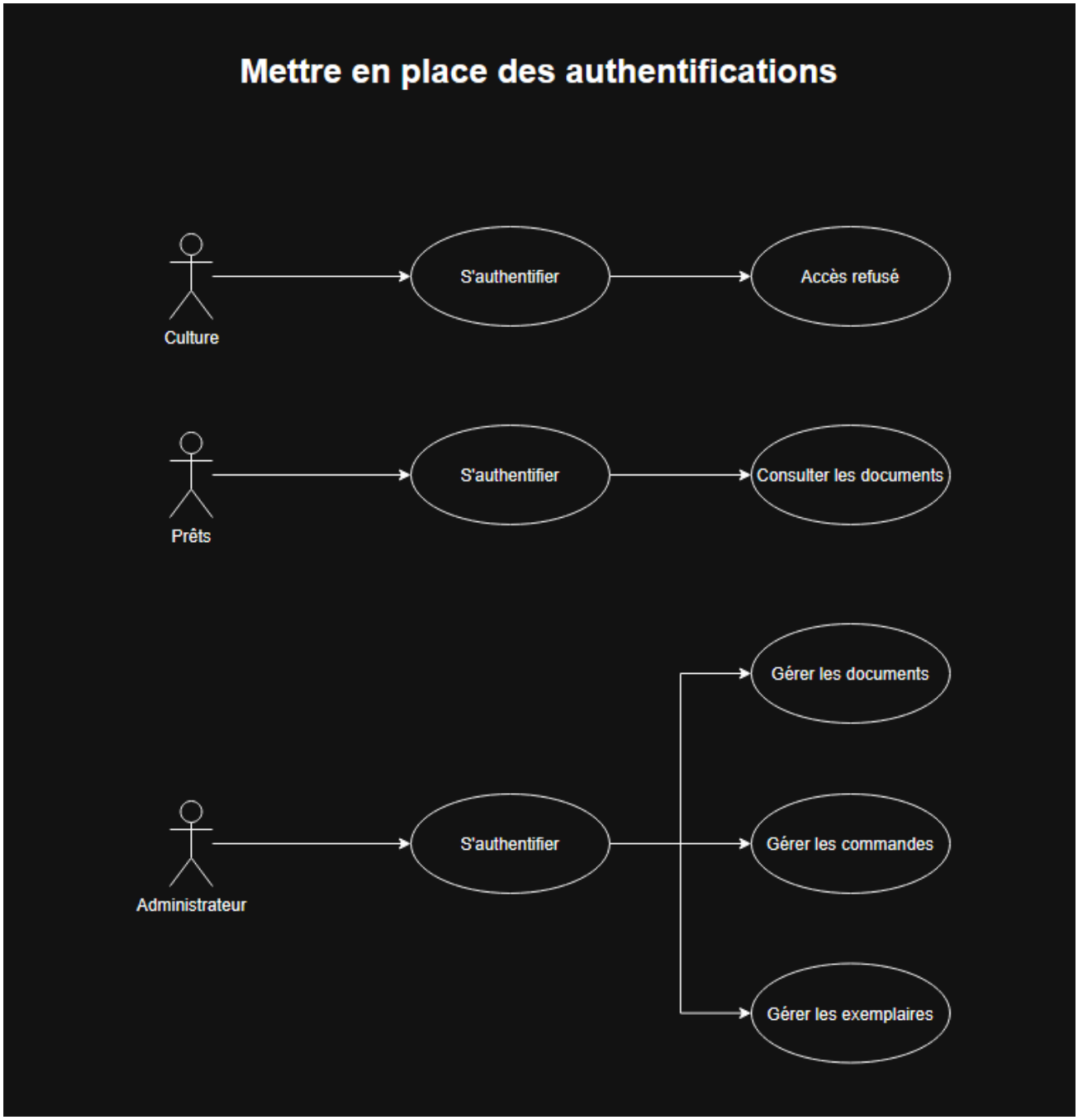
Demandes de la mission

- 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.
- Ajouter une première fenêtre d'authentification. Faire en sorte que l'application démarre sur cette fenêtre.
- Suivant le type de 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).



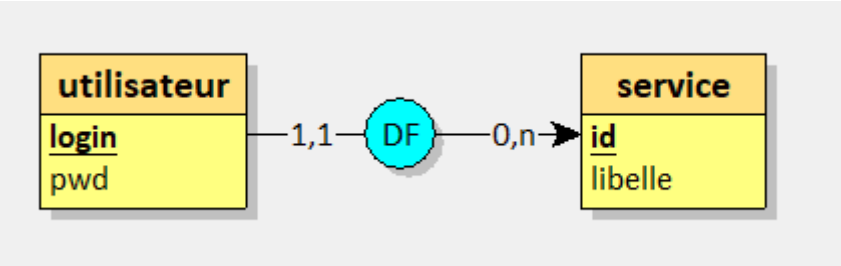
Temps de réalisation estimé : 4h
Temps de réalisation réel : 3h

Diagramme de cas d’utilisation



MCD

Voici les ajouts dans le MCD :



Nouvelle fenêtre

Voici à quoi ressemble la fenêtre d'authentification :

Au lancement de l'application, l'utilisateur doit se connecter. Si il est au service culture, un message lui indique qu'il n'a pas accès à l'application. Si il est au service des prêts, il a accès à l'application mais seulement en mode consultation, certains menus lui sont donc cachés. Si il est administrateur, il a accès à toutes l'application ainsi qu'à la fenêtre d'alerte pour les abonnements qui arrivent à expiration.

Base de données

J'ai créé les tables service et utilisateur dans la base de données, et je les ai remplis avec des données correspondants aux trois services mentionnés juste avant.

Explication de code

Liste des classes impactées dans l'API :

MyAccessBDD.php :

```
private function selectUtilisateur(?array $champs) : ?array
```

J'ai ajouté cette méthode qui récupère un utilisateur par son login et son mot de passe. Elle retourne les informations de l'utilisateur et son service si les identifiants sont corrects.

Liste des classes impactées dans l'application C# :

J'ai créé la classe Service.cs représentant un service de la médiathèque :

```
namespace MediaTekDocuments.model
{
    /// <summary>
    /// Classe métier Service
    /// </summary>
    1 référence | 0 modification | 0 auteur, 0 modification
    public class Service
    {
        1 référence | 0 modification | 0 auteur, 0 modification
        public string Id { get; }
        1 référence | 0 modification | 0 auteur, 0 modification
        public string Libelle { get; }

        0 références | 0 modification | 0 auteur, 0 modification
        public Service(string id, string libelle)
        {
            Id = id;
            Libelle = libelle;
        }
    }
}
```

J’ai également créé la classe Utilisateur.cs représentant un utilisateur avec son login, mot de passe, id du service et libellé du service :

```
namespace MediaTekDocuments.model
{
    /// <summary>
    /// Classe métier Utilisateur
    /// </summary>
    10 références | 0 modification | 0 auteur, 0 modification
    public class Utilisateur
    {
        1 référence | 0 modification | 0 auteur, 0 modification
        public string Login { get; }
        1 référence | 0 modification | 0 auteur, 0 modification
        public string Pwd { get; }
        4 références | 0 modification | 0 auteur, 0 modification
        public string IdService { get; }
        1 référence | 0 modification | 0 auteur, 0 modification
        public string Libelle { get; }

        0 références | 0 modification | 0 auteur, 0 modification
        public Utilisateur(string login, string pwd, string idService, string libelle)
        {
            Login = login;
            Pwd = pwd;
            IdService = idService;
            Libelle = libelle;
        }
    }
}
```

Access.cs :

J’ai ajouté cette méthode qui envoie une requête GET à l’API avec le login et le mot de passe. Elle retourne l’objet Utilisateur si les identifiants sont corrects, et null si ils sont incorrects :

```
public Utilisateur GetUtilisateur(string login, string pwd)
```

FrmMediatekController.cs :

J’ai créé la méthode suivante pour faire le lien entre la vue et la classe Access :

```
public Utilisateur GetUtilisateur(string login, string pwd)
```

FrmAuthentification.cs :

Nouveau formulaire d'authentification qui s'ouvre au démarrage. Vérifie les identifiants saisis et stocke l'utilisateur connecté dans utilisateurConnecte si la connexion réussi :

```
public FrmAuthentification()
private void btnAuthentification_Click(object sender, EventArgs
e)
```

Program.cs :

J'ai modifié Program.cs pour que l'application démarre sur FrmAuthentification. Si la connexion réussit et que le service est « Culture », un message qui dit que l'accès est refusé s'affiche et l'application se ferme. Sinon FrmMediatek s'ouvre en recevant l'utilisateur connecté en paramètre.

```
/// <summary>
/// Point d'entrée principal de l'application.
/// </summary>
[STAThread]
0 références | Nathan Boudier, il y a 2 jours | 1 auteur, 1 modification
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);

    // ouverture de la fenêtre d'authentification
    FrmAuthentification frmAuthentification = new FrmAuthentification();
    if (frmAuthentification.ShowDialog() == DialogResult.OK)
    {
        Utilisateur utilisateur = frmAuthentification.utilisateurConnecte;

        // service Culture : pas d'accès
        if (utilisateur.IdService == "00003")
        {
            MessageBox.Show("Vous n'avez pas les droits suffisants pour accéder à cette application.", "Accès refusé");
            return;
        }

        // ouverture de l'application principale
        FrmMediatek frmMediatek = new FrmMediatek(utilisateur);
        Application.Run(frmMediatek);
    }
}
```

FrmMediatek.cs :

J'ai modifié le constructeur qui reçoit maintenant l'utilisateur connecté en paramètre :

```
internal FrmMediatek(Utilisateur utilisateur)
```

La méthode suivante est appelé dans le constructeur, elle adapte l'interface selon le service de l'utilisateur. Pour le service Prêts, elle supprime les onglets commandes et cache les boutons d'ajout, de modification et de suppression :

```
private void GererAcces()
```

Enfin j'ai ajouté la méthode suivante pour que l'alerte des abonnements arrivant à expiration ne s'affiche que pour le service Administratif :

```
private void AfficheAlerteAbonnements()
```

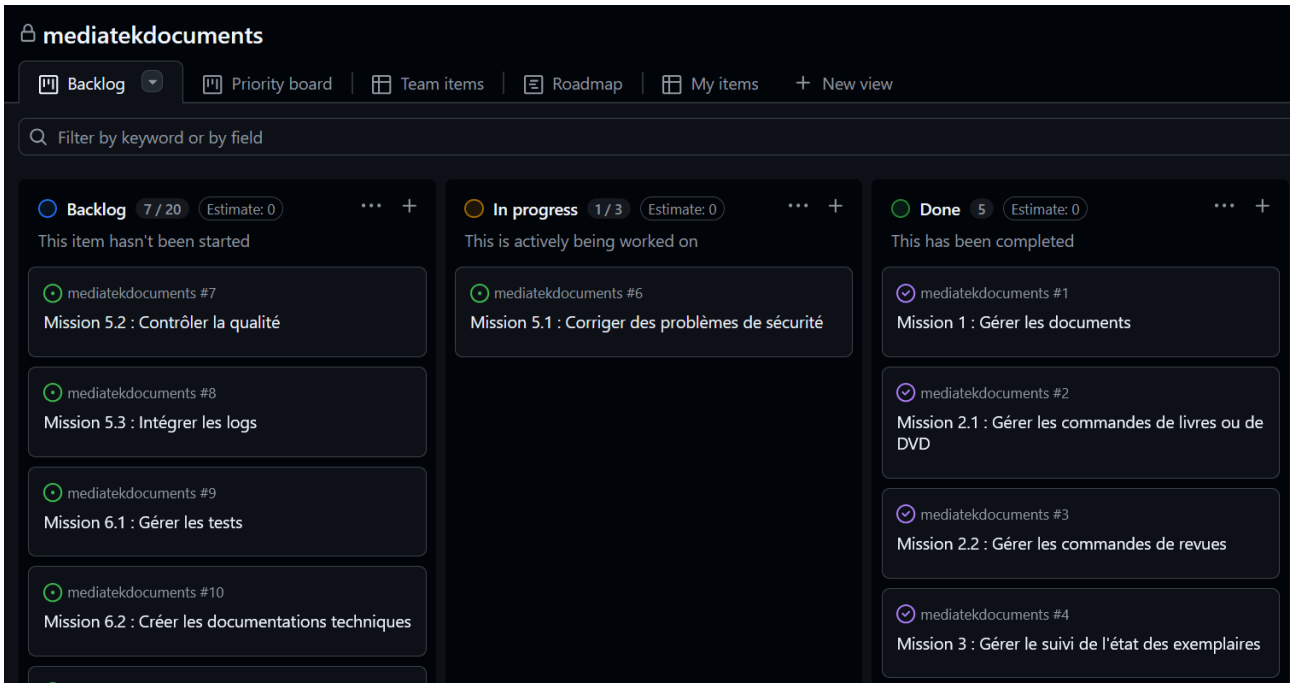
Mission 5.1 : Corriger des problèmes de sécurités

Demandes de la mission

Pour chaque problème, créer une "issue" dans le dépôt GitHub correspondant, affecter l'issue à un des développeurs (si vous êtes 2, vous ou votre collègue, sinon vous-même), créer une branche pour proposer une correction de code en expliquant la correction, faire un pull request, accepter le pull request si le résultat correspond bien aux attentes.

Problème n°1 :
Actuellement, l'accès à l'API se fait en authentification basique, avec le couple "login:pwd" en dur dans le code de l'application (dans le constructeur de la classe Access). Le but est de sécuriser cette information.

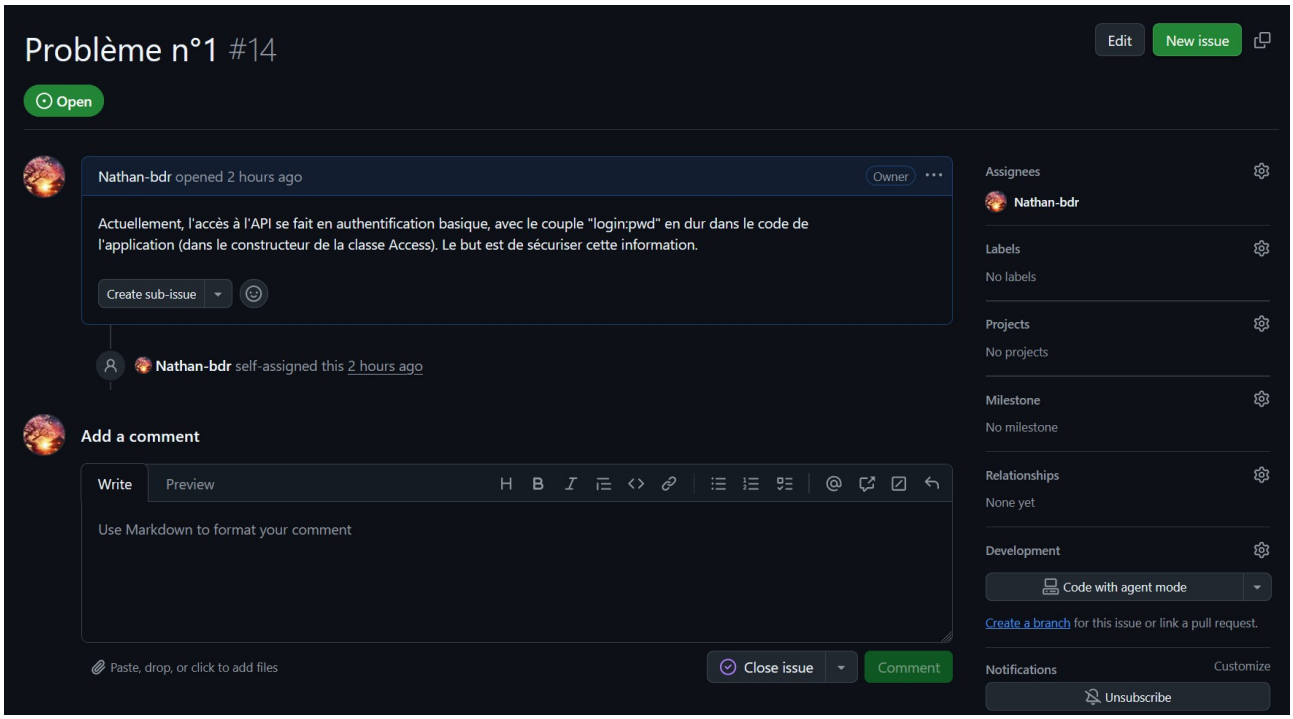
Problème n°2 :
Si, pour accéder à l'API directement dans un navigateur, on donne juste l'adresse de l'API sans mettre de paramètres :
http://localhost//rest_mediatekdocuments/
on obtient la liste des fichiers contenus dans le dossier de l'API. Le but est d'avoir un retour d'erreur.



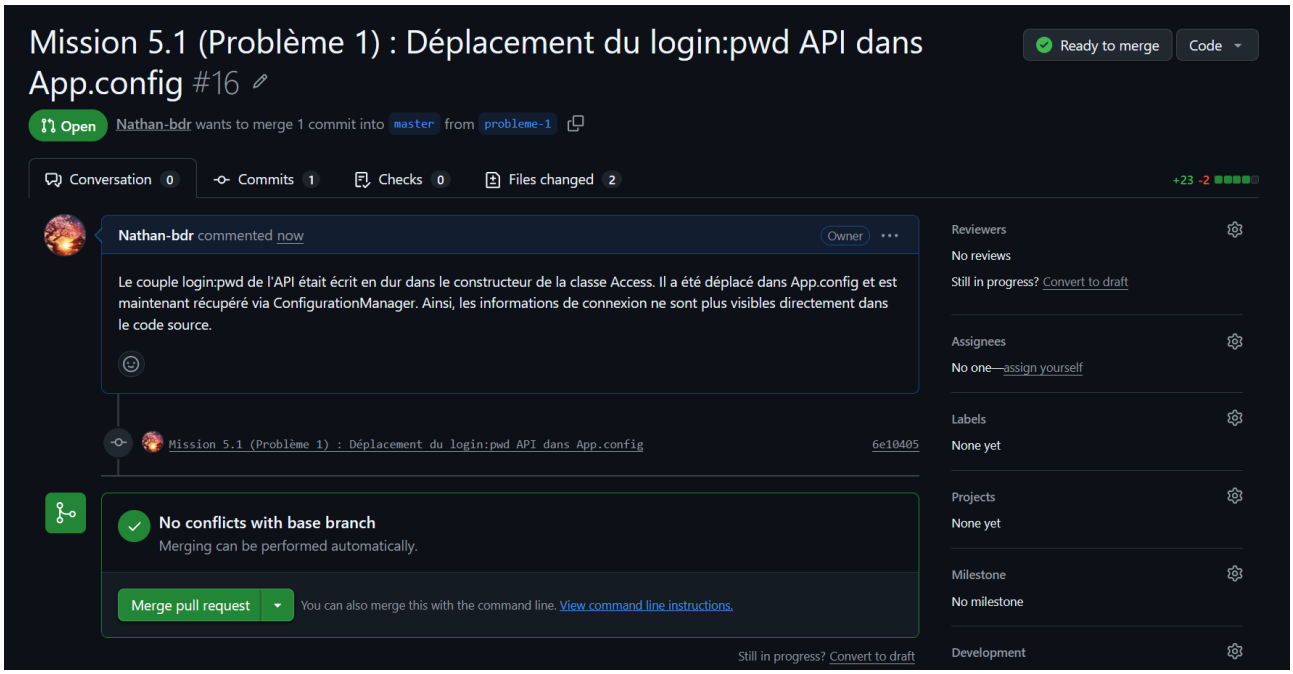
Temps de réalisation estimé : 2h
Temps de réalisation réel : 2h30

Problème n°1

J’ai d’abord créé une issue pour le problème 1 :



Voici ensuite le pull request :



Dans le constructeur de la classe Access.cs, le couple login:pwd permettent d'accéder à l'API était écrit en dur directement dans le code :

```
authenticationString = "admin:adminpwd";
```

Cela pose un problème de sécurité car toute personne ayant accès au code source peut voir ces informations.

J'ai donc déplacé ces informations dans le fichier App.config dans une section connectionStrings :

```
<connectionStrings>
    <add name="MediaTekDocuments.Properties.Settings.apiAuthentification"
         connectionString="admin:adminpwd" />
</connectionStrings>
```

Dans Access.cs, j'ai ajouté une méthode pour récupérer cette valeur depuis App.config via ConfigurationManager :

```
/// <summary>
/// Récupère la chaîne d'authentification depuis App.config
/// </summary>
1 référence | Nathan Boudier, Il y a 26 minutes | 1 auteur, 1 modification
private static string GetAuthenticationString(string name)
{
    string returnValue = null;
    ConnectionStringSettings settings = ConfigurationManager.ConnectionStrings[name];
    if (settings != null)
        returnValue = settings.ConnectionString;
    return returnValue;
}
```

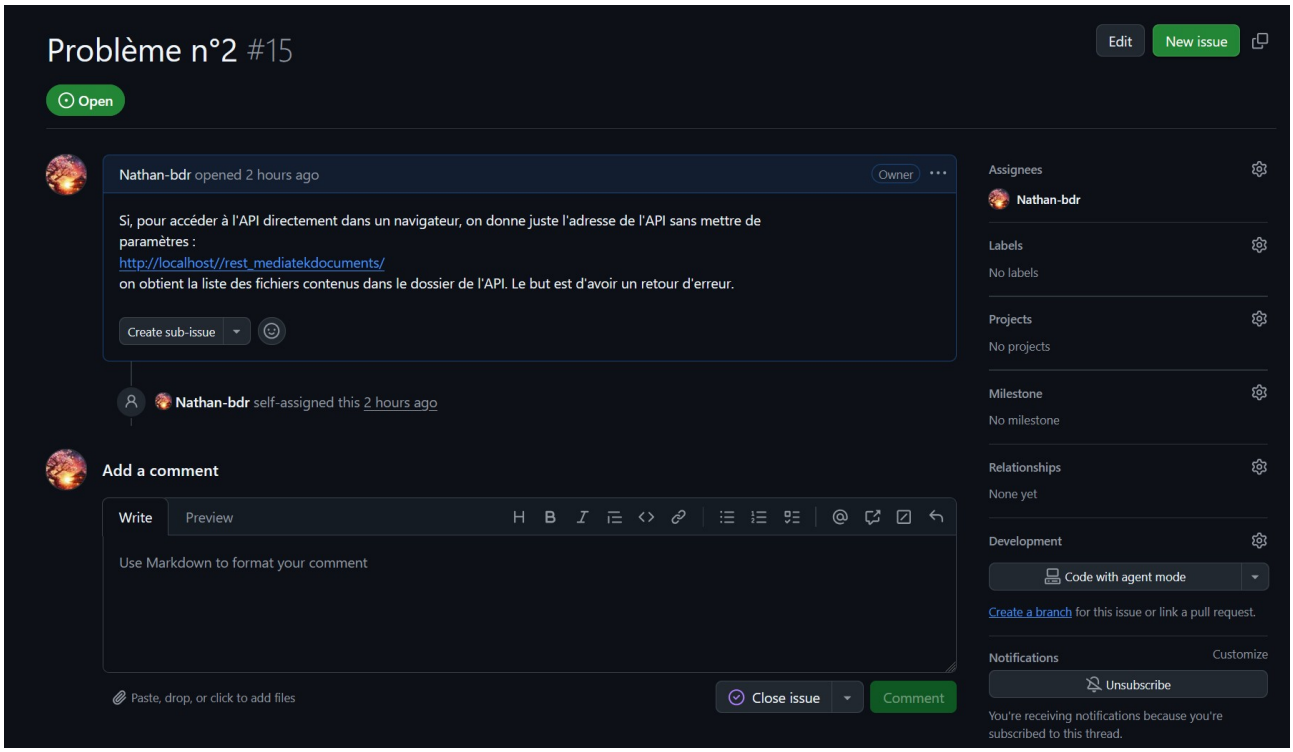
J'ai aussi modifié le constructeur pour qu'il utilise cette méthode :

```
authenticationString = GetAuthenticationString(connectionName);
```

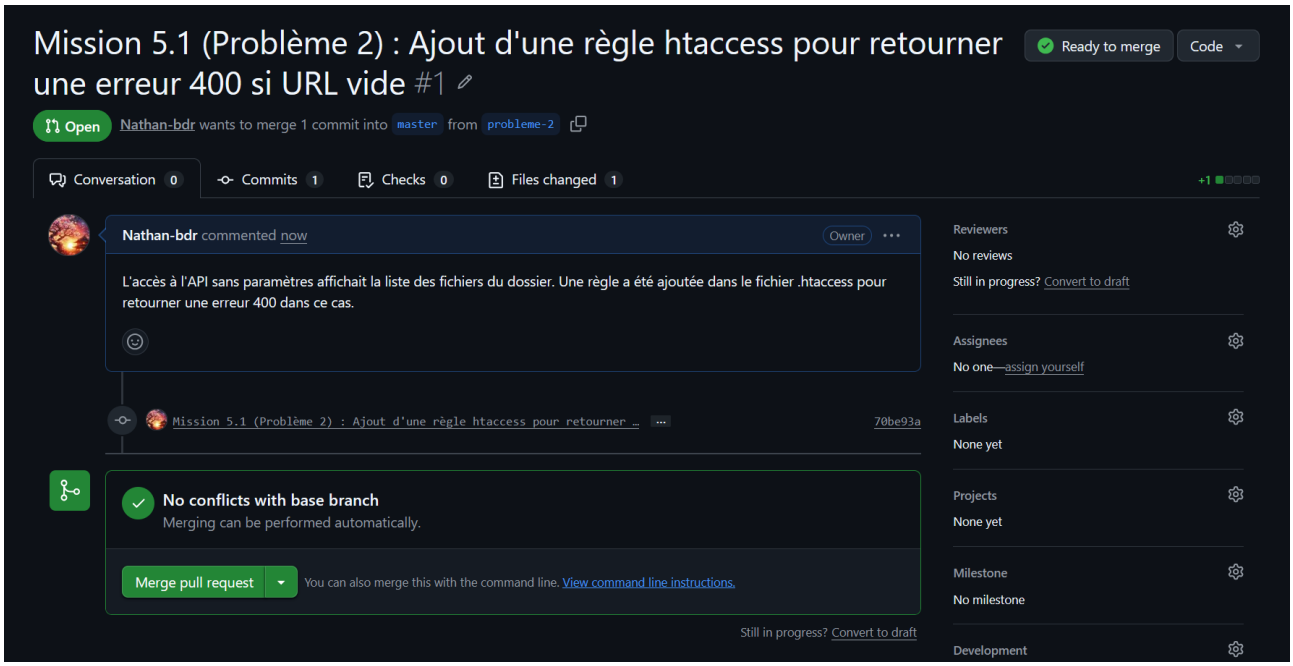
Les informations de connexion ne sont donc désormais plus visibles dans le code source.

Problème n°2

J’ai d’abord créé une issue pour le problème 2 :



Voici ensuite le pull request :



En accédant à l’adresse de l’API sans paramètre dans un navigateur, la liste des fichiers contenus dans le dossier de l’API s’affiche, ce qui représente un problème de sécurité car cela expose la structure du projet.

J’ai donc ajouté une règle dans le fichier .htaccess :

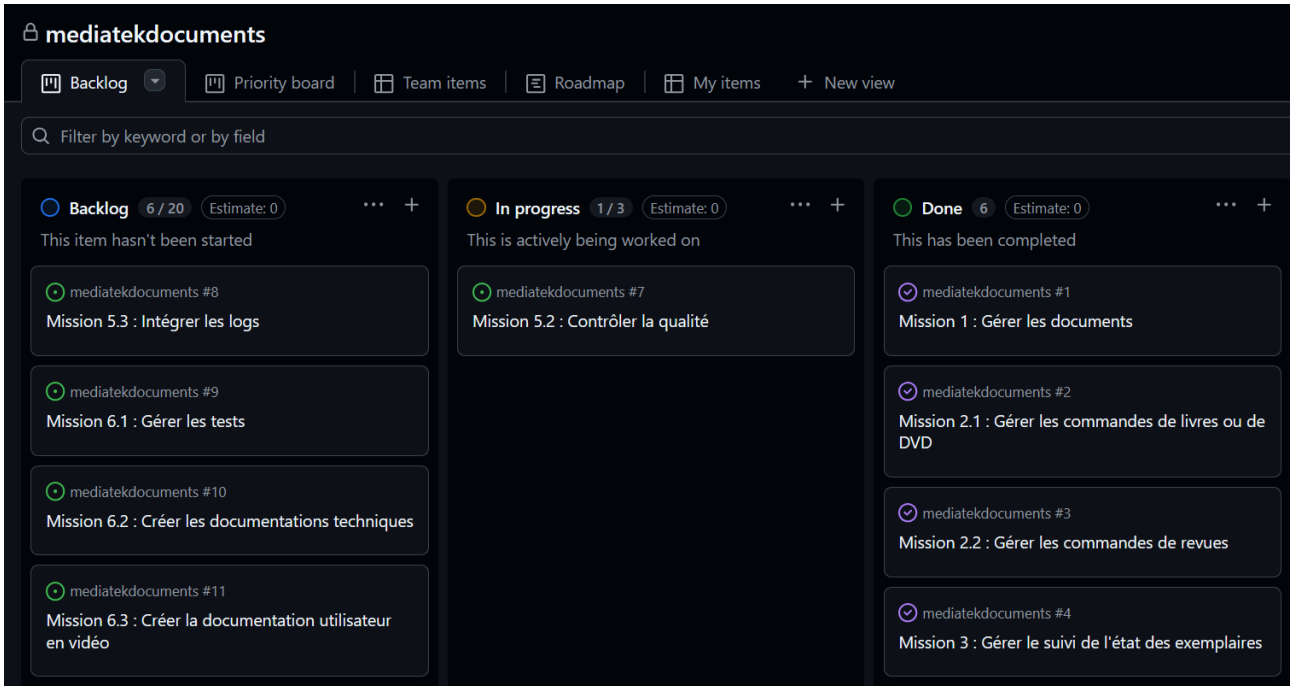
```
RewriteRule ^$ - [R=400,L]
```

Désormais, accéder à l’API sans paramètres retourne une erreur 400 « Bad Request » au lieu d’afficher la liste des fichiers.

Mission 5.2 : Contrôler la qualité

Demandes de la mission

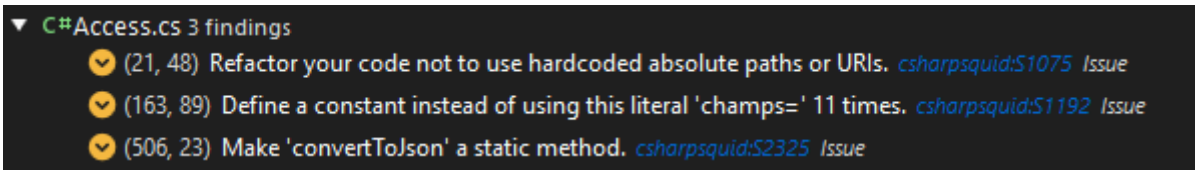
- Contrôler que SonarQube for IDE est configuré dans Visual Studio.
- Corriger les problèmes relevés par SonarQube for IDE dans le code ajouté (excepté les problèmes qui ne doivent pas être corrigés, comme les noms des méthodes événementielles qui commencent par une minuscule).



Temps de réalisation estimé : 1h
Temps de réalisation réel : 1h

Erreurs et avertissements

Voici les erreurs et avertissements de SonarQube :



Le premier avertissement concerne la ligne suivante :

```
/// <summary>
/// adresse de l'API
/// </summary>
private static readonly string uriApi = "http://localhost/rest_mediatekdocuments/";
```

C'est le même problème que pour la clé login:pwd, donc on fait la même chose. On ajoute l'URL de l'API dans le connectionStrings de App.config :

```
<connectionStrings>
    <add name="MediaTekDocuments.Properties.Settings.apiAuthentification"
         connectionString="admin:adminpwd" />
    <add name="MediaTekDocuments.Properties.Settings.uriApi"
         connectionString="http://localhost/rest_mediatekdocuments/" />
</connectionStrings>
```

Et voici la ligne modifiée, on réutilise la méthode GetAuthentificationString que l'on avait créé :

```
/// <summary>
/// adresse de l'API
/// </summary>
private static readonly string uriApi = GetAuthenticationString("MediaTekDocuments.Properties.Settings.uriApi");
```

Le deuxième avertissement concerne les multiples occurrences de la chaîne « champs= » :

```
List<Exemplaire> liste = TraitementRecup<Exemplaire>(POST, "exemplaire", "champs=" + jsonExemplaire);
```

J'ai donc créé une constante pour éviter de répéter la chaîne :

```
/// <summary>
/// préfixe pour les paramètres envoyés dans le body
/// </summary>
private const string CHAMPS = "champs=";
```

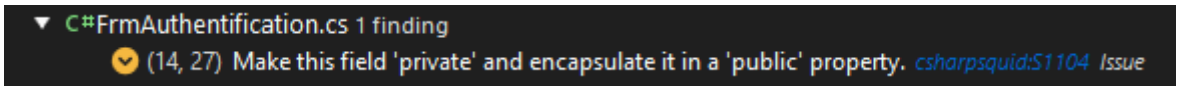
```
List<Exemplaire> liste = TraitementRecup<Exemplaire>(POST, "exemplaire", CHAMPS + jsonExemplaire);
```

Le troisième indique que la méthode suivante peut être statique car elle n'utilise pas de membres d'instances de la classe.

```
private String convertToJson(Object nom, Object valeur)
{
    Dictionary<Object, Object> dictionary = new Dictionary<Object, Object>();
    dictionary.Add(nom, valeur);
    return JsonConvert.SerializeObject(dictionary);
}
```

Il suffit donc d'ajouter static à la méthode :

```
private static String convertToJson(Object nom, Object valeur)
{
    Dictionary<Object, Object> dictionary = new Dictionary<Object, Object>();
    dictionary.Add(nom, valeur);
    return JsonConvert.SerializeObject(dictionary);
}
```



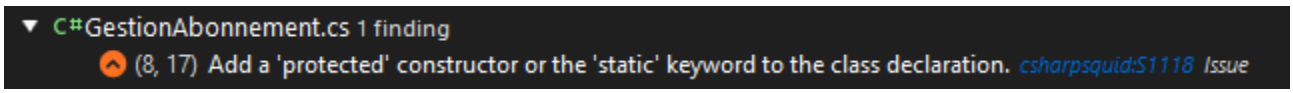
Cet avertissement indique que le champ public suivant devrait être privé et accessible via une propriété :

```
/// <summary>
/// Utilisateur connecté, accessible depuis l'extérieur
/// </summary>
public Utilisateur utilisateurConnecte = null;
```

On fait donc la modification suivante :

```
/// <summary>
/// Utilisateur connecté, accessible depuis l'extérieur
/// </summary>
2 références | 0 modification | 0 auteur, 0 modification
public Utilisateur UtilisateurConnecte { get; private set; } = null;
```

Il faut juste bien mettre un U majuscule au début aux endroits qui appelaient utilisateurConnecte.



Cet avertissement dit que comme la classe ne contient que des méthodes statiques, elle devrait elle-même être déclarée static.

```
public class GestionAbonnement
{
    /// <summary>
    /// Vérifie si une date de parution est comprise entre deux dates
    /// </summary>
    4 références | Nathan Boudier, Il y a 11 heures | 1 auteur, 1 modification
    public static bool ParutionDansAbonnement(DateTime dateCommande, DateTime dateFinAbonnement, DateTime dateParution)
    {
        return dateParution >= dateCommande && dateParution <= dateFinAbonnement;
    }
}
```

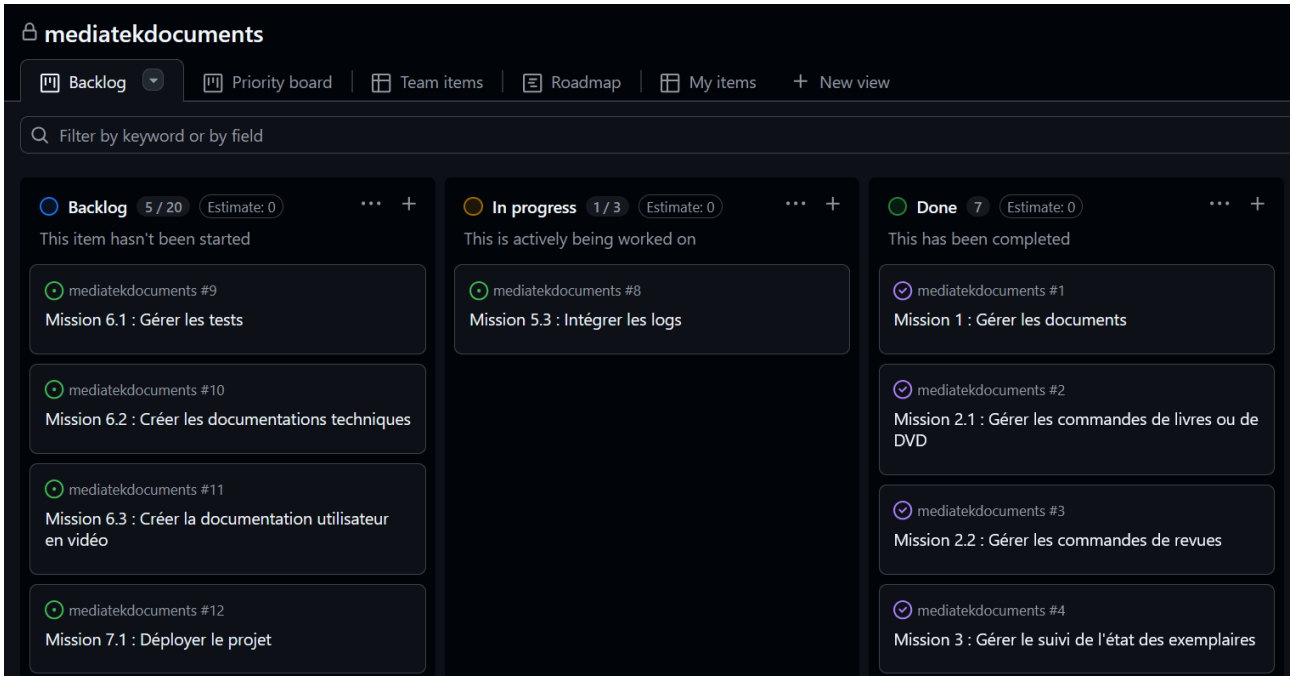
On ajoute donc static à la classe :

```
public static class GestionAbonnement
{
    /// <summary>
    /// Vérifie si une date de parution est comprise entre deux dates
    /// </summary>
    4 références | Nathan Boudier, Il y a 11 heures | 1 auteur, 1 modification
    public static bool ParutionDansAbonnement(DateTime dateCommande, DateTime dateFinAbonnement, DateTime dateParution)
    {
        return dateParution >= dateCommande && dateParution <= dateFinAbonnement;
    }
}
```

Mission 5.3 : Intégrer les logs

Demandes de la mission

Dans la classe Access, ajouter le code de configuration des logs et des logs au niveau de chaque affichage console (à enregistrer dans un fichier de logs).



Temps de réalisation estimé : 1h
Temps de réalisation réel : 1h

Démarche

J’ai installé le package Serilog dans le projet pour pouvoir gérer les logs.

Voici le code qui permet de configurer les logs dans le constructeur de la classe Access.cs :

```
Log.Logger = new LoggerConfiguration()
    .MinimumLevel.Verbose()
    .WriteTo.Console()
    .WriteTo.File("logs\\log.txt")
    .CreateLogger();
```

Cette configuration permet d’écrire les logs à la fois dans la console et dans un fichier log.txt dans le dossier logs.

Exemple d’ajout d’un log en cas d’erreur lors d’un appel à l’API :

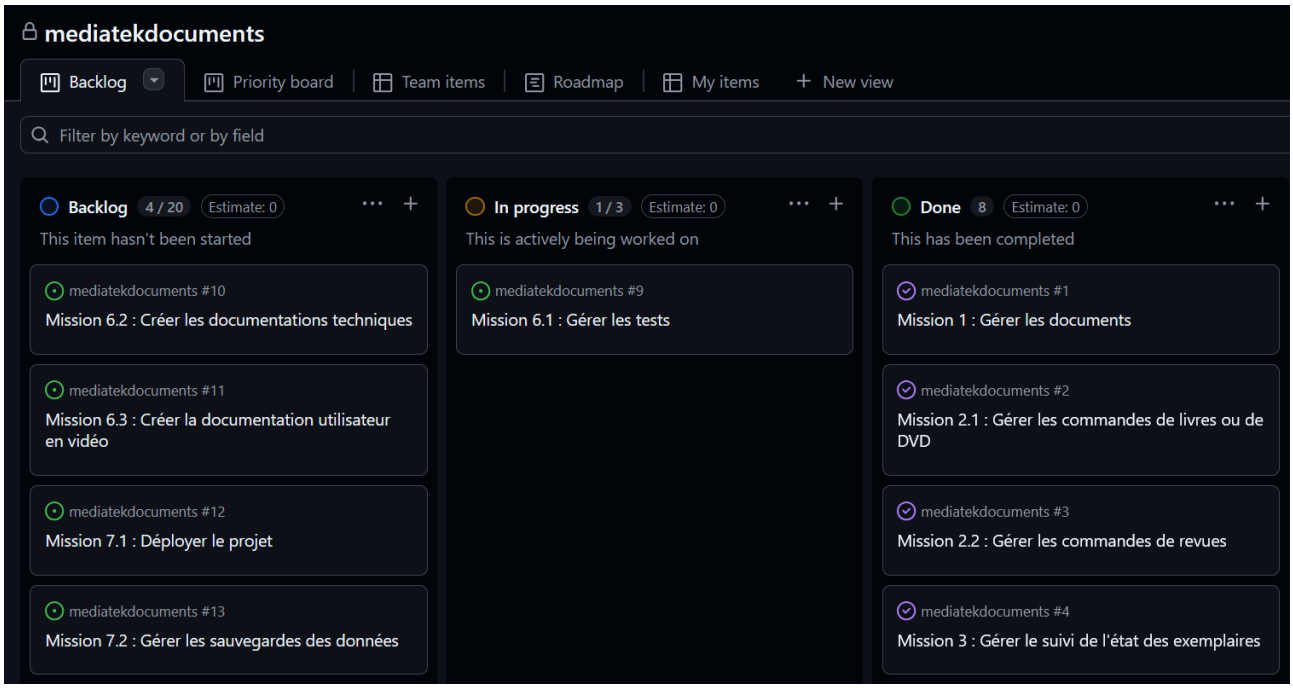
```
Log.Fatal("Access.TraitementRecup : erreur={0}", e.Message);
```

Mission 6.1 : Gérer les tests

Demandes de la mission

Écrire les tests unitaires sur les classes du package Model (en plus du test unitaire écrit précédemment).

Construire une collection de tests dans Postman pour contrôler les fonctionnalités de l'API d'accès à la BDD.



Temps de réalisation estimé : 5h
Temps de réalisation réel : 4h30

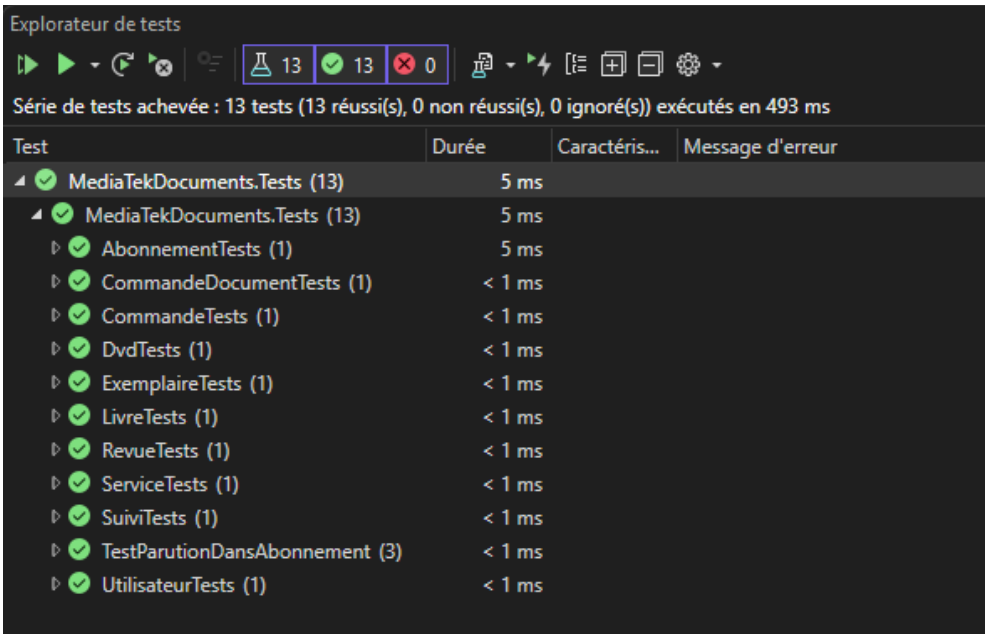
Tests unitaires

J’ai créé des classes de test pour toutes les classes du package Model. Chaque test vérifie que le constructeur initialise bien toutes les propriétés de la classe.
Voici par exemple LivreTests.cs :

```
public class LivreTests
{
    private const string id = "00001";
    private const string titre = "titre test";
    private const string image = "image test";
    private const string isbn = "isbn test";
    private const string auteur = "auteur test";
    private const string collection = "collection test";
    private const string idGenre = "10001";
    private const string genre = "genre test";
    private const string idPublic = "00001";
    private const string lePublic = "public test";
    private const string idRayon = "LV001";
    private const string rayon = "rayon test";
    private static readonly Livre livre = new Livre(id, titre, image, isbn, auteur, collection,
        idGenre, genre, idPublic, lePublic, idRayon, rayon);

    [TestMethod]
    | 0 références | 0 modification | 0 auteur, 0 modification
    public void LivreTest()
    {
        Assert.AreEqual(id, livre.Id, "devrait réussir : id valorisé");
        Assert.AreEqual(titre, livre.Titre, "devrait réussir : titre valorisé");
        Assert.AreEqual(image, livre.Image, "devrait réussir : image valorisé");
        Assert.AreEqual(isbn, livre.Isbn, "devrait réussir : isbn valorisé");
        Assert.AreEqual(auteur, livre.Auteur, "devrait réussir : auteur valorisé");
        Assert.AreEqual(collection, livre.Collection, "devrait réussir : collection valorisé");
        Assert.AreEqual(idGenre, livre.IdGenre, "devrait réussir : idGenre valorisé");
        Assert.AreEqual(genre, livre.Genre, "devrait réussir : genre valorisé");
        Assert.AreEqual(idPublic, livre.IdPublic, "devrait réussir : idPublic valorisé");
        Assert.AreEqual(lePublic, livre.Public, "devrait réussir : public valorisé");
        Assert.AreEqual(idRayon, livre.IdRayon, "devrait réussir : idRayon valorisé");
        Assert.AreEqual(rayon, livre.Rayon, "devrait réussir : rayon valorisé");
    }
}
```

Tous les tests unitaires réussissent :



Tests fonctionnels Postman

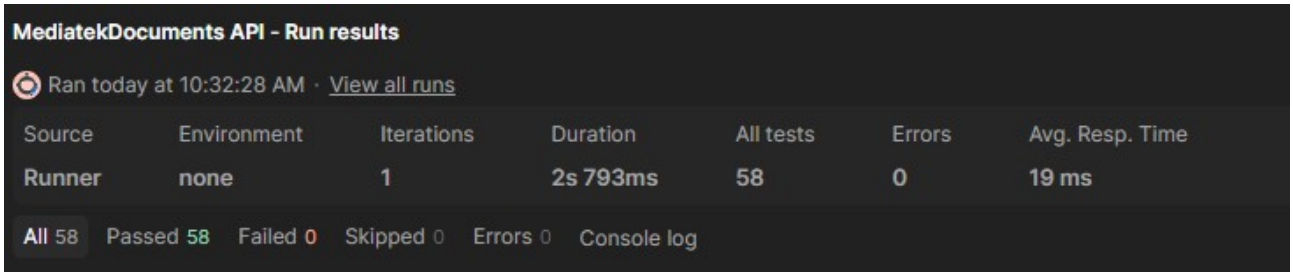
J’ai créé dans Postman une collection de tests MediatekDocuments API. Dans cette collection j’ai créé tous les tests GET, POST, PUT et DELETE nécéssaires :

Chaque requête contient au minimum deux tests : une vérification du code HTTP 200 et une vérification du code de retour de l'API.
Voici par exemple POST livre :

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Livre ajouté avec succès", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData["code"]).to.eql(200);
    pm.expect(jsonData["result"]).to.eql(1);
});
```

Tous les tests passent :



Plan de tests

Tests unitaires sur les classes du package model

But du test	Action de contrôle	Résultat attendu	Bilan
Contrôler le constructeur de la classe Livre	Test unitaire lancé après avoir créé un objet Livre avec des valeurs de test	Toutes les propriétés sont correctement valorisées	OK
Contrôler le constructeur de la classe Dvd	Test unitaire lancé après avoir créé un objet Dvd avec des valeurs de test	Toutes les propriétés sont correctement valorisées	OK
Contrôler le constructeur de la classe Revue	Test unitaire lancé après avoir créé un objet Revue avec des valeurs de test	Toutes les propriétés sont correctement valorisées	OK
Contrôler le constructeur de la classe Exemplaire	Test unitaire lancé après avoir créé un objet Exemplaire avec des valeurs de test	Toutes les propriétés sont correctement valorisées	OK
Contrôler le constructeur de la classe Commande	Test unitaire lancé après avoir créé un objet Commande avec des valeurs de test	Toutes les propriétés sont correctement valorisées	OK
Contrôler le constructeur de la classe CommandeDocument	Test unitaire lancé après avoir créé un objet CommandeDocument avec des valeurs de test	Toutes les propriétés sont correctement valorisées	OK
Contrôler le constructeur de la classe Abonnement	Test unitaire lancé après avoir créé un objet Abonnement avec des valeurs de test	Toutes les propriétés sont correctement valorisées	OK
Contrôler le constructeur de la classe Utilisateur	Test unitaire lancé après avoir créé un objet Utilisateur avec des valeurs de test	Toutes les propriétés sont correctement valorisées	OK
Contrôler le constructeur de la classe Service	Test unitaire lancé après avoir créé un objet Service avec des valeurs de test	Toutes les propriétés sont correctement valorisées	OK
Contrôler le constructeur de la classe Suivi	Test unitaire lancé après avoir créé un objet Suivi avec des valeurs de test	Toutes les propriétés sont correctement valorisées	OK
Contrôler que ParutionDansAbonnement retourne vrai si la date est dans la période	Test unitaire avec dateCommande=01/01/2024, dateFinAbonnement=31/12/2024, dateParution=15/06/2024	true	OK
Contrôler que ParutionDansAbonnement retourne faux si la date est avant la période	Test unitaire avec dateCommande=01/01/2024, dateFinAbonnement=31/12/2024, dateParution=31/12/2023	false	OK
Contrôler que ParutionDansAbonnement retourne faux si la date est après la période	Test unitaire avec dateCommande=01/01/2024, dateFinAbonnement=31/12/2024, dateParution=01/01/2025	false	OK
Contrôler que ParutionDansAbonnement retourne vrai si la date est égale à la date de début	Test unitaire avec dateCommande=01/01/2024, dateFinAbonnement=31/12/2024, dateParution=01/01/2024	true	OK
Contrôler que ParutionDansAbonnement retourne vrai si la date est égale à la date de fin	Test unitaire avec dateCommande=01/01/2024, dateFinAbonnement=31/12/2024, dateParution=31/12/2024	true	OK

Tests fonctionnels dans Postman (fonctionnalités de l'API)

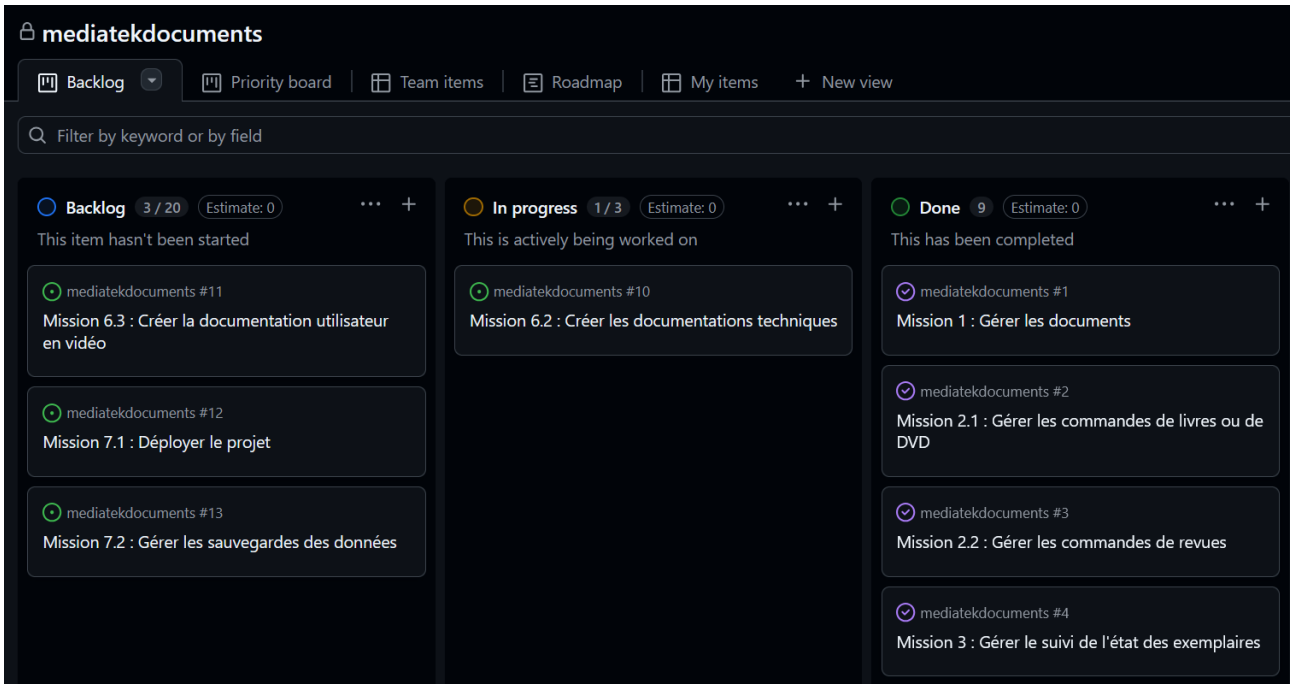
But du test	Action de contrôle	Résultat attendu	Bilan
Récupérer la liste des livres	GET http://localhost/rest_mediatekdocuments/livre	code 200, result est un tableau	OK
Récupérer la liste des dvd	GET http://localhost/rest_mediatekdocuments/dvd	code 200, result est un tableau	OK
Récupérer la liste des revues	GET http://localhost/rest_mediatekdocuments/revue	code 200, result est un tableau	OK
Récupérer la liste des genres	GET http://localhost/rest_mediatekdocuments/genre	code 200, result est un tableau	OK
Récupérer la liste des publics	GET http://localhost/rest_mediatekdocuments/public	code 200, result est un tableau	OK
Récupérer la liste des rayons	GET http://localhost/rest_mediatekdocuments/rayon	code 200, result est un tableau	OK
Récupérer la liste des états	GET http://localhost/rest_mediatekdocuments/etat	code 200, result est un tableau	OK
Récupérer la liste des suivis	GET http://localhost/rest_mediatekdocuments/suivi	code 200, result est un tableau	OK
Ajouter un livre	POST http://localhost/rest_mediatekdocuments/livre avec champs JSON	code 200, result = 1	OK
Ajouter un dvd	POST http://localhost/rest_mediatekdocuments/dvd avec champs JSON	code 200, result = 1	OK
Ajouter une revue	POST http://localhost/rest_mediatekdocuments/revue avec champs JSON	code 200, result = 1	OK
Ajouter un exemplaire	POST http://localhost/rest_mediatekdocuments/exemplaire avec champs JSON	code 200, result = 1	OK
Ajouter une commande document	POST http://localhost/rest_mediatekdocuments/commandedocument avec champs JSON	code 200, result = 1	OK
Ajouter un abonnement	POST http://localhost/rest_mediatekdocuments/abonnement avec champs JSON	code 200, result = 1	OK
Modifier un livre	PUT http://localhost/rest_mediatekdocuments/livre/99999 avec champs JSON	code 200, result = 1	OK
Modifier un dvd	PUT http://localhost/rest_mediatekdocuments/dvd/99998 avec champs JSON	code 200, result = 1	OK
Modifier une revue	PUT http://localhost/rest_mediatekdocuments/revue/99997 avec champs JSON	code 200, result = 1	OK
Modifier l'état d'un exemplaire	PUT http://localhost/rest_mediatekdocuments/exemplaire/00001 avec champs JSON	code 200, result = 1	OK
Modifier le suivi d'une commande	PUT http://localhost/rest_mediatekdocuments/commandedocument/99990 avec champs JSON	code 200, result = 1	OK
Supprimer un exemplaire	DELETE http://localhost/rest_mediatekdocuments/exemplaire/{"id":"00001","Numero":999}	code 200, result = 1	OK

Supprimer une commande document	DELETE http://localhost/rest_mediatekdocuments/commandedocument/{"id":"99990"}	code 200, result = 1	OK
Supprimer un abonnement	DELETE http://localhost/rest_mediatekdocuments/abonnement/{"id":"99991"}	code 200, result = 1	OK
Supprimer un livre	DELETE http://localhost/rest_mediatekdocuments/livre/{"id":"99999"}	code 200, result = 1	OK
Supprimer un dvd	DELETE http://localhost/rest_mediatekdocuments/dvd/{"id":"99998"}	code 200, result = 1	OK
Supprimer une revue	DELETE http://localhost/rest_mediatekdocuments/revue/{"id":"99997"}	code 200, result = 1	OK

Mission 6.2 : Créer les documentations techniques

Demandes de la mission

- Contrôler, dans chaque application, que les commentaires normalisés sont bien tous ajoutés et corrects.
- Générer la documentation technique de l'application C#, en suivant les explications données dans l'article "Documentation technique sous Visual Studio".
- Générer la documentation technique de l'API REST, en suivant les explications données dans l'article "Génération de la documentation technique sous Netbeans".
- Transférer les documentations dans les dépôts.



Temps de réalisation estimé : 1h
Temps de réalisation réel : 1h

Commentaires normalisés

J’ai vérifié que les commentaires normalisés étaient bien ajoutés partout. Voici un exemple de commentaire :

```
/// <summary>
/// Vérifie les identifiants et retourne l'utilisateur si trouvé
/// </summary>
/// <param name="login">login de l'utilisateur</param>
/// <param name="pwd">mot de passe de l'utilisateur</param>
1 référence | Nathan Boudier, Il y a moins de 5 minutes | 1 auteur, 2 modifications
public Utilisateur GetUtilisateur(string login, string pwd)
```

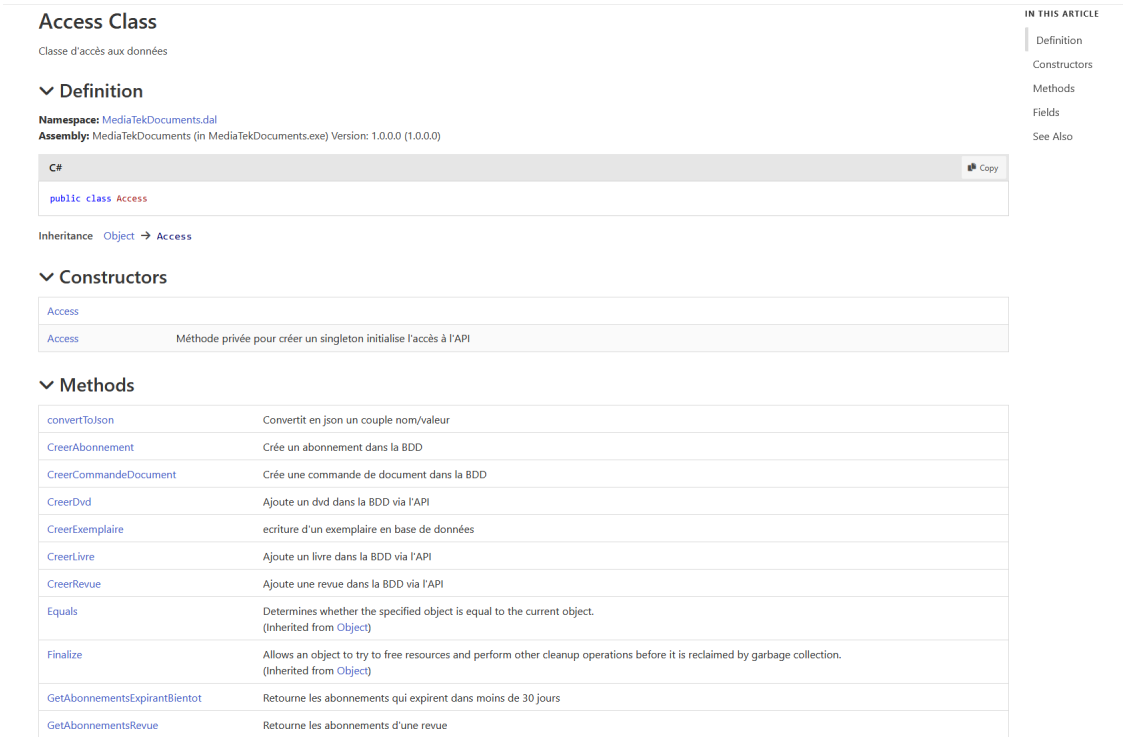
Documentations techniques

Pour retirer les erreurs du type :
[Missing <summary> documentation for "T:MediaTekDocuments.NamespaceDoc"]

J’ai ajouté le code suivant pour chaque namespace, avec l’attribut EditorBrowsable pour que ces internal class n’apparaissent pas dans la documentation :

```
/// <summary>
/// Namespace contenant la couche d'accès aux données
/// </summary>
[EditorBrowsable(EditorBrowsableState.Never)]
0 références | Nathan Boudier, Il y a 7 minutes | 1 auteur, 1 modification
internal class NamespaceDoc { }
```

Voici un extrait de la documentation de l'application mediatekdocuments :



Access Class

Classe d'accès aux données

IN THIS ARTICLE
- Definition
- Constructors
- Methods
- Fields
- See Also

Definition

Namespace: MediaTekDocuments.dal
Assembly: MediaTekDocuments (in MediaTekDocuments.exe) Version: 1.0.0.0 (1.0.0.0)

C# Copy

```
public class Access
```

Inheritance Object → Access

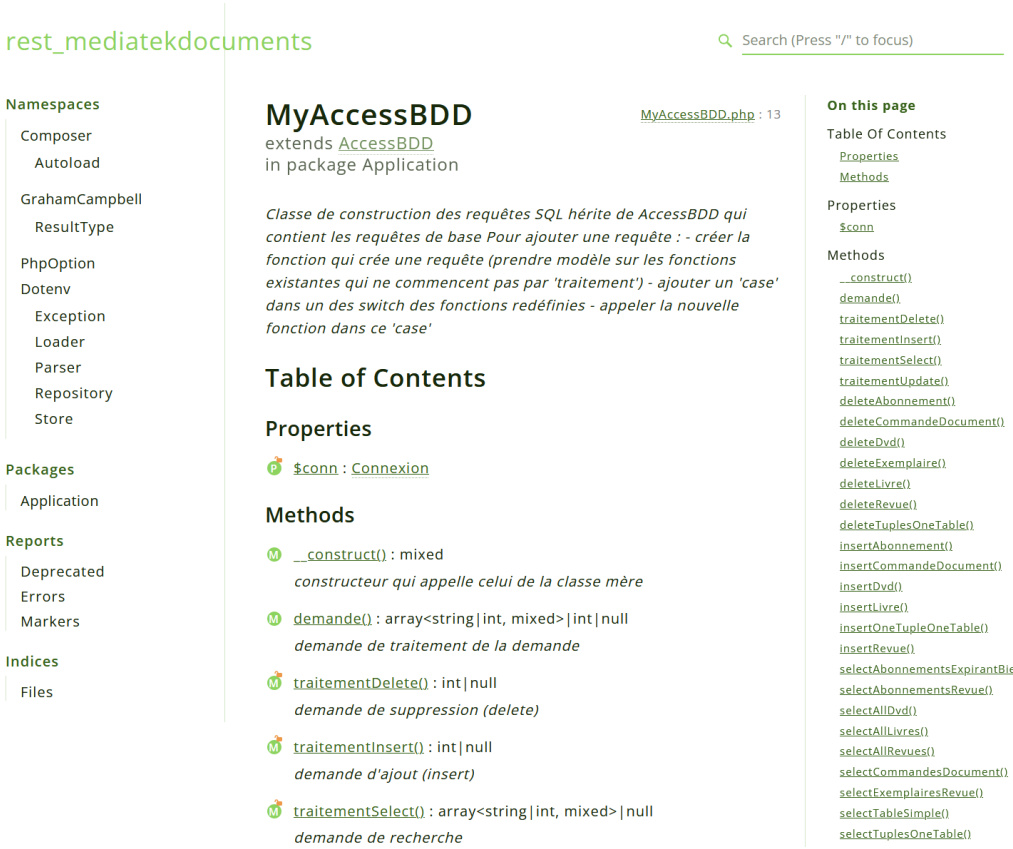
Constructors

Access	
Access	Méthode privée pour créer un singleton initialise l'accès à l'API

Methods

convertToJson	Convertit en json un couple nom/valeur
CreerAbonnement	Crée un abonnement dans la BDD
CreerCommandeDocument	Crée une commande de document dans la BDD
CreerDvd	Ajoute un dvd dans la BDD via l'API
CreerExemplaire	ecriture d'un exemplaire en base de données
CreerLivre	Ajoute un livre dans la BDD via l'API
CreerRevue	Ajoute une revue dans la BDD via l'API
Equals	Determines whether the specified object is equal to the current object. (Inherited from Object)
Finalize	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object)
GetAbonnementsExpirantBientot	Retourne les abonnements qui expirent dans moins de 30 jours
GetAbonnementsRevue	Retourne les abonnements d'une revue

Et voici un extrait de la documentation de l'API :



rest_mediatekdocuments

Search (Press "/" to focus)

Namespaces
- Composer
  - Autoload
- GrahamCampbell
  - ResultType
- PhpOption
- Dotenv
  - Exception
  - Loader
  - Parser
  - Repository
  - Store

Packages
- Application

Reports
- Deprecated
- Errors
- Markers

Indices
- Files

MyAccessBDD

MyAccessBDD.php : 13

extends AccessBDD
in package Application

Classe de construction des requêtes SQL hérite de AccessBDD qui contient les requêtes de base Pour ajouter une requête : - créer la fonction qui crée une requête (prendre modèle sur les fonctions existantes qui ne commencent pas par 'traitement') - ajouter un 'case' dans un des switch des fonctions redéfinies - appeler la nouvelle fonction dans ce 'case'

Table of Contents

Properties

$conn : Connexion

Methods

__construct() : mixed
constructeur qui appelle celui de la classe mère

demande() : array<string|int, mixed>|int|null
demande de traitement de la demande

traitementDelete() : int|null
demande de suppression (delete)

traitementInsert() : int|null
demande d'ajout (insert)

traitementSelect() : array<string|int, mixed>|null
demande de recherche

On this page

Table Of Contents
- Properties
- Methods

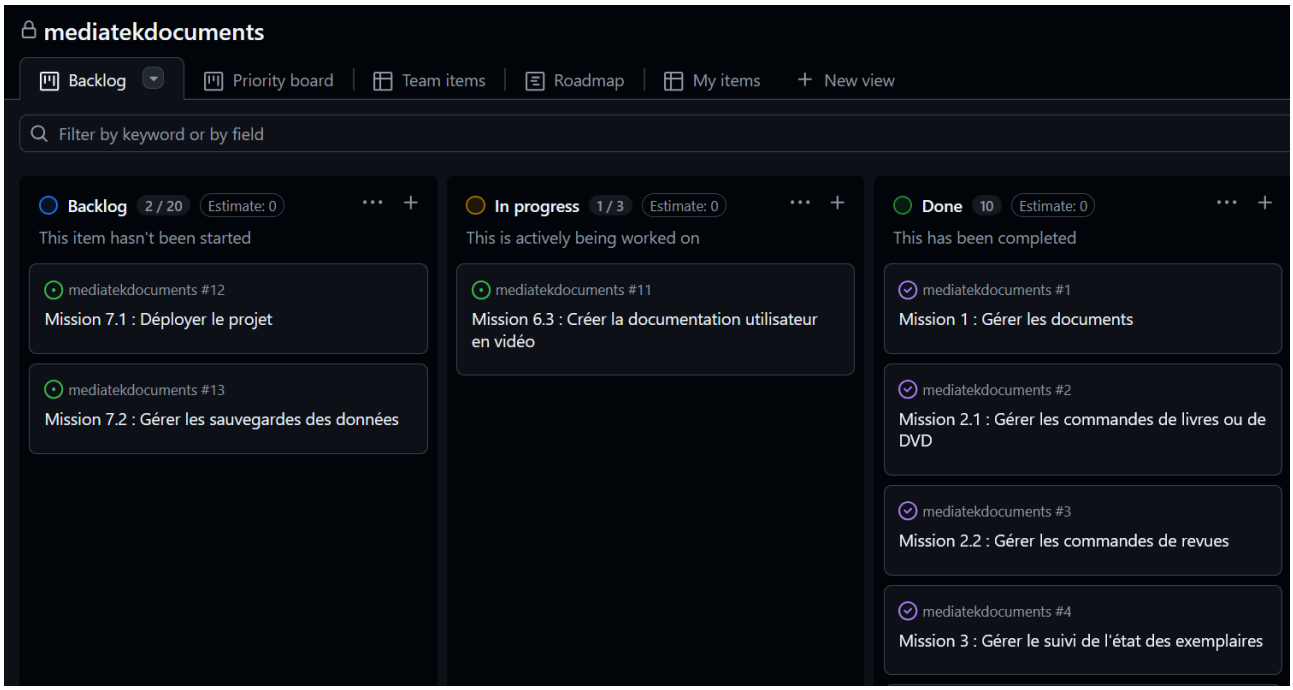
Properties
- $conn

Methods
- __construct()
- demande()
- traitementDelete()
- traitementInsert()
- traitementSelect()
- traitementUpdate()
- deleteAbonnement()
- deleteCommandeDocument()
- deleteDvd()
- deleteExemplaire()
- deleteLivre()
- deleteRevue()
- deleteTuplesOneTable()
- insertAbonnement()
- insertCommandeDocument()
- insertDvd()
- insertLivre()
- insertOneTupleOneTable()
- insertRevue()
- selectAbonnementsExpirantBientot()
- selectAbonnementsRevue()
- selectAllDvd()
- selectAllLivres()
- selectAllRevues()
- selectCommandesDocument()
- selectExemplairesRevue()
- selectTableSimple()
- selectTuplesOneTable()

Mission 6.3 : Créer une documentation utilisateur en vidéo

Demandes de la mission

Créer une vidéo de 10mn maximum qui présente l'ensemble des fonctionnalités de l'application C#.



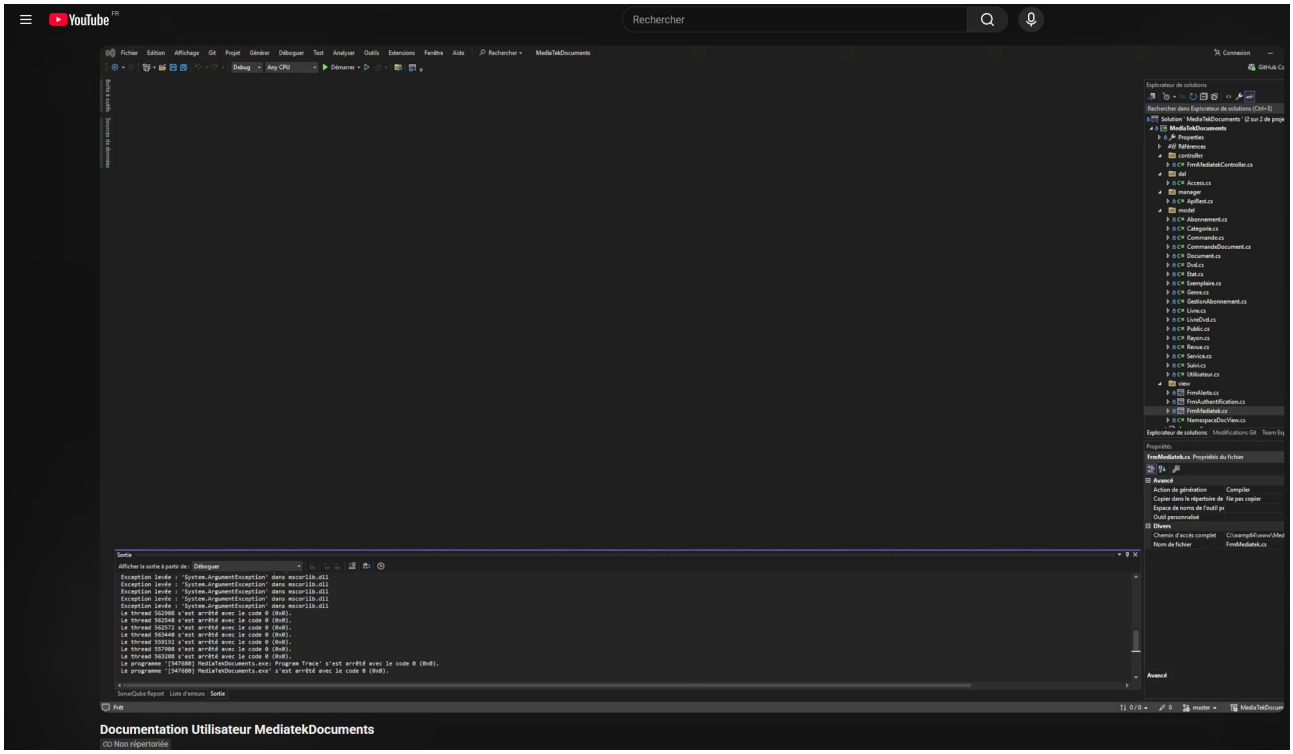
Temps de réalisation estimé : 2h
Temps de réalisation réel : 2h

Vidéo

La documentation utilisateur est disponible ici :
Documentation Utilisateur MediatekDocuments

J’ai réalisé la vidéo avec le logiciel OBS Studio.

Durée de la vidéo : 9:47



Mission 7.1 : Déployer le projet

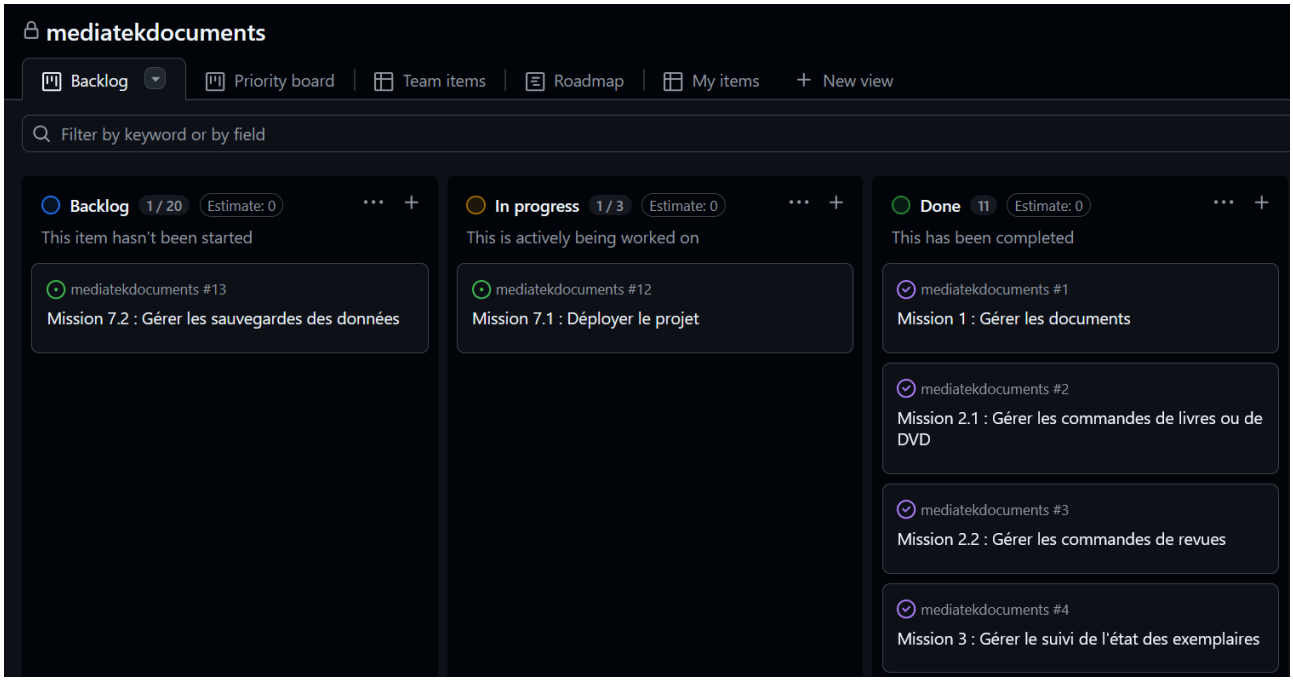
Demandes de la mission

Mettre en ligne l'API :

- Renforcer la sécurité de l'API : changer les username/password de l'accès à l'API (les informations de connexions à la BDD, mises dans le fichier '.env', devront être modifiées en tenant compte des informations prises chez l'hébergeur).
- Déployer l'API et la BDD. Pour cela, consultez l'article "API en ligne" sur le wiki de rest_mediatekdocuments.
- Tester l'API avec Postman.

Créer un installeur pour l'application C# :

- Modifier le code pour l'accès à l'API distante.
- Créer l'installeur, tester son installation et l'utilisation de l'application installée.
- L'envoyer vers le dépôt.



Temps de réalisation estimé : 3h
Temps de réalisation réel : 3h

Déploiement

J’ai choisit d’héberger l’API et la BDD sur OVH, là où j’ai déjà mon portfolio et l’atelier n°1.

- J’ai d’abord importé les tables de la base mediatek86 dans ma base de données sur OVH, car dans mon abonnement je n’ai le droit qu’à une base de données.
- J’ai créé le sous-domaine api-mediatekdocuments.nathan-boudier.com sur OVH avec comme dossier racine api_mediatekdocuments.
- J’ai ensuite uploadé les fichiers de l’API dans ce dossier du serveur via FileZilla.
- J’ai modifié le fichier .env pour mettre à jour les informations de connexion à la BDD.
- Enfin, j’ai modifié le .htaccess pour résoudre le problème d’authentification mentionné dans le wiki du dépôt. J’ai donc ajouté ces deux lignes :

```
RewriteCond %{HTTP:Authorization} ^(.+)$
RewriteRule ^(.*)$ - [E=HTTP_AUTHORIZATION:%1]
```

J’ai testé l’API sur Postman avec une requête GET http://api-mediatekdocuments.nathan-boudier.com/ , elle répond correctement.

Modification App.config :

J’ai modifié l’url de l’API dans App.config de l’application C# :

```
<add name="MediaTekDocuments.Properties.Settings.uriApi"
     connectionString="http://api-mediatekdocuments.nathan-
boudier.com/" />
```

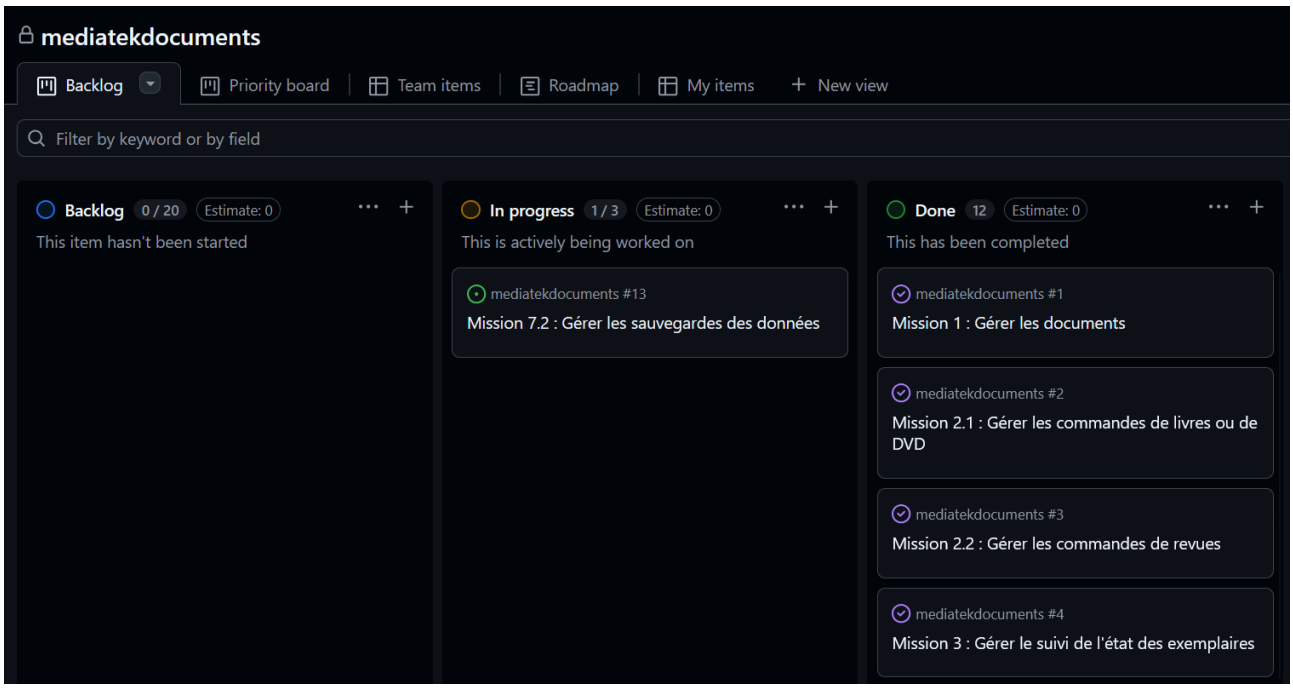
Création de l’installeur :

J’ai créé l’installeur avec l’extension Microsoft Visual Studio Installer Projects :
- J’ai ajouté un projet MediatekDocumentsInstaller à la solution
- J’ai configuré le projet et généré l’installeur
- J’ai testé l’installation de l’application et son bon fonctionnement
- Enfin, j’ai fait un push de l’installeur vers le dépôt GitHub.

Mission 7.2 : Gérer les sauvegardes automatiques des données

Demandes de la mission

- Une sauvegarde journalière automatisée doit être programmée pour la BDD : voir l'article "Automatiser la sauvegarde d'une BDD" dans le wiki du dépôt.
- La restauration pourra se faire manuellement, en exécutant le script de sauvegarde.



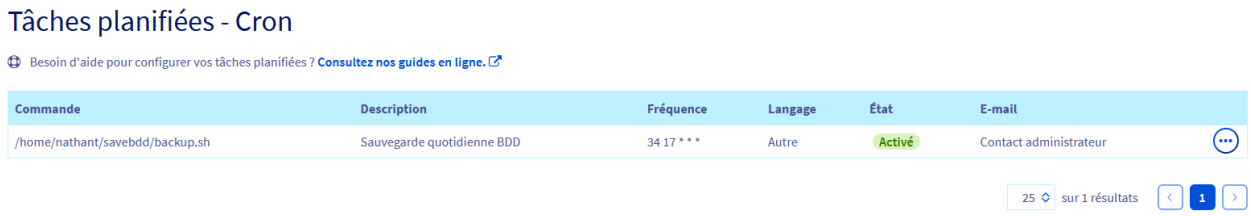
Temps de réalisation estimé : 1h
Temps de réalisation réel : 1h

Tâche Cron

J’ai créé un script backup.sh dans le dossier savebdd à la racine du compte OVH. Ce script permet de sauvegarder la BDD dans un fichier compressé nommé avec la date du jour :

```
#!/bin/sh
DATE=`date -I`
find /home/nathant/savebdd/bdd* -mtime -1 -exec rm {} \;
mysqldump -u nathantmediatek -pTON_MOT_DE_PASSE --databases
nathantmediatek --single-transaction | gzip >
/home/nathant/savebdd/bddbackup_${DATE}.sql.gz
```

J’ai ensuite configuré une tâche CRON dans l’espace OVH pour exécuter ce script une fois par jour.



Cependant j’ai constaté que cette tâche ne fonctionnait pas, sûrement à cause de la commande mysqldump qui semble ne pas être accessible dans OVH, ce qui empêche l’exécution du script.

Il y a une alternative dans OVH qui propose des sauvegardes automatiques de la base de données, qui sont consultables dans l’espace client.

Mode opératoire pour la restauration :

Pour restaurer la BDD, il faudrait d’abord récupérer le fichier de sauvegarde de la BDD dans le dossier savebdd via FileZilla. Il faudrait ensuite dézipper ce fichier pour obtenir le fichier .sql, puis se connecter au phpMyAdmin OVH et supprimer toutes les tables. Enfin il n’y aurait plus qu’a importer le fichier .sql récupéré pour restaurer la BDD.

Avec les sauvegardes automatiques OVH, il faut cliquer sur les 3 points à droite de la BDD, choisir l’option Restaurer une sauvegarde, choisir la date de sauvegarde souhaitée, et confirmer la restauration.

Bilan Final

Objectifs atteints

Toutes les missions ont été réalisées et sont opérationnelles.

La gestion des documents (mission 1) permet d'ajouter, modifier et supprimer des livres, DVD et revues dans le respect des règles imposées. La gestion des commandes (mission 2) permet de suivre les commandes de livres et DVD ainsi que les abonnements aux revues, avec une alerte automatique pour les abonnements expirant bientôt. La gestion des exemplaires (mission 3) permet de modifier l'état et de supprimer des exemplaires. Le système d'authentification (mission 4) gère trois niveaux d'accès selon le service de l'employé. La sécurité et la qualité (mission 5) ont été assurées en sécurisant les informations de connexion, en nettoyant le code et en intégrant des logs. Les tests et la documentation (mission 6) ont été réalisés avec des tests unitaires, des tests fonctionnels Postman et des documentations techniques générées. Enfin, le déploiement (mission 7) a été réalisé chez OVH avec un installeur pour l'application C#.

Problèmes rencontrés

Plusieurs difficultés ont été rencontrées au cours du projet. La gestion des transactions PHP a nécessité l'ajout de méthodes dans la classe Connexion. Le nommage des clés JSON entre le C# et le PHP a causé des erreurs car le C# sérialise avec des majuscules alors que le PHP attendait des minuscules. La clé primaire composée de la table exemplaire a causé des bugs lors des modifications et suppressions. Enfin, le déploiement sur OVH a nécessité des ajustements du .htaccess pour que l'authentification Basic Auth fonctionne correctement.