Atelier professionnel n°2 : MediaTekDocuments

Compte rendu de l'activité

FRANCART Jérémy
BTS SIO DEUXIEME ANNEE

Sommaire :

Contexte de l'atelier

Cet atelier prend place au sein de l'entreprise InfoTech Services 86 dans laquelle nous travaillons en tant que développeur junior. Cette entreprise a remporté des appels d'offres pour effectuer de multiples interventions pour le réseau de MediaTek86. MediaTek86 est un réseau qui gère les médiathèques de la Vienne, et qui a entre autres pour rôle développer la médiathèque numérique pour l'ensemble des médiathèques du département. C'est dans ce dernier but que MediaTek86 a fait recours aux services de InfoTech Services 86. Parmi ces interventions, nous avons été confiés la création d'une application de gestion des différents documents de la médiathèque afin de pouvoir assurer la disponibilité des livres, DVDs et revues aux utilisateurs

Mission globale

Notre mission consiste à développer et déployer une application en C# utilisant Windows Forms. Cette application devra exploiter une API REST construite en PHP qui fait interface avec la base de données. Depuis l'application, il faut gérer les manipulations suivantes : lister les documents de la médiathèque (livres, DVDs, revues), lister les exemplaires d'un document, lister les commandes d'un livre ou DVD, lister les abonnements d'une revue. Si l'utilisateur qui s'authentifie à l'application est habilité, il pourra alors aussi ajouter, modifier, supprimer les documents ainsi que leurs exemplaires, commandes / abonnements. Aussi, il faudra aussi construire une documentation technique et utilisateur.

Actuellement, l'application, à partir de l'API, permet de lister les documents et les parutions (exemplaire d'une revue) ainsi que d'ajouter une nouvelle parution. Nous devons maintenant développer les fonctionnalités liées à l'ajout, modification et suppression des documents, des exemplaires et des commandes

Nous gérerons de multiples aspects qui vont être divisés en plusieurs étapes :

Étape 1 : Préparation de l'environnement de travail

Étape 2 : Gestion des documents

Étape 3 : Gestion des commandes

Étape 4 : Gestion du suivi de l'état des exemplaires

Étape 5 : Mise en place des authentifications

Étape 6 : Assurance de la sécurité, de la qualité et intégration des logs

Étape 7 : Mise en place de tests et de la documentation

Étape 8 : Déploiement de la solution et gestion des sauvegardes et restauration

Réalisation de la mission

Étape 1 : Préparation de l'environnement de travail

Avant de pouvoir travailler sur l'application, il faut que l'on prépare notre environnement de travail local.

On commence par récupérer les dépôts de l'application C# ainsi que de l'API REST actuelle.

On installera l'API REST localement en utilisant WampServer.

Puis, on installe les bundles utilisés par l'API REST en exécutant "composer install".

On crée et importe la base de données actuelle avec le script de sauvegarde de la base de données depuis phpMyAdmin

Nous pouvons maintenant vérifier que l'API est bien fonctionnelle avec Postman que l'on utilisera pour tester nos différentes requêtes :

Maintenant, nous allons configurer nos IDEs pour pouvoir travailler sur le code source.

Ici, les IDEs Rider et Visual Studio ont été utilisés pour l'application C#. Pour l'API REST, l'IDE PhpStorm a été utilisé.

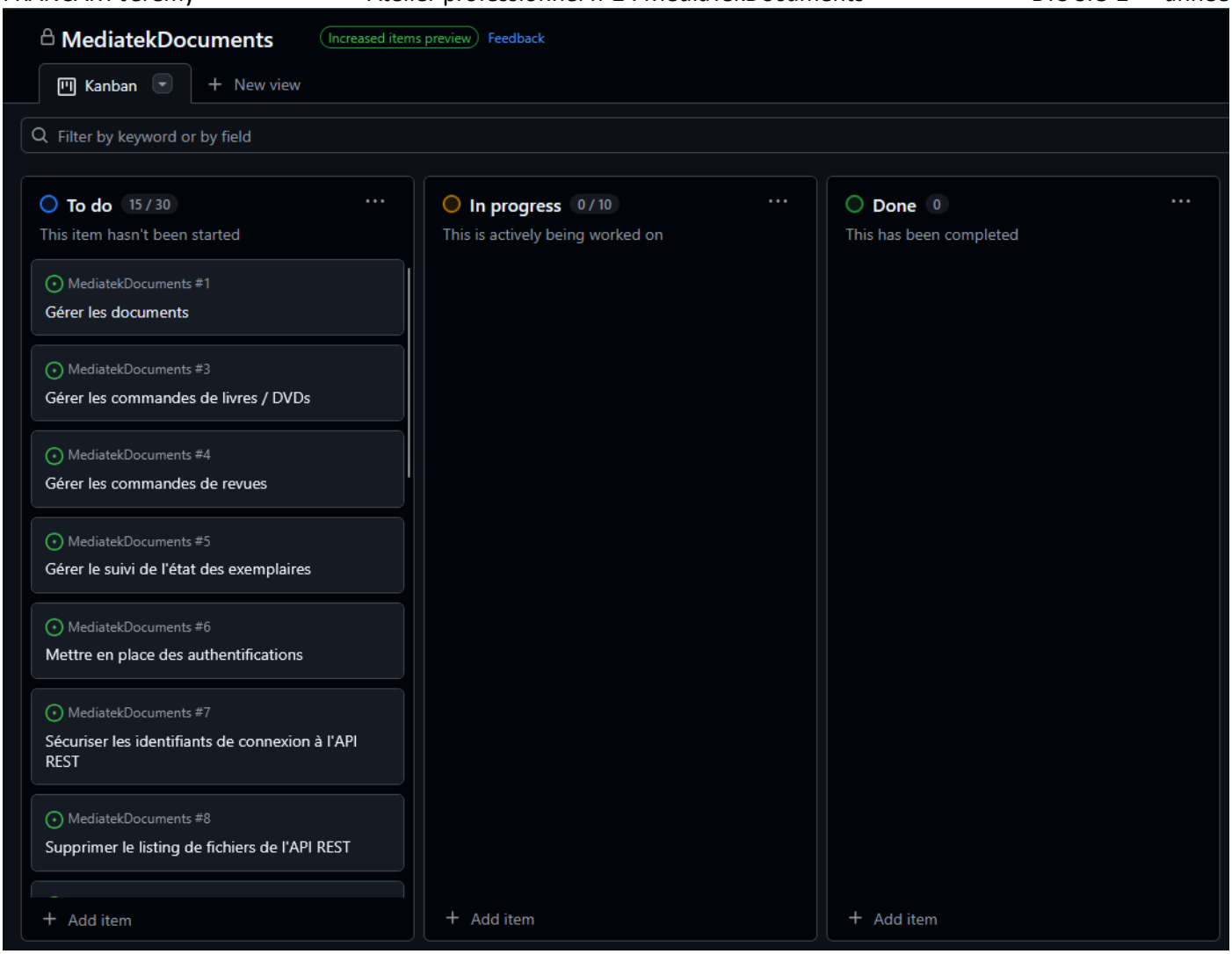
On installe les packages NuGet Newtonsoft.Json et Microsoft.AspNet.WebApi.Client afin de régler les erreurs lors de la compilation de l'application C#.

Maintenant, j'ai passé un temps pour tester et analyser le code pour mieux comprendre comment l'application fonctionne.

Ensuite, nous créons les dépôts GitHub pour l'application et l'API puis on push la version actuelle.

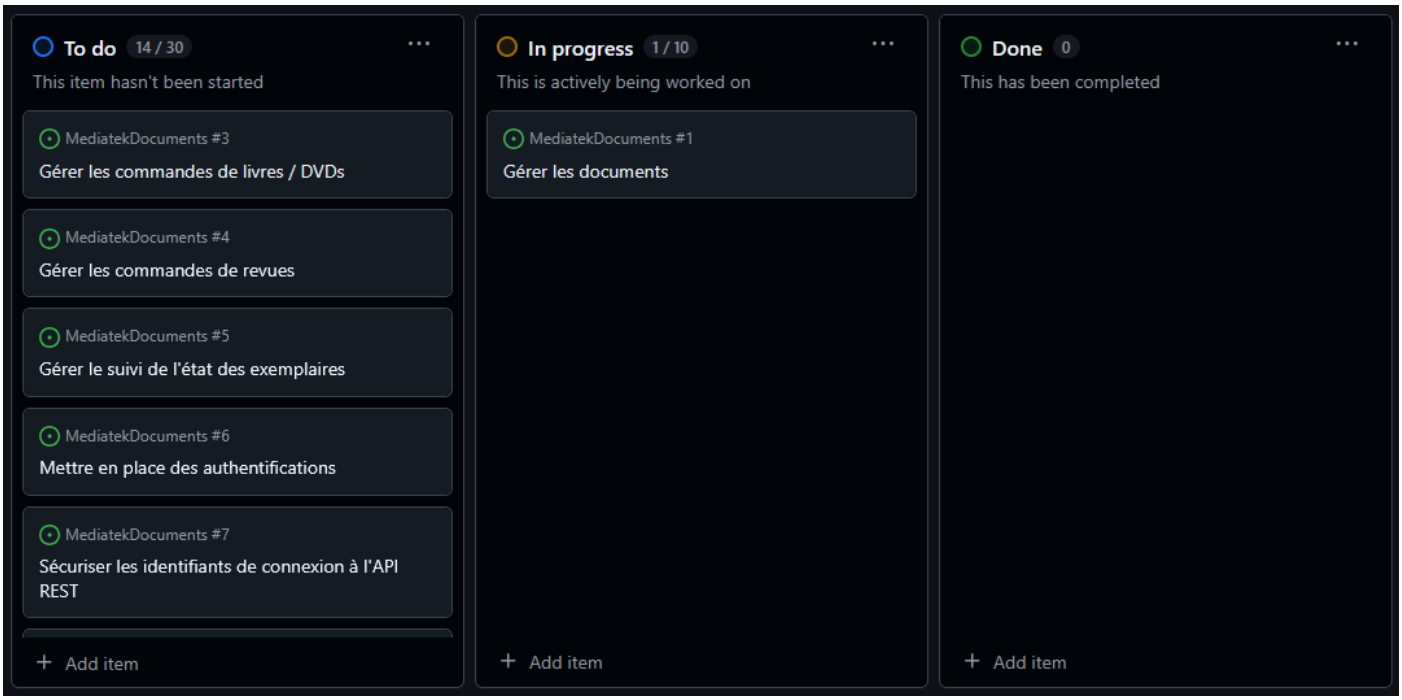
Enfin, on va utiliser un Kanban automatisé pour pouvoir gérer les tâches et l'avancement du projet, on crée un Projet GitHub et on configure un Kanban automatisé qui va créer un élément pour chaque issue créée et va automatiquement clôturer les issues quand l'élément est placé dans "Done".

Une fois le Kanban créé, on remplit les tâches à réaliser en créant une issue par tâche.


MediatekDocuments
Increased items preview
Feedback
Kanban
New view
Filter by keyword or by field
To do 15 / 30
This item hasn't been started
MediatekDocuments #1
Gérer les documents
MediatekDocuments #3
Gérer les commandes de livres / DVDs
MediatekDocuments #4
Gérer les commandes de revues
MediatekDocuments #5
Gérer le suivi de l'état des exemplaires
MediatekDocuments #6
Mettre en place des authentifications
MediatekDocuments #7
Sécuriser les identifiants de connexion à l'API REST
MediatekDocuments #8
Supprimer le listing de fichiers de l'API REST
Add item
In progress 0 / 10
This is actively being worked on
Add item
Done 0
This has been completed
Add item

Étape 2 : Gestion des documents

Temps de réalisation estimé : 8h - Temps de réalisation réel : 12h

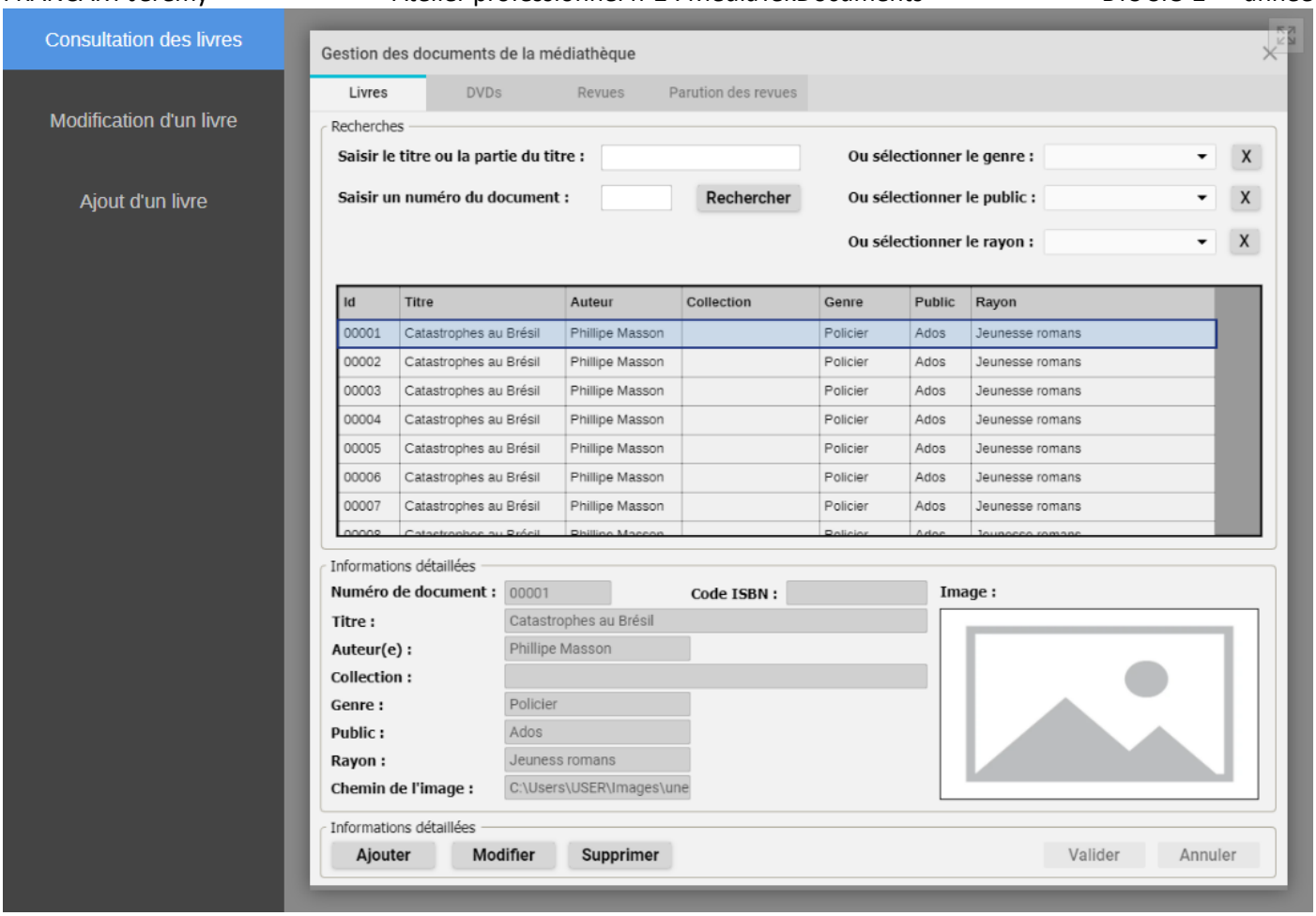


Dans cette étape, nous allons ajouter la gestion des documents (livres, DVD, revues). C'est-à-dire la consultation (affichage en liste et en détail, avec possibilités de tris, recherches et filtres), l'ajout, la modification (excepté l'identifiant du document) et la suppression (uniquement s'il n'y a pas de commandes et d'exemplaires du document). Toutes les sécurités doivent être mises en place pour éviter des erreurs de manipulation.

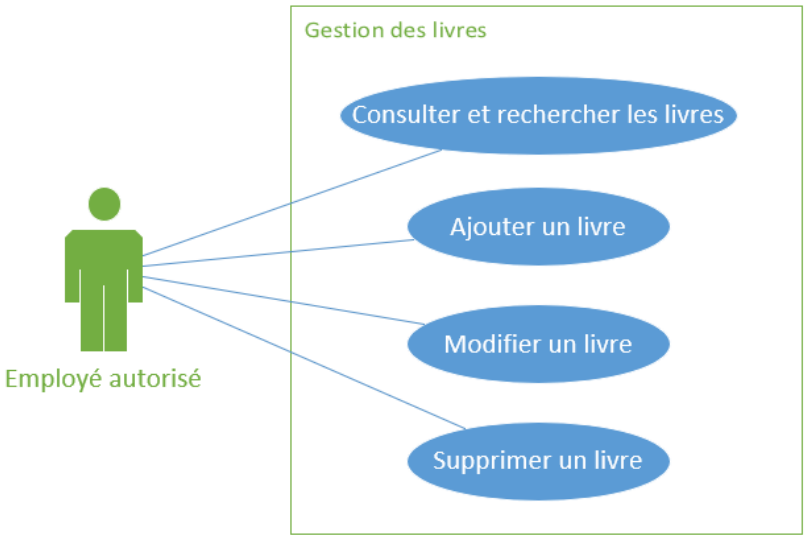
Commençons par nous concentrer par la gestion des livres dans l'onglet livres.

On commence par réaliser une maquette correspondant à cette tâche, il a été décidé d'ajouter une zone d'actions en bas de la fenêtre, les actions valider et annuler ne seront disponibles que lorsque l'on effectue une addition ou une modification tandis que les actions ajouter, modifier et supprimer ne seront disponibles que lorsque l'on n'est pas en cours d'addition ou de modification. Aussi, les actions modifier et supprimer ne seront disponibles que si un document valide est sélectionné.

Voici ce que cela donne :



Aussi, voici le diagramme de cas d'utilisation correspond à la tâche :



(nous nous occuperons des autorisations d'un employé ultérieurement.)

Nous allons maintenant implémenter ce design dans l'application. On modifie la hauteur de la fenêtre pour venir accompagner la zone d'actions, on ajoute la zone d'action avec les boutons puis on connecte l'événement de clic sur les boutons à des méthodes événementielles.

Au clic sur le bouton ajouter et modifier, on désactive le changement d'onglet et la zone de recherche. Aussi, on active la zone des informations détaillées d'où l'on peut changer les informations du document et on active également les boutons valider et annuler

On active le champ du numéro de document seulement en cas d'ajout

Enfin, on affiche un bouton permettant de sélectionner une image depuis une boite de dialogue de l'explorateur de fichiers.

Au clic sur le bouton valider, selon la modification ou l'ajout en cours, le formulaire FrmMediatek va faire appel à la bonne méthode du contrôleur qui va ensuite rediriger la demande vers la classe Access. Ensuite, la classe Access va effectuer une requête vers l'API REST. Les classes modifiées sont donc :

- FrmMediatekController.cs : Ajout des demandes de création, modification et suppression de livres
- FrmMediatek.cs/.Designer.cs : Ajout des nouveaux composants graphiques, gestion des actions et de l'interface graphique pour l'ajout, la modification et la suppression d'un livre
- Access.cs : Ajout des nouvelles méthodes pour créer, modifier, supprimer et vérifier si on peut supprimer un livre

L'API REST a également été modifiée pour prendre en compte ces nouvelles fonctionnalités :

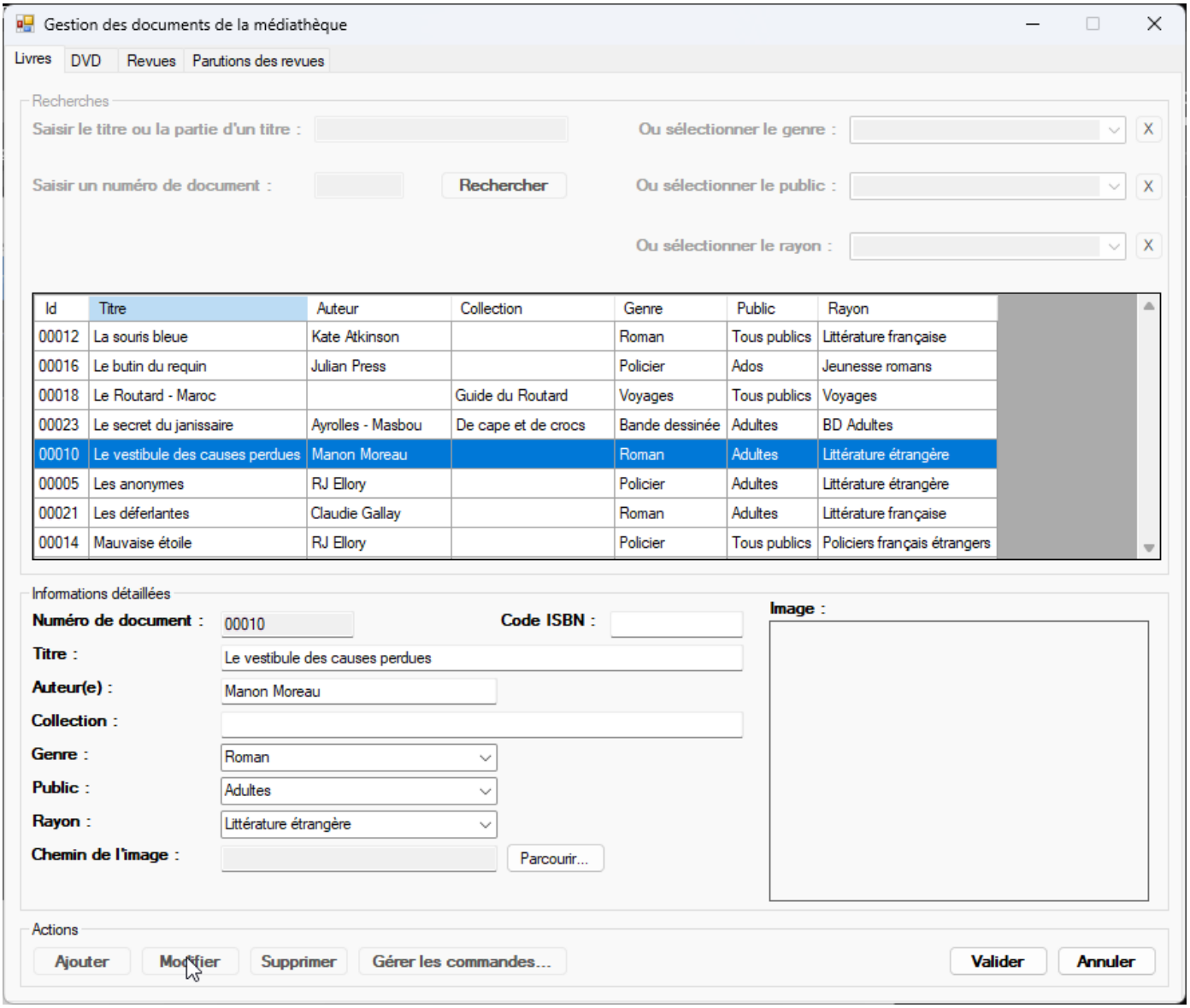
- Ajout des procédures beginTransaction(), commit() et rollback() dans le fichier Connexion.php ce qui nous permettra de gérer les transactions dans les requêtes SQL de l'API
- Les signatures de traitementInsert et traitementUpdate ont aussi été modifiées (dans la classe MyAccessBDD et la classe mère AccessBDD) afin de pouvoir retourner des listes à l'application cliente. Cela nous permettra de retourner un objet créé ou modifié en tant que confirmation par exemple

```
63  -   abstract protected function traitementInsert(string $table, ?array $champs) : ?int;
64  -   abstract protected function traitementUpdate(string $table, ?string $id, ?array $champs) : ?int;
63  +   abstract protected function traitementInsert(string $table, ?array $champs) : int|array|null;
64  +   abstract protected function traitementUpdate(string $table, ?string $id, ?array $champs) : int|array|null;
```

- Implémentation des méthodes d'insert, update et delete au sein de transactions car on doit effectuer plusieurs requêtes et si l'une échoue, il faut abandonner la transaction pour revenir à l'état initial. Exemple avec l'insertion :

```php
/**
 * Insertion d'un nouveau livre dans la base de données
 * @param array|null $champs Les champs de la requête contenant tous les champs nécessaires à la création d'un livre
 * @return array[]|null Les champs passés en paramètres en cas d'insertion réussie ou null en cas d'erreur
 */
private function insertLivre(?array $champs): array|null {
    if (empty($champs)){
        return null;
    }

    if (!$this->conn->beginTransaction()) {
        return null;
    }

    if (!$this->insertDocument($champs['Id'], $champs['Titre'], $champs['Image'], $champs['IdRayon'], $champs['IdPublic'], $champs['IdGenre'])) {
        $this->conn->rollback();
        return null;
    }

    $requete = 'insert into livres_dvd values(:id);';
    if(!$this->conn->updateBDD($requete, ['id' => $champs['Id']])) {
        $this->conn->rollback();
        return null;
    }

    $requete = 'insert into livre values(:id, :isbn, :auteur, :collection);';
    if(!$this->conn->updateBDD($requete, [
        'id' => $champs['Id'],
        'isbn' => $champs['Isbn'],
        'auteur' => $champs['Auteur'],
        'collection' => $champs['Collection']
    ])) {
        $this->conn->rollback();
        return null;
    }

    $this->conn->commit();
    return [$champs];
}
```

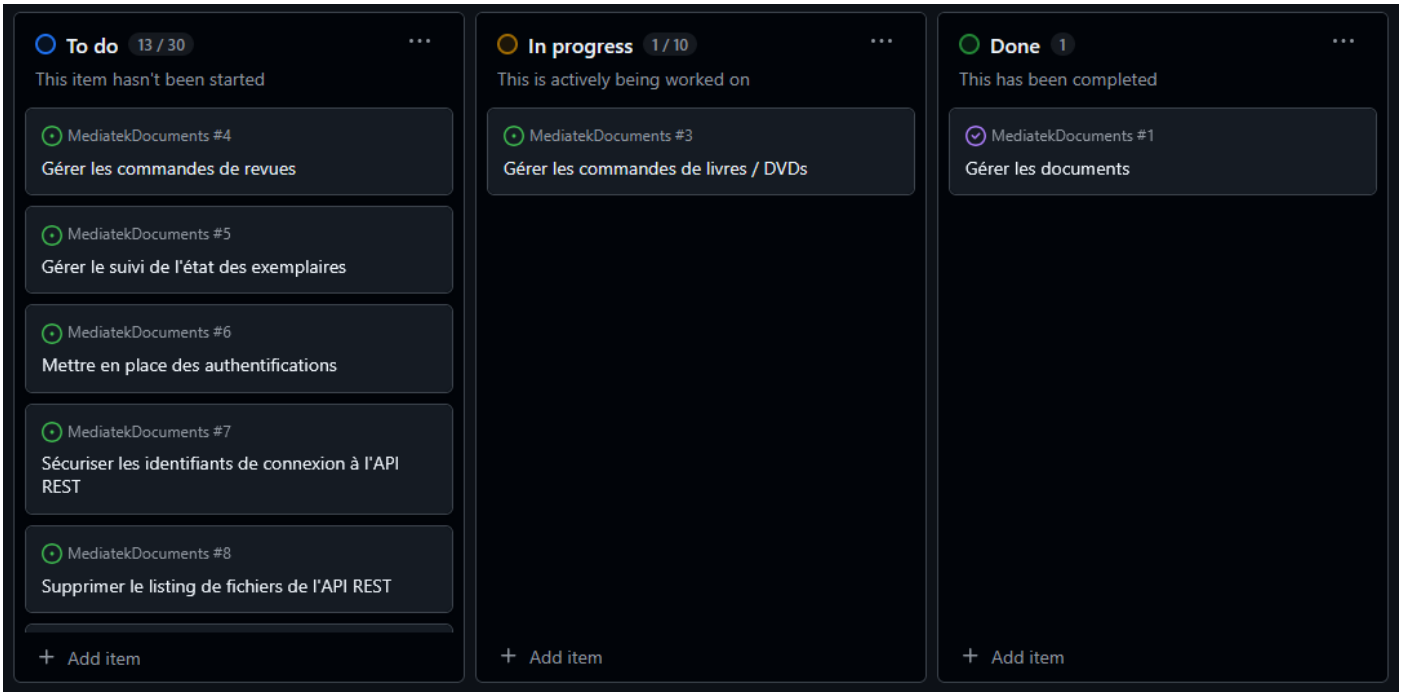
Une fois la gestion des livres implémentée, on implémente de la même manière la gestion des DVDs et la gestion des revues. Voici le résultat avec par exemple la modification d'un livre :



La réalisation de cette tâche m'a pris plus de temps que prévu le temps que je me familiarise avec le code source des 2 dépôts ainsi que le temps de trouver une démarche cohérente avec la demande et enfin la réalisation de la maquette et des diagrammes de cas d'utilisation (pour la maquette, seulement une page a été maquettée car les autres pages étaient très similaires). Le reste des maquettes et des diagrammes de cas d'utilisation sont disponibles sur la page dédiée au projet sur mon portfolio.

Étape 3 : Gestion des commandes

Tâche 1 : Gérer les commandes de livres ou de DVDs
Temps de réalisation estimé : 8h - Temps de réalisation réel : 8h



On commence par créer la table "suivi" contenant les différentes stades possibles d'une commande puis on la remplit avec les différents stades possibles :

```
CREATE TABLE suivi (
   `id` VARCHAR(5) NOT NULL,
   `stade` VARCHAR(20) NOT NULL,
   PRIMARY KEY (`id`)
) ENGINE = InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

```
INSERT INTO suivi VALUES ("00001", "En cours"), ("00002", "Relancée"), ("00003", "Livrée"), ("00004", "Réglée");
```

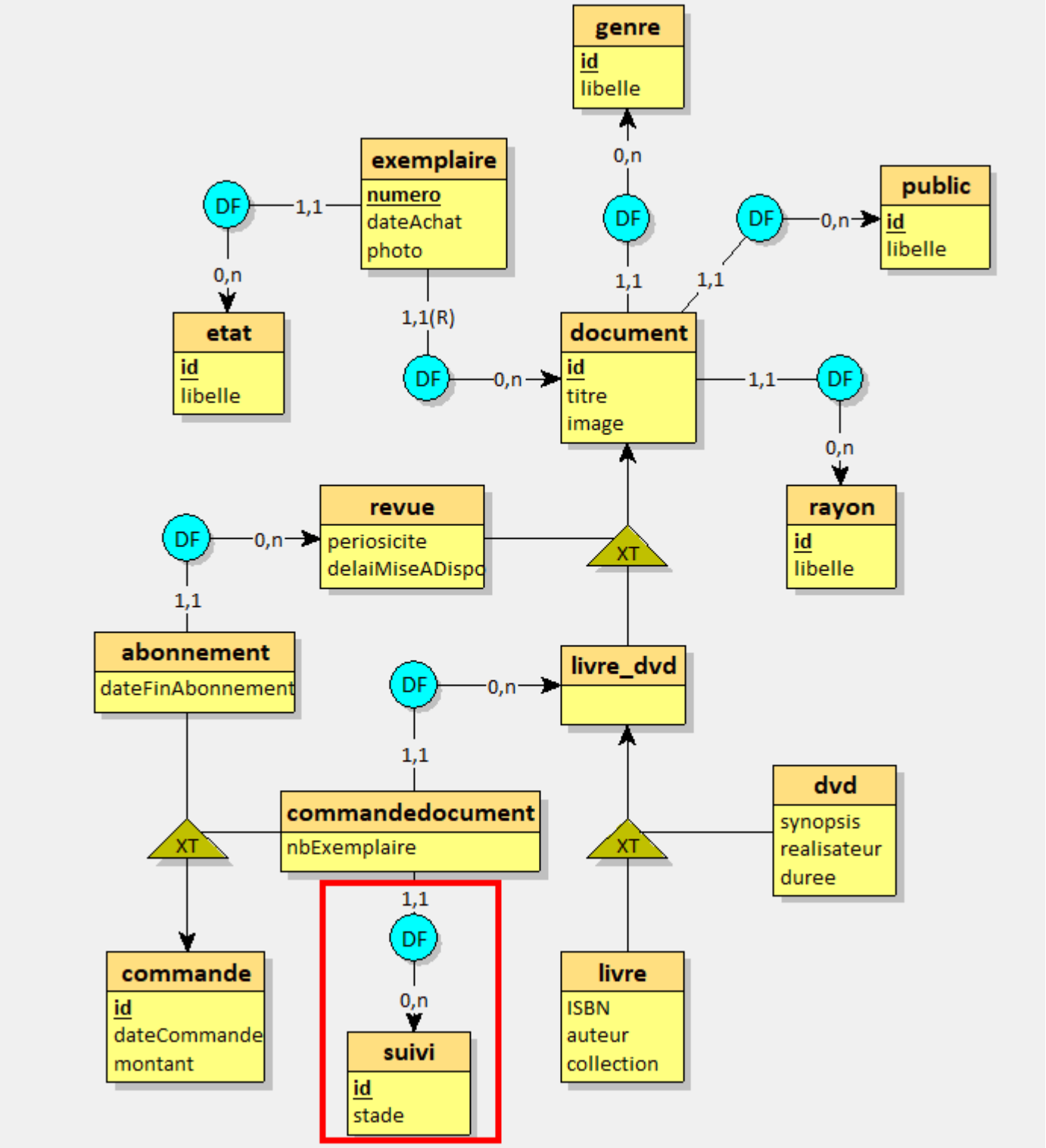
Maintenant, on va relier cette table à la table "commandedocument" en ajoutant l'identifiant du suivi dans la table "commandedocument" :

```
ALTER TABLE commandedocument
ADD COLUMN idSuivi VARCHAR(5) NOT NULL;
ALTER TABLE commandedocument
ADD CONSTRAINT fk_idSuivi FOREIGN KEY (idSuivi) REFERENCES suivi(id);
```

Enfin, on met à jour les commandes actuelles pour leur donner l'état de suivi par défaut "En cours" :

```
UPDATE commandedocument
SET idSuivi = "00001";
```

Voici ce que cela donne au niveau du modèle conceptuel de données :



Une commande de document est liée à un suivi et un suivi peut avoir de multiples commandes de document liées.

Concernant l'ajout dans l'application, il a été décidé de créer une nouvelle fenêtre pour ne pas trop encombrer la fenêtre principale. Cette fenêtre est accessible après avoir choisi un livre / DVD et cliqué sur « Gérer les commandes ». La fenêtre en elle-même est similaire à la fenêtre principale. Les traitements ont juste été adaptés aux commandes.

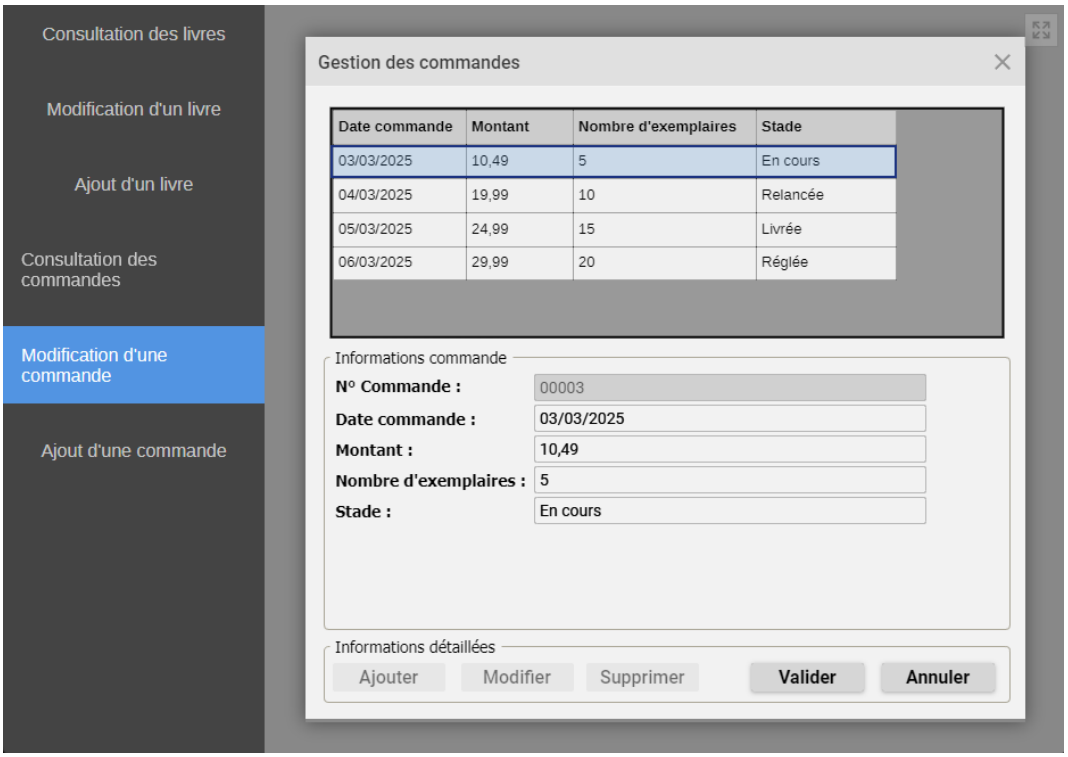
Le code de la nouvelle fenêtre est dans la classe FrmMediatekCommandesLivreDvd

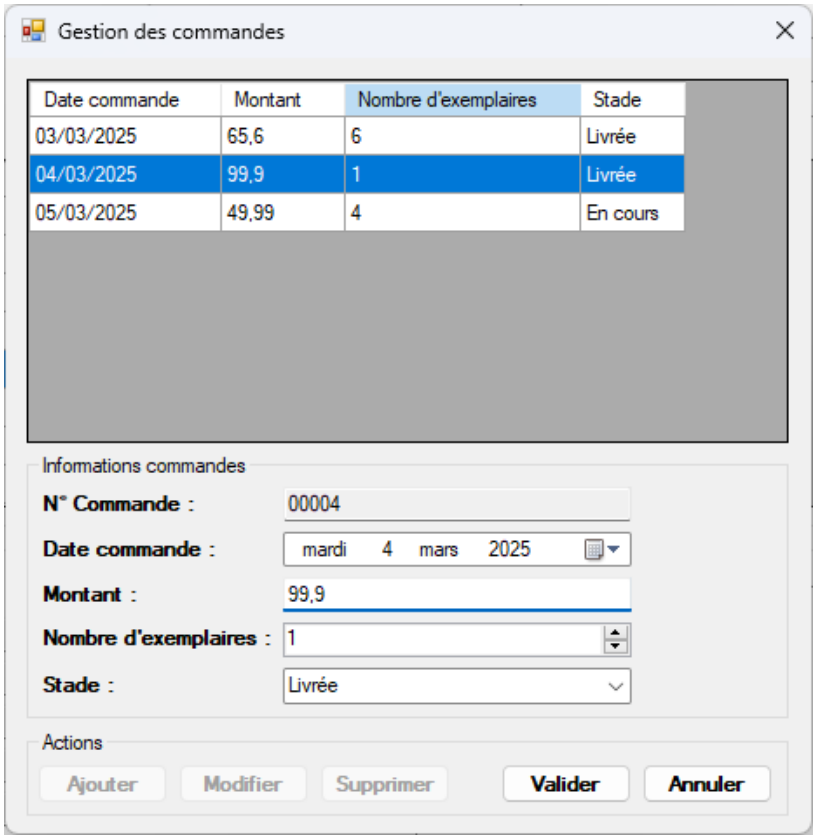
On va réutiliser la même fenêtre pour les livres et les DVDs comme le fonctionnement d'une commande est le même, on peut utiliser la classe LivreDvd pour représenter ou un livre ou un DVD

Au niveau de l'API REST, on a rajouté quelques méthodes dans MyAccessBDD.php pour gérer les insertions et mises à jour de commandes de dvds / livre qui seront exécutées si la table envoyée en paramètre est "commandedocument"

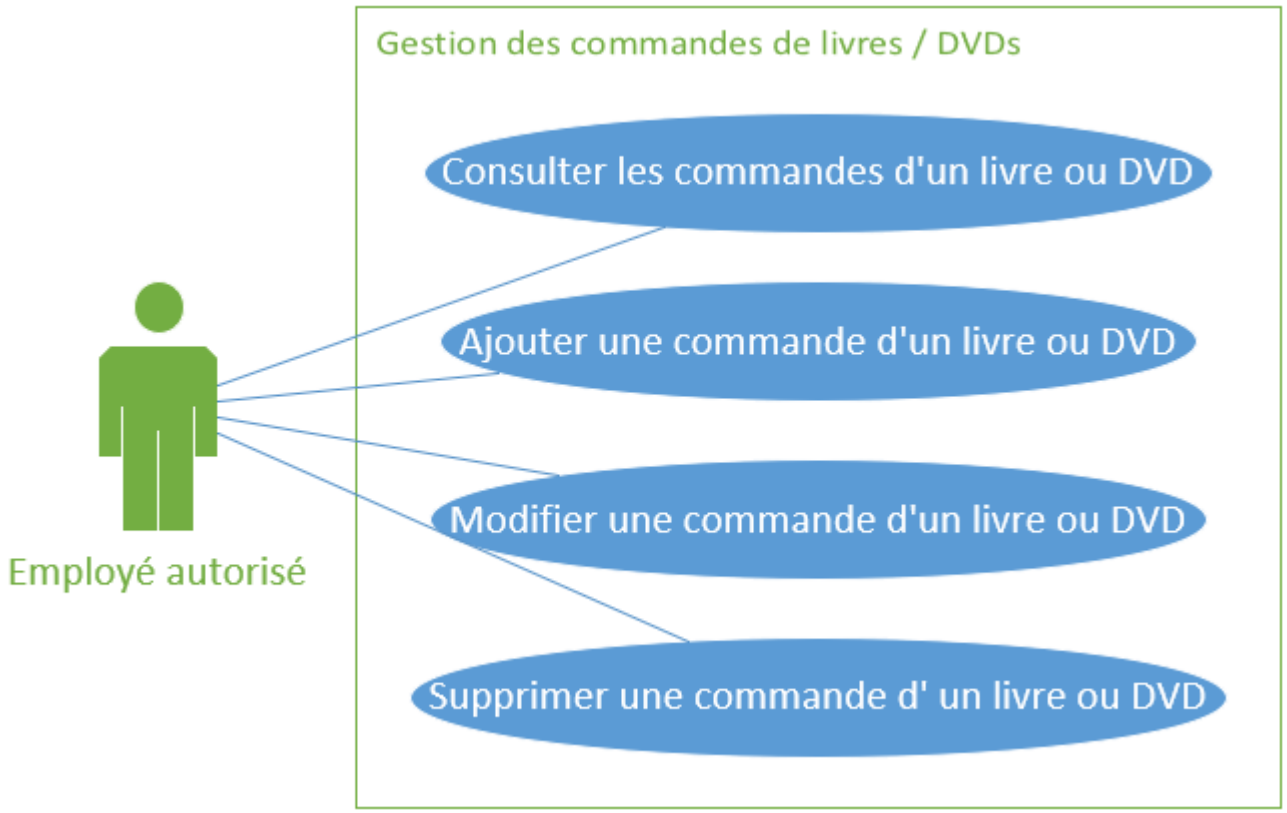
Aussi, le script de la bdd a été mis a jour dans le dépôt de l'API REST.

Voici la maquette puis le résultat final :





Et voici le diagramme de cas d'utilisation :



Des vérifications avant ajout et modification ont aussi dû être implémentées :

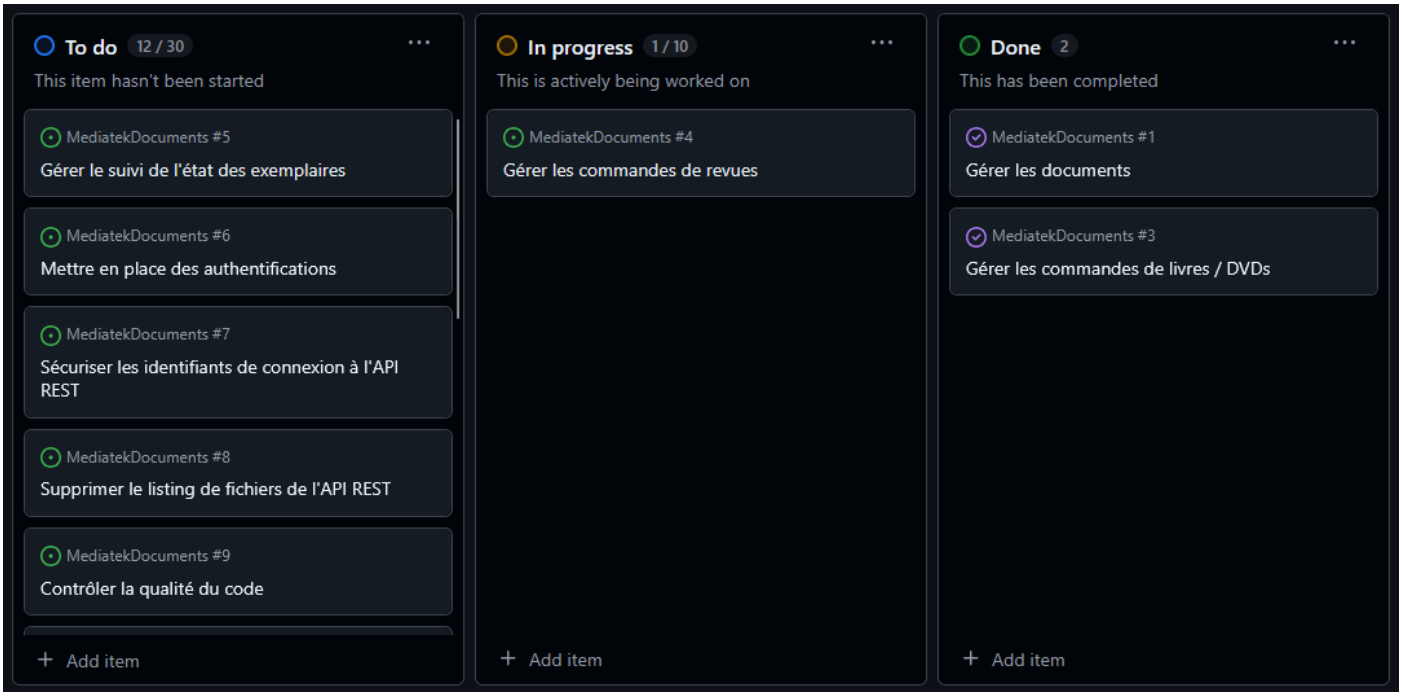
- L'étape de suivi ne peut passer de « Livrée » ou « Réglée » à « En cours » ou « Relancée »
- Une commande ne peut pas être réglée si elle n'est pas livrée
- Une commande ne peut être supprimée si elle a déjà été livrée

Aussi, afin de rendre les traitements plus sécurisés et automatisés, 2 triggers ont été créés au niveau de la base de données :

- Un trigger qui supprime la commande de la table "commande" après avoir supprimé la commande de la table "commandedocument" (table mère / fille)
- Un trigger qui insert des exemplaires dans la table "exemplaire" une fois qu'une commande passe à l'état livrée

Tâche 2 : Gérer les commandes de revues

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



Cette demande est en réalité très similaire à la gestion des commandes de livre / DVD.

On ajoute dans la classe Model les classes Abonnement représentant un abonnement d'une revue et la classe Commande qui servira de classe mère aux classes CommandeLivreDvd et Abonnement, comme c'est déjà le cas dans la base de données.

De la même manière que pour les commandes de livre / DVD, on créé une fenêtre pour la gestion des abonnements d'une revue accessible depuis un bouton "Gestion des abonnements". Depuis cette fenêtre, on ajoute les actions pour ajouter, supprimer et modifier un abonnement.

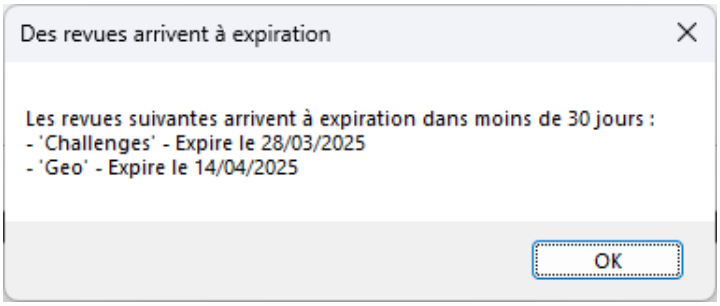
Aussi, on ajoute un trigger pour supprimer le tuple dans commande après suppression dans abonnement (similaire au trigger dans "commandedocument")

On gère également les restrictions sur la suppression d'un abonnement : aucun exemplaire ne doit être rattaché. Pour vérifier cette restriction, on créé une méthode dans la classe Abonnement pour vérifier si un exemplaire est paru dans un abonnement :

```
/// <summary>
/// Vérifie si une date d'un exemplaire d'une revue est paru dans un abonnement
/// </summary>
/// <param name="dateParution">La date de parution</param>
/// <returns>True si la date de parution est comprise dans l'abonnement, sinon false</returns>
public bool ParutionDansAbonnement(DateTime dateParution)
{
    return dateParution >= DateCommande && dateParution < DateFinAbonnement;
}
```

On a ensuite créé un projet de tests avec un test unitaire sur cette méthode, nous verrons plus d'informations sur la gestion des tests ultérieurement.

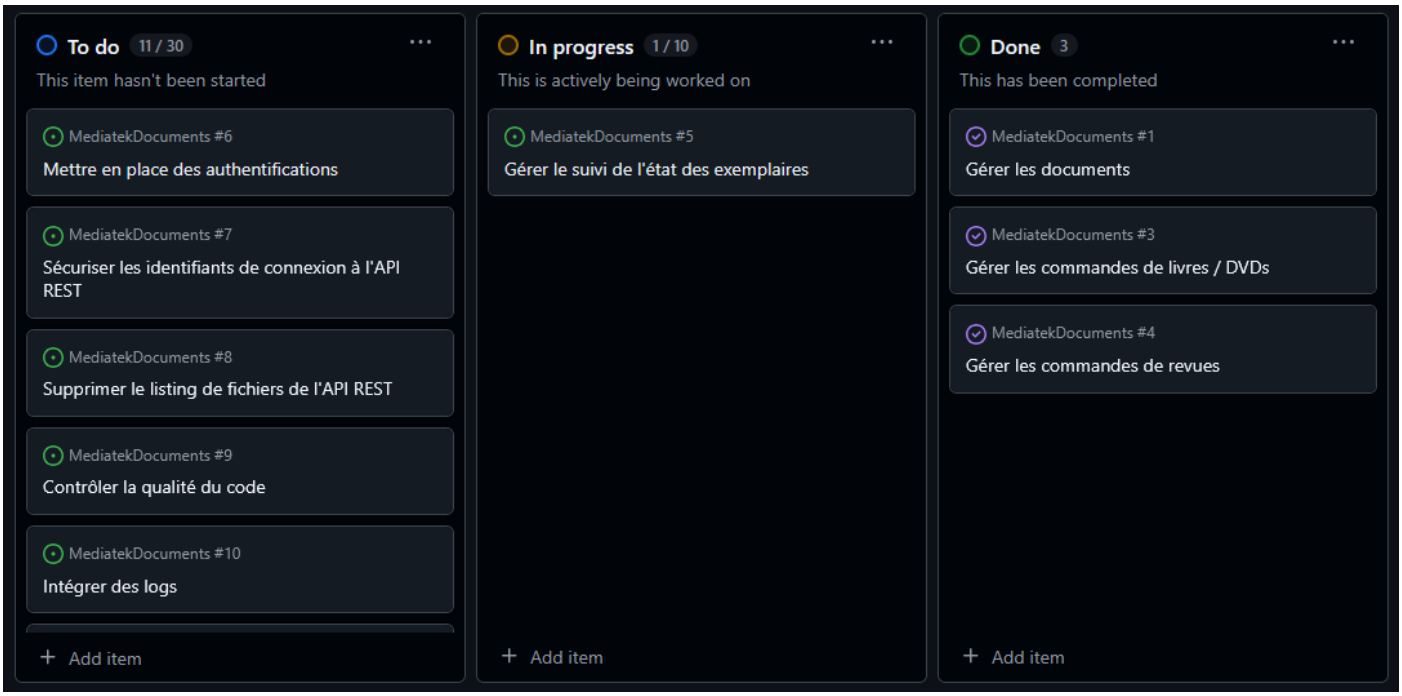
Enfin, au démarrage de l'application, on affiche les revues dont l'abonnement arrive à expiration dans moins de 30 jours :



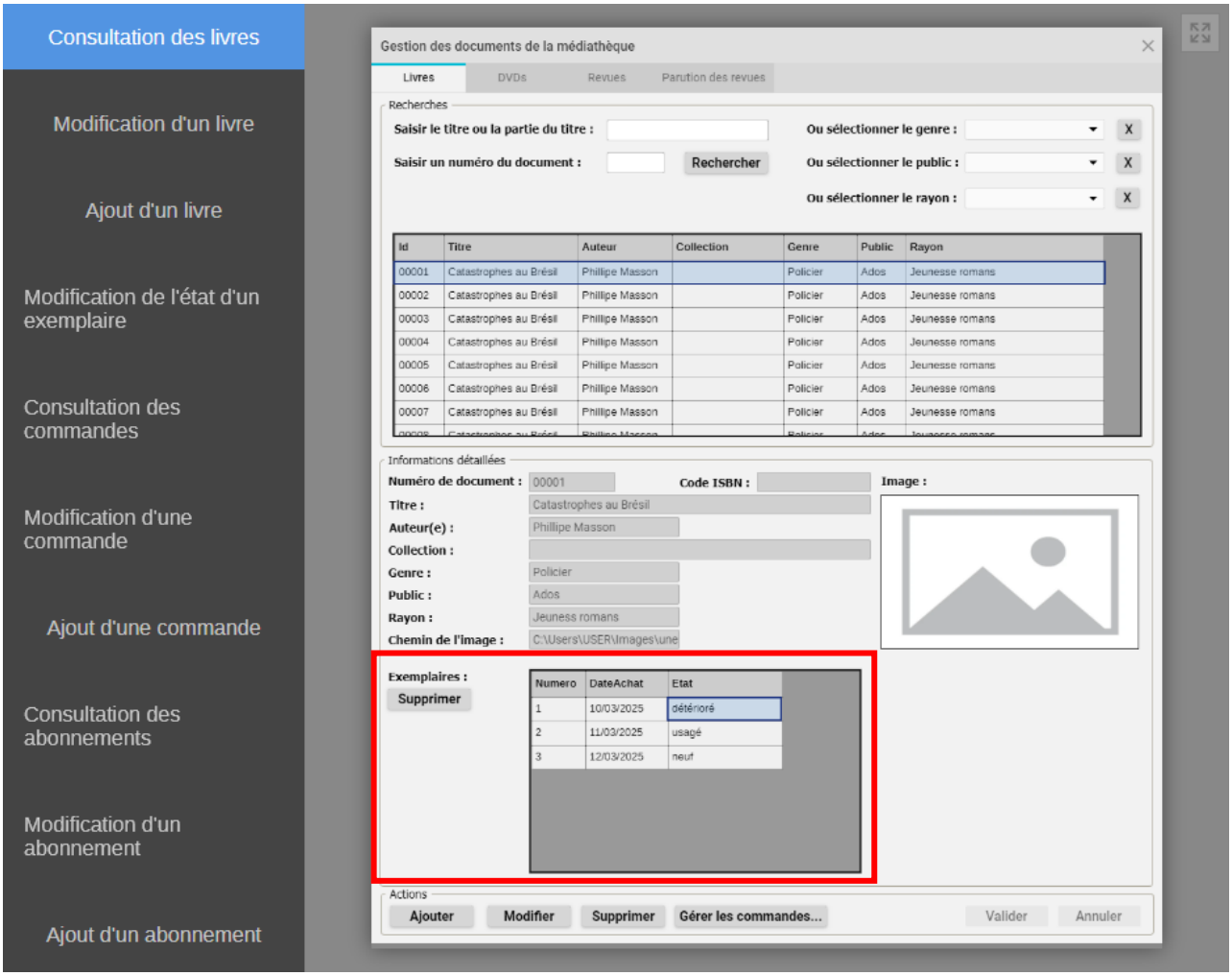
Les maquettes et diagrammes concernant cette tâche sont disponibles sur la page dédiée au projet sur mon portfolio.

Étape 4 : Gestion du suivi de l'état des exemplaires

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

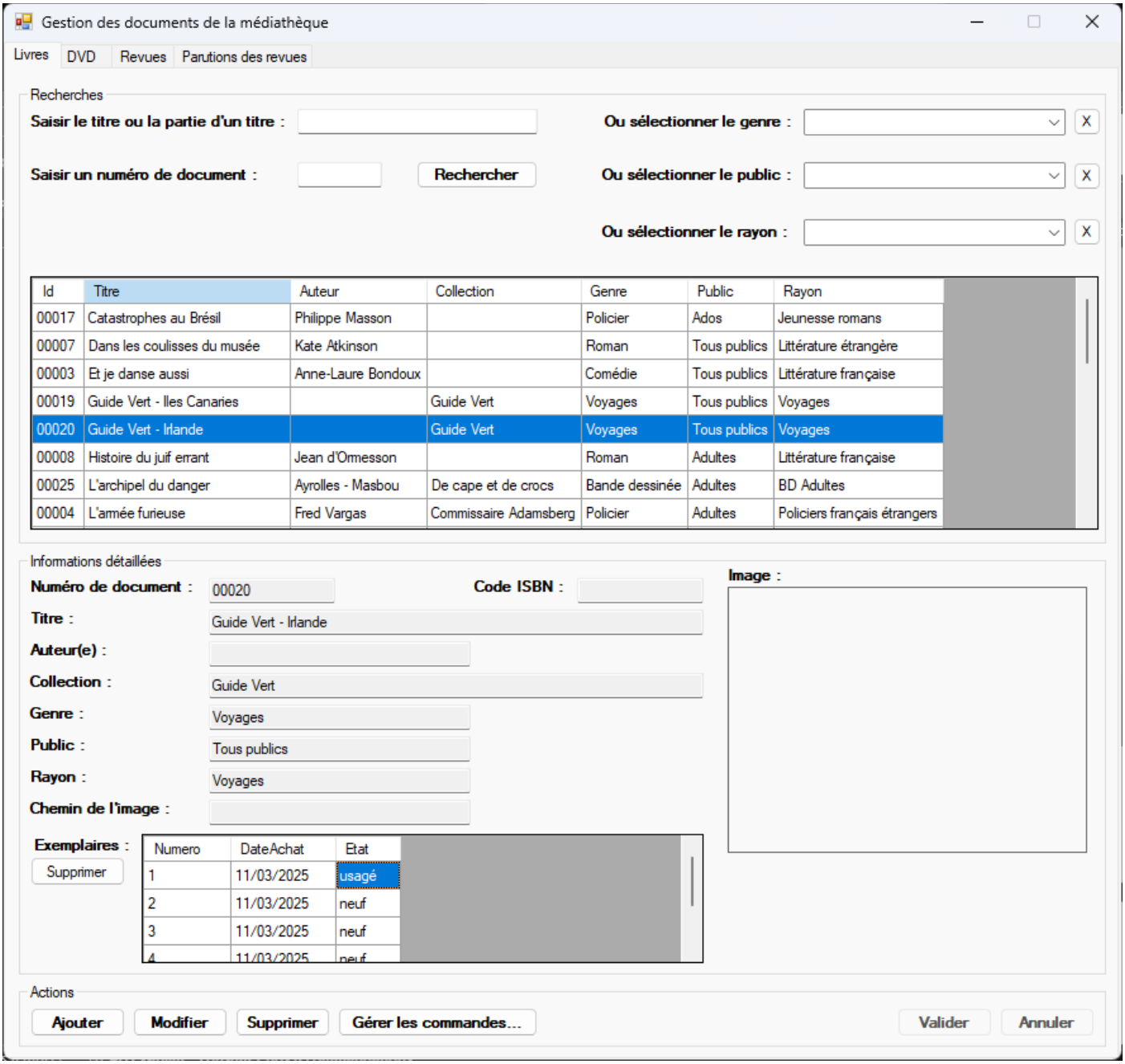


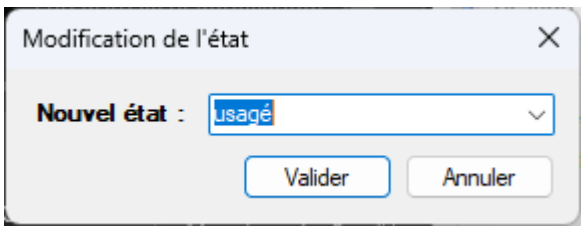
Afin de pouvoir gérer l'état de suivi des exemplaires, nous devons d'abord afficher les exemplaires dans la fenêtre, voici une maquette pour voir les modifications que l'on va effectuer :



On ajoute une section montrant les exemplaires d'un livre qui sera mise à jour à chaque sélection d'un livre dans la liste. On a aussi un bouton pour supprimer un exemplaire et enfin, on pourra changer l'état d'un exemplaire en double cliquant sur son état.

Au niveau de l'application, on créé un mini-formulaire pour pouvoir modifier l'état d'un exemplaire que l'on va appeler FrmModifEtat. Le formulaire apparaitra au double clic sur l'état d'un exemplaire et va sélectionner l'état de l'exemplaire a modifier par défaut :



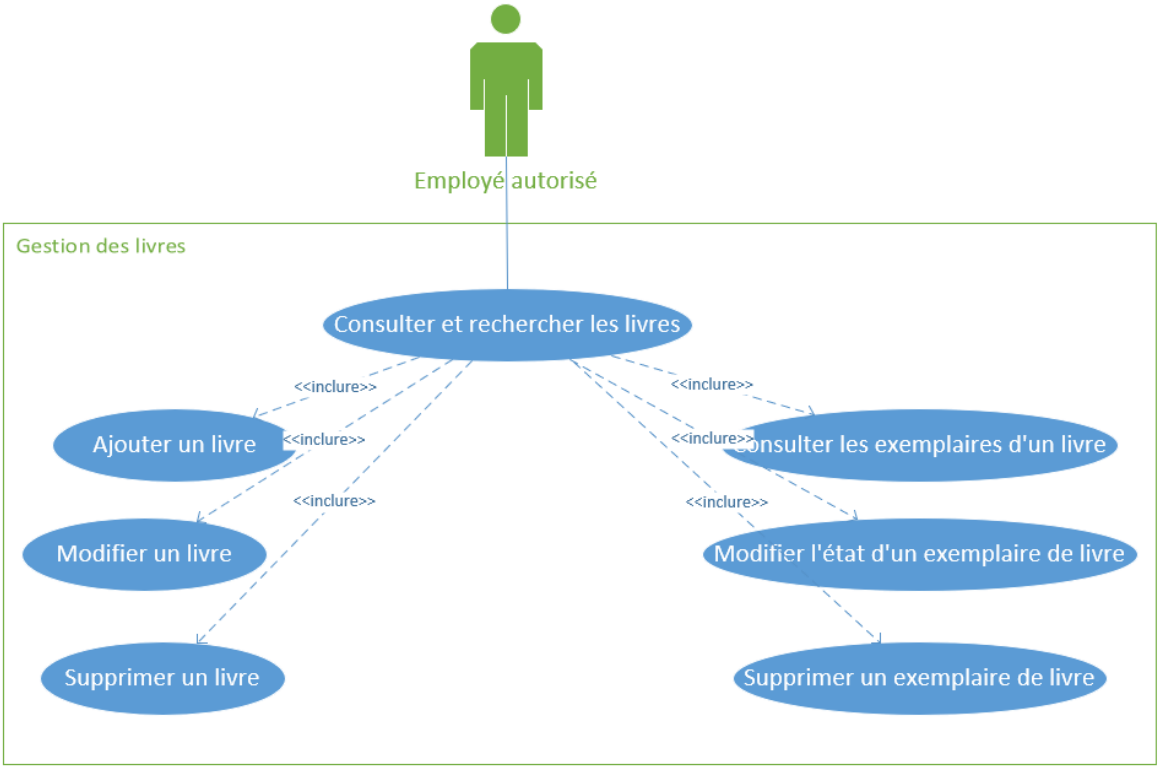


L'API REST a aussi été modifiée pour prendre en charge en charge les mises à jour d'exemplaires et aussi pour ajouter le champ de l'état d'un exemplaire en plus des autres champs déjà retournés lors de la sélection d'exemplaires.

Afin de pouvoir créer un exemplaire dans la base de données sans avoir a créer de méthode spécialisée, une méthode ToRestApiObject() sur un exemplaire a été créée. Cette méthode retourne un objet compatible avec les champs de la table exemplaire afin que l'API puisse l'ajouter à partir de tout ses champs depuis insertOneTupleOneTable sans méthode spécialisée :

```
/// <summary>
/// Méthode retournant un objet compatible avec les méthodes d'insertion et modification de l'API REST
/// </summary>
/// <returns>L'objet Exemplaire compatible avec les méthodes d'insertion et modification de l'API REST</returns>
public object ToRestApiObject()
{
    return new
    {
        Numero,
        DateAchat,
        Photo,
        IdEtat,
        Id
    };
}
```

Voici le diagramme de cas d'utilisation correspondant à cette tâche :

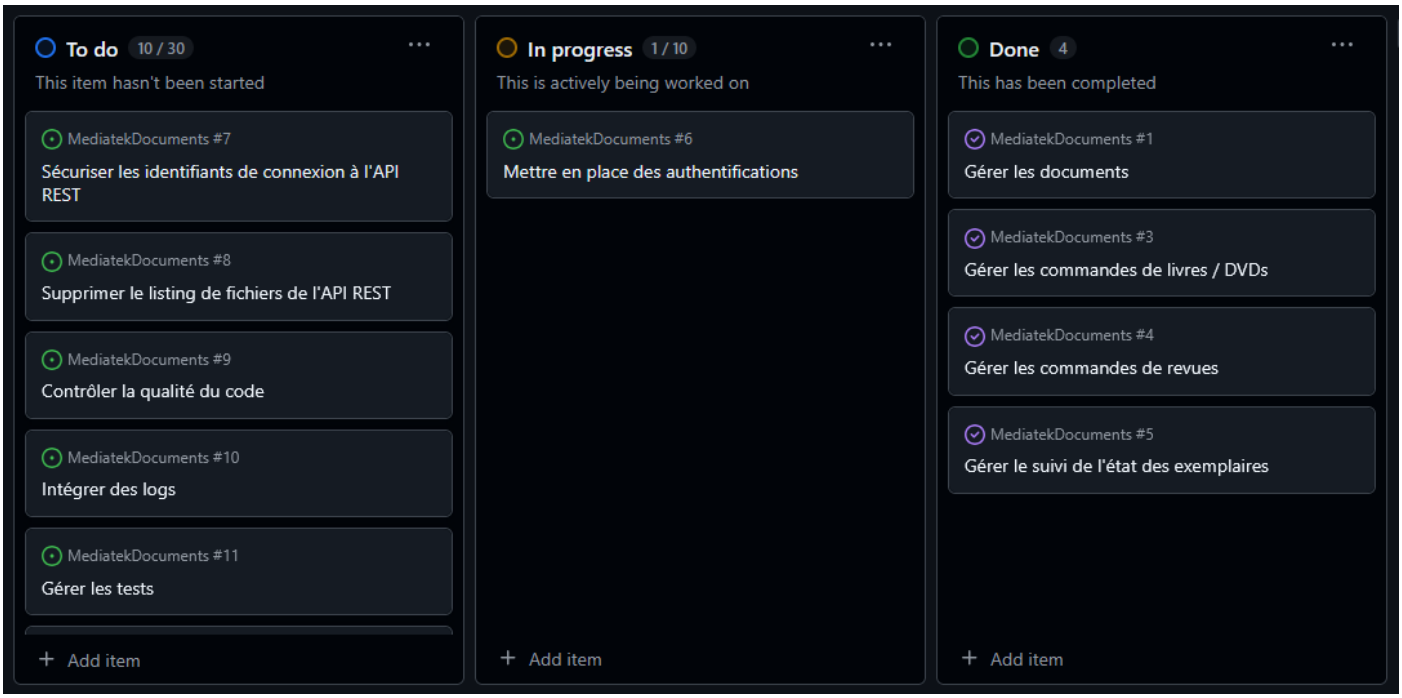


Pour les parutions de revues, la même logique a été suivie : la liste de parutions a été réutilisée en changeant la colonne "photo" à "état" et en ajoutant un bouton supprimer.

La maquette et diagramme UML de cas d'utilisation est disponible sur la page dédiée au projet sur mon portfolio

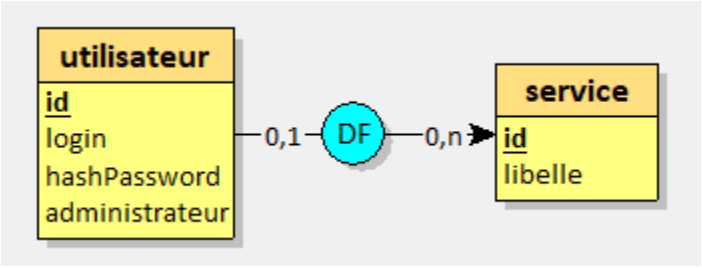
Étape 5 : Mise en place des authentifications

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



Pour commencer, nous allons ajouter la table utilisateur et la table service. Un utilisateur peut avoir 2 états différents : soit il est administrateur et ne possède pas de service, soit il n'est pas administrateur et est rattaché à un service ("Administratif", "Prets", "Culturel").

Voici ce que cela donne au niveau du MCD :



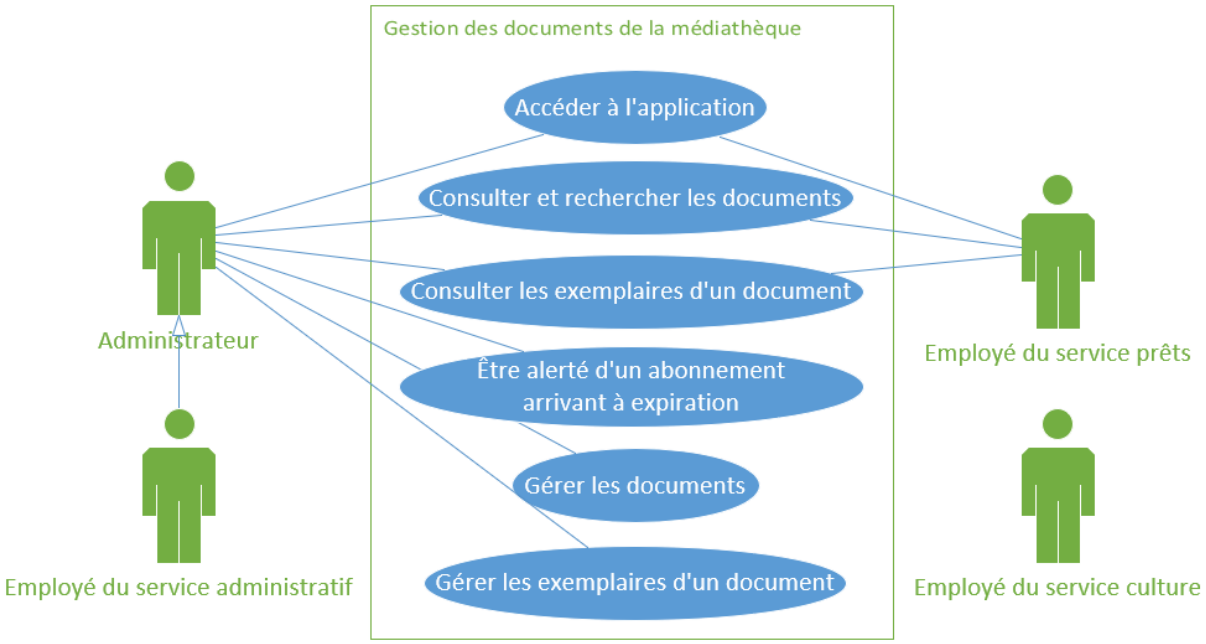
On va maintenant insérer des utilisateurs et des services après avoir créé la table :

INSERT INTO service(libelle) VALUES ("Administratif"), ("Prets"), ("Culture");
INSERT INTO utilisateur(login, hashPassword, administrateur, idService)
VALUES
("admin", SHA2("<mot de passe>", 256), 1, null),
("jfrancart", SHA2("<mot de passe>", 256), 0, 1),
("adumont", SHA2("<mot de passe>", 256), 0, 2),
("pmenier", SHA2("<mot de passe>", 256), 0, 3);

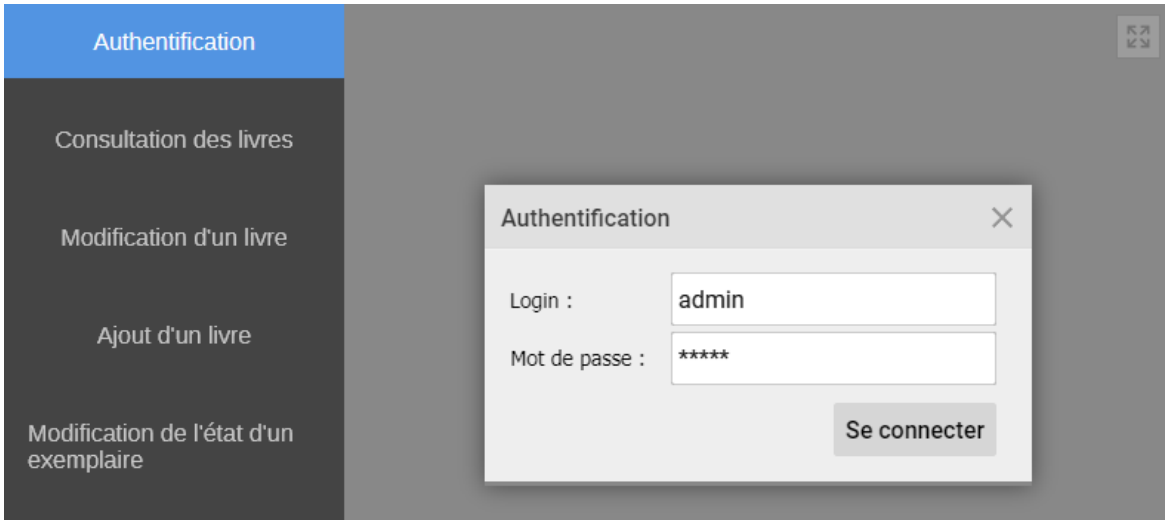
Les 2 états différent d'un utilisateur va permettre de contrôler ses habilitations à l'application :

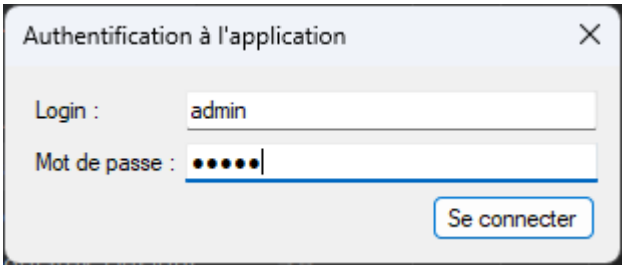
- Un administrateur à accès a toutes les fonctionnalités de l'application
- Un utilisateur du service administratif à aussi accès a toutes les fonctionnalités de l'application mais la séparation peut être utile au cas ou d'autres restrictions doivent être gérées dans le futur
- Un utilisateur du service prêts à accès à l'application en lecture seule
- Un utilisateur du service culturel n'est pas habilité à utiliser l'application, il n'y a donc pas accès

Voici le diagramme de cas d'utilisation correspondant :



Maintenant, nous allons créer une fenêtre d'authentification qui s'exécutera au lancement de l'application :





Une fois le clic sur le bouton se connecter, le formulaire va contrôler que l'authentification est correcte puis va construire un objet de type FrmMediatek en lui passant l'utilisateur qui vient de s'authentifier. Si l'utilisateur n'est pas habilité à utiliser l'application, elle se fermera.

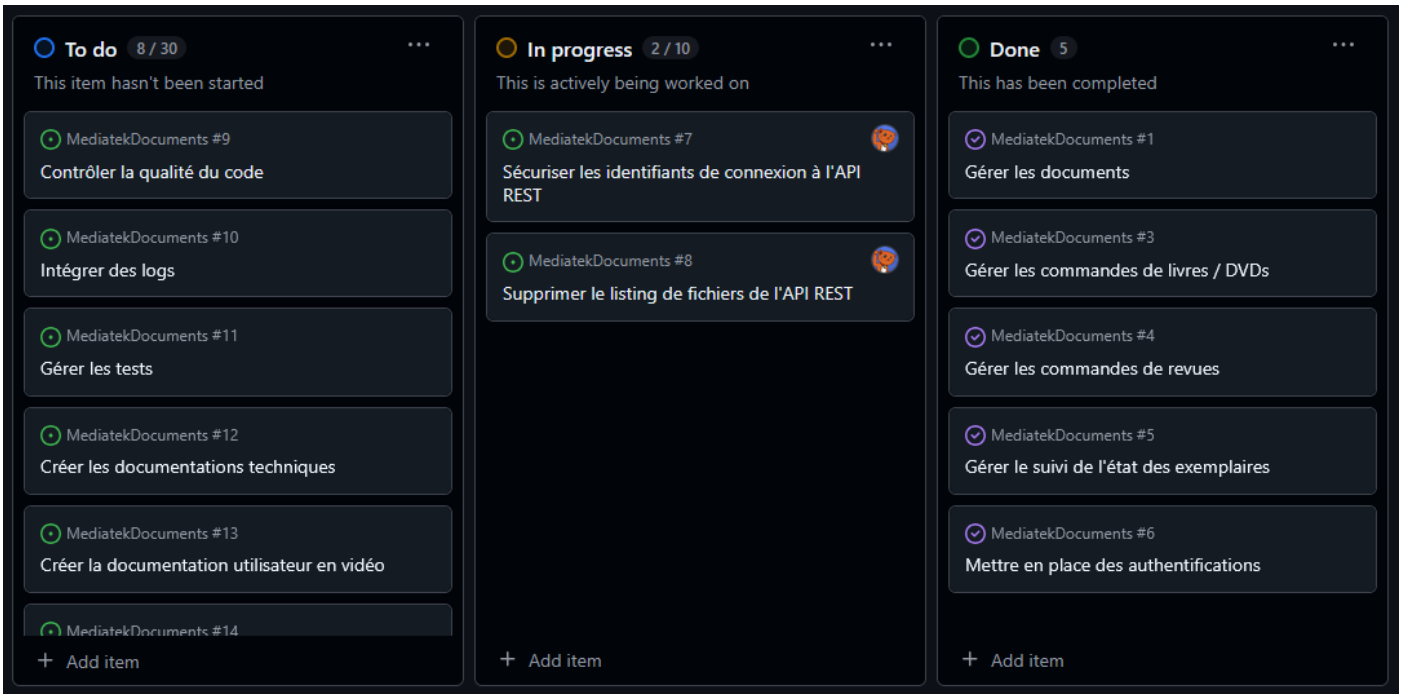
Enfin, au niveau du formulaire FrmMediatek, on désactive des fonctionnalités selon l'utilisateur de l'application :

```
/// <summary>
/// Constructeur : création du contrôleur lié à ce formulaire et activation / désactivation
/// des fonctionnalités en fonction de l'utilisateur de l'application
/// </summary>
internal FrmMediatek(Utilisateur utilisateur)
{
    InitializeComponent();
    this.controller = new FrmMediatekController();
    this.utilisateur = utilisateur;

    if (!utilisateur.Administrateur && utilisateur.Service != "Administratif")
    {
        if (utilisateur.Service == "Prets")
        {
            grpLivresActions.Enabled = false;
            btnLivresExemplairesSupprimer.Visible = false;
            dgvLivresExemplaires.CellMouseDoubleClick -= dgvLivresExemplaires_CellMouseDoubleClick;

            grpDvdActions.Enabled = false;
            btnDvdExemplairesSupprimer.Visible = false;
            dgvDvdExemplaires.CellMouseDoubleClick -= dgvDvdExemplaires_CellMouseDoubleClick;

            grpRevuesActions.Enabled = false;

            grpReceptionExemplaire.Enabled = false;
            btnReceptionParutionsSupprimer.Visible = false;
            dgvReceptionExemplairesListe.CellMouseDoubleClick -= dgvReceptionExemplairesListe_CellMouseDoubleClick;
        }

        return;
    }

    List<RevueAbonnementAExpiration> revuesExpirationProchaine = controller.GetRevuesAbonnementAExpirationProchaine();

    if (revuesExpirationProchaine.Count != 0)
```

Étape 6 : Assurance de la sécurité, la qualité et intégration des logs

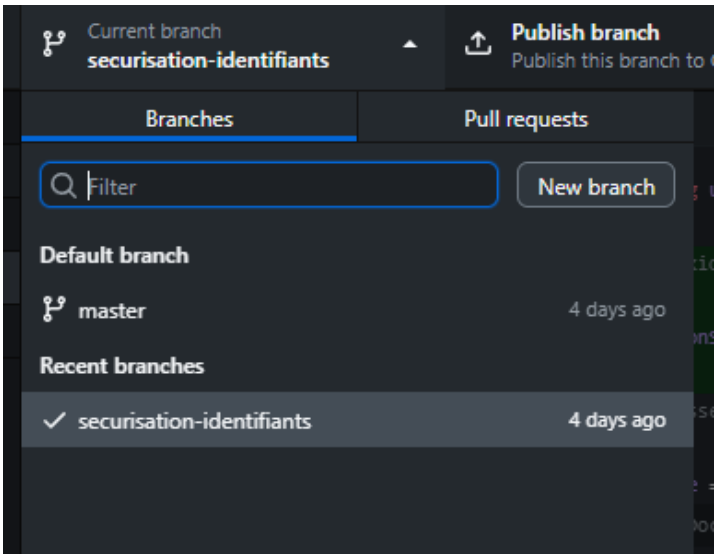
Tâche 1 : Corriger des problèmes de sécurité
Temps de réalisation estimé : 2h - Temps de réalisation réel : 2h

Pour cette tâche, des problèmes relevant de la sécurité de l'application et de l'API REST ont été identifiés et nous devons les corriger.



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.

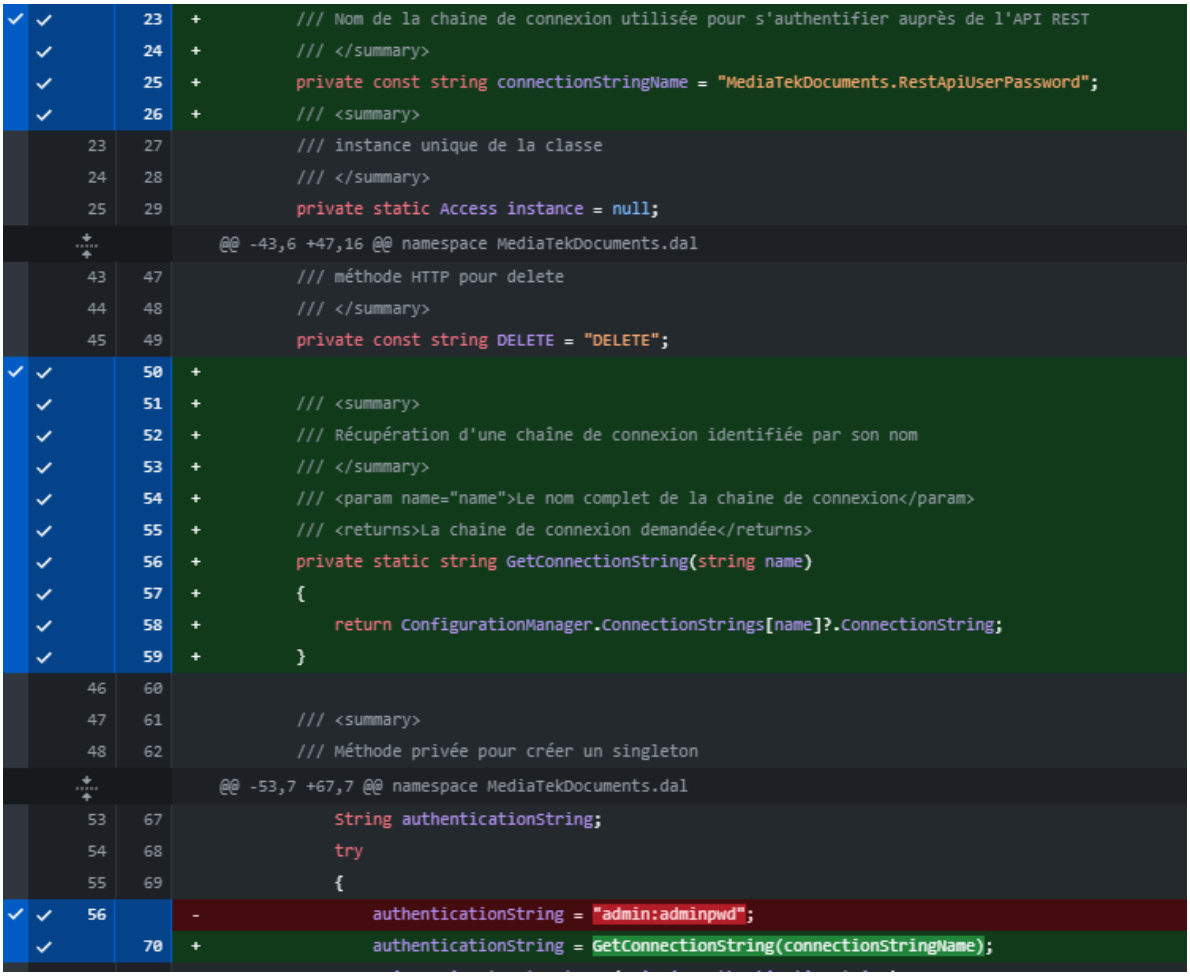
Pour résoudre ce problème, nous allons d'abord créer une branche séparée sur notre dépôt local pour travailler individuellement sur ce problème.



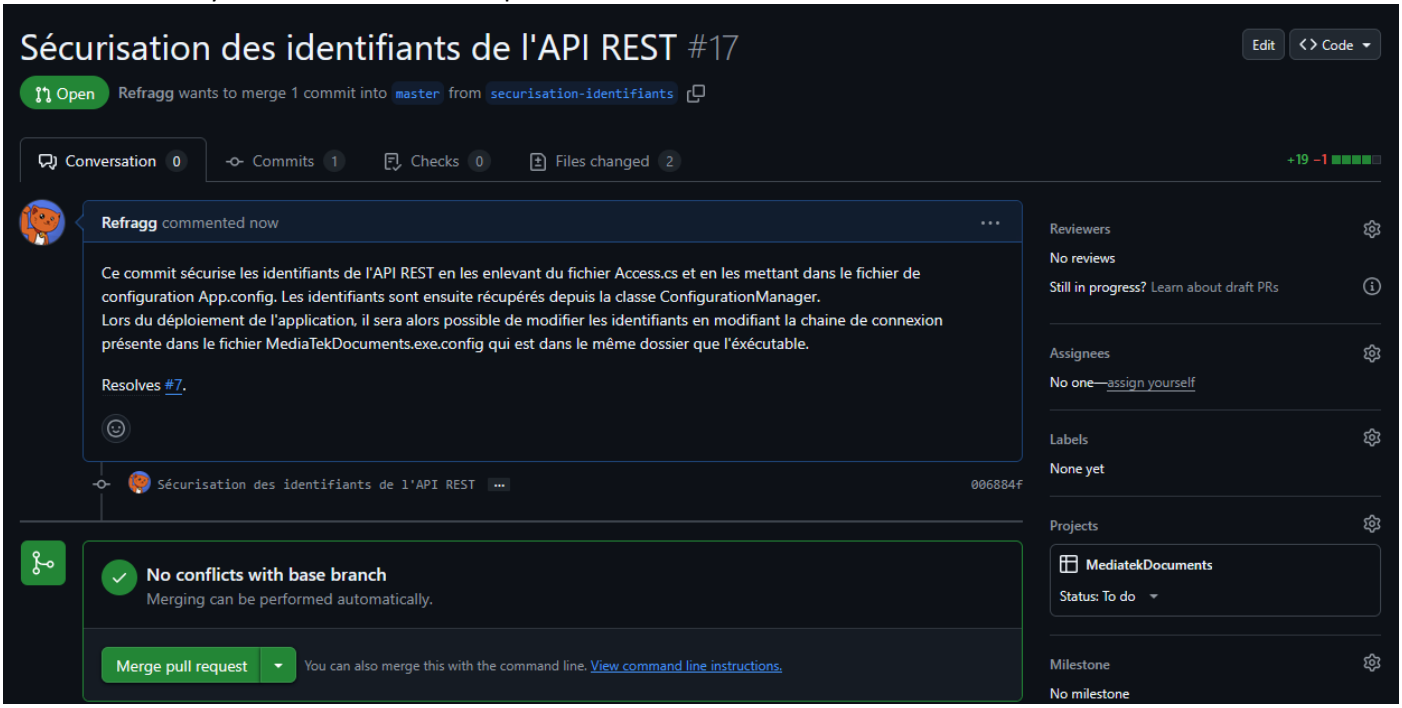
Nous allons transférer les identifiants d'accès à la base de données vers une chaine de connexion dans le fichier App.config. Ce fichier contiendra la valeur par défaut de admin:adminpwd pour l'authentification de l'API mais il sera possible de modifier cette valeur depuis le fichier MediaTekDocuments.exe.config qui est disponible dans le dossier où se trouve l'exécutable de l'application

```
3     <configSections>
4     </configSections>
5     <connectionStrings>
6         <add name="MediaTekDocuments.RestApiUserPassword"
7              connectionString="admin:adminpwd" />
8     </connectionStrings>
9     <startup>
10        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
11    </startup>
```

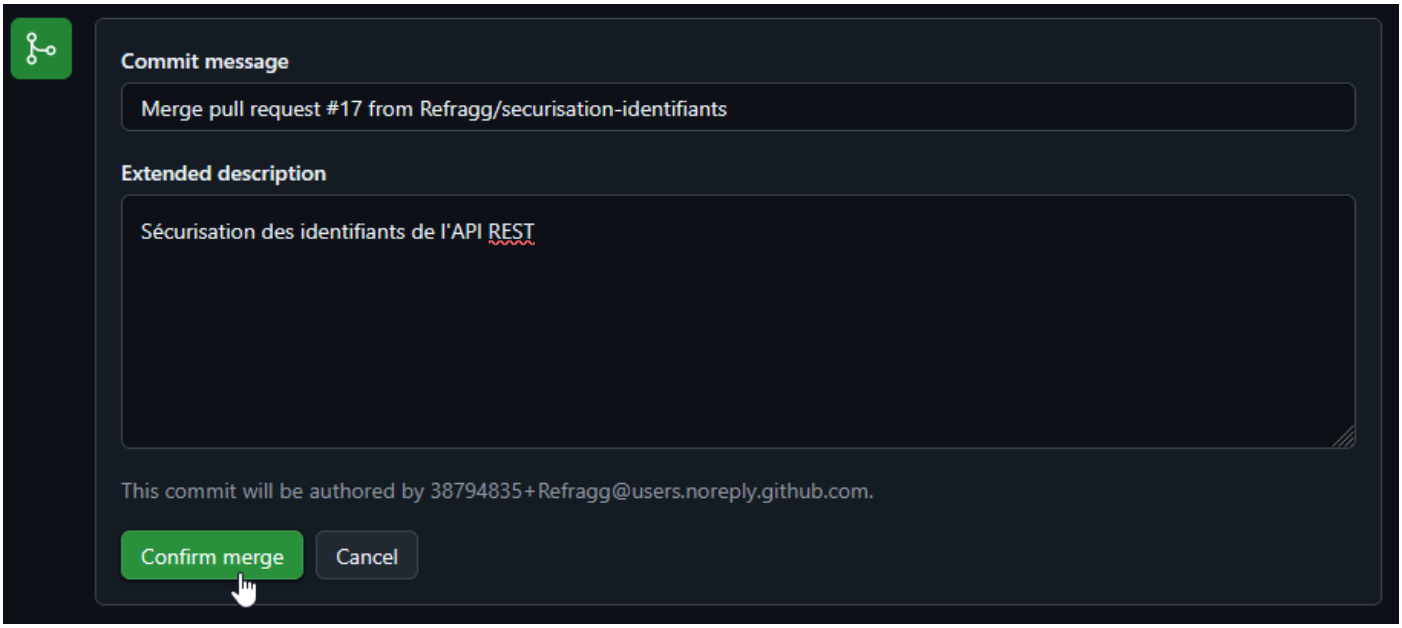
Nous pouvons maintenant récupérer la chaine de connexion grâce à la classe ConfigurationManager comme ceci :



Nous pouvons maintenant réaliser un commit sur la nouvelle branche et créer une pull request afin de demander le merge depuis la branche securisation-identifiants vers la branche master



Nous pouvons maintenant accepter les changements et effectuer le merge pull request, ce qui va merger les changements de la branche securisation-identifiants vers master.



Une fois le merge de la pull request fait, on peut supprimer la branche.

Le commentaire « Resolves #7 » de la pull request permet de clôturer l'issue automatiquement une fois que la pull request est merge.

Problème 2 : Si, pour accéder à l'API directement dans un navigateur, on donne juste l'addresse de l'API sans mettre de paramètres (par exemple : 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.

Pour résoudre ce problème, nous allons modifier les règles dans le fichier .htaccess de l'API REST afin de rediriger les requêtes qui n'ont pas encore été matchées vers une route qui renvoie une erreur. La

redirection est sur index.php avec le paramètre notfound=1 si index.php trouve ce paramètre, il demandera alors au contrôleur de retourner un message d'erreur :

.htaccess +10 -7

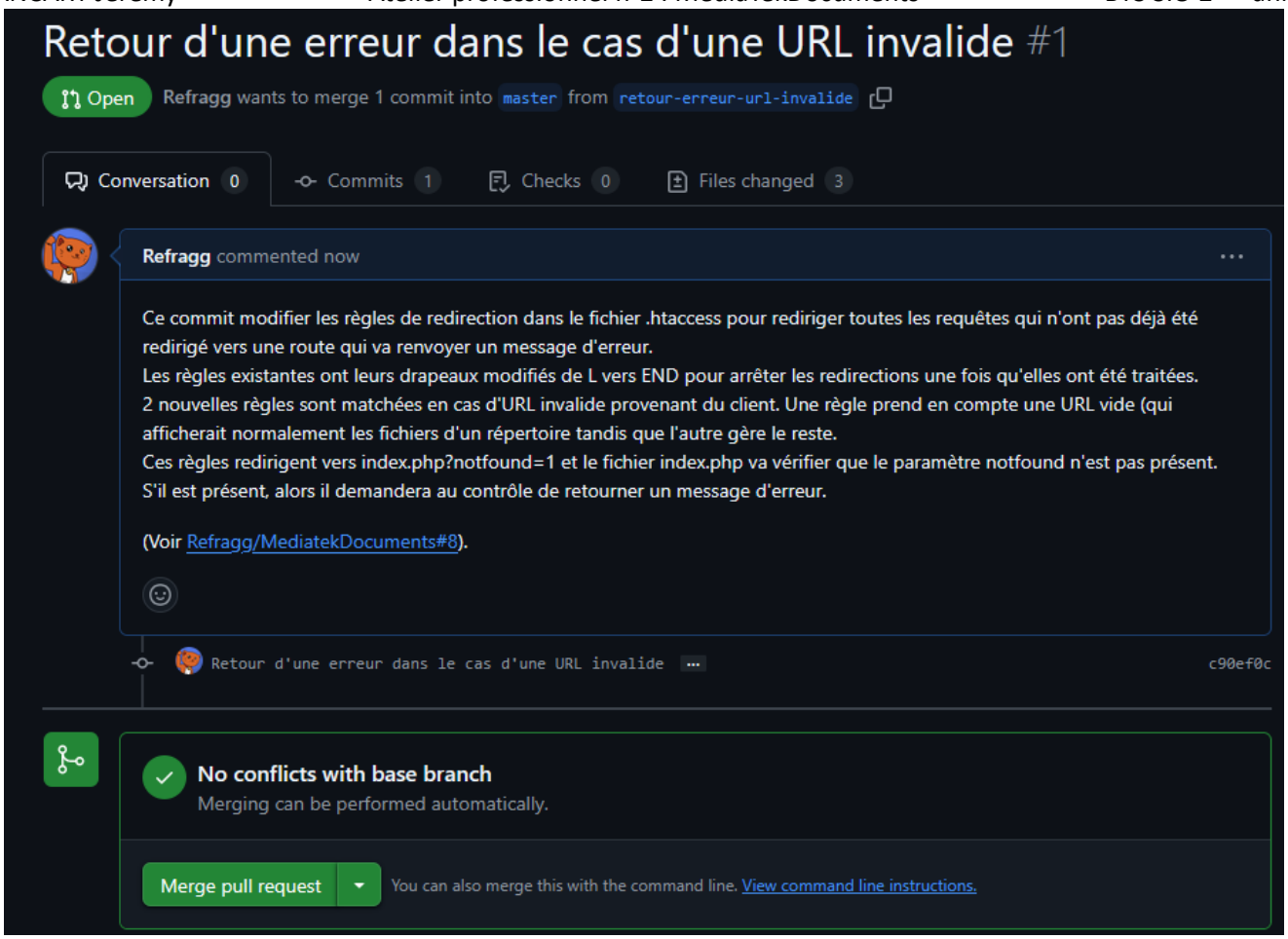
```
@@ -1,15 +1,18 @@
 RewriteEngine on
 RewriteCond %{REQUEST_METHOD} =GET
-RewriteRule ^([a-zA-Z0-9_]+)$ src/index.php?table=$1 [B,L]
+RewriteRule ^([a-zA-Z0-9_]+)$ src/index.php?table=$1 [B,END]
 RewriteCond %{REQUEST_METHOD} =GET
-RewriteRule ^([a-zA-Z0-9_]+)/({.*})$ src/index.php?table=$1&champs=$2 [B,L]
+RewriteRule ^([a-zA-Z0-9_]+)/({.*})$ src/index.php?table=$1&champs=$2 [B,END]
 RewriteCond %{REQUEST_METHOD} =POST
-RewriteRule ^([a-zA-Z0-9_]+)$ src/index.php?table=$1 [B,L]
+RewriteRule ^([a-zA-Z0-9_]+)$ src/index.php?table=$1 [B,END]
 RewriteCond %{REQUEST_METHOD} =PUT
-RewriteRule ^([a-zA-Z0-9_]+)$ src/index.php?table=$1 [B,L]
+RewriteRule ^([a-zA-Z0-9_]+)$ src/index.php?table=$1 [B,END]
 RewriteCond %{REQUEST_METHOD} =PUT
-RewriteRule ^([a-zA-Z0-9_]+)/([a-zA-Z0-9]+)$ src/index.php?table=$1&id=$2 [B,L]
+RewriteRule ^([a-zA-Z0-9_]+)/([a-zA-Z0-9]+)$ src/index.php?table=$1&id=$2 [B,END]
 RewriteCond %{REQUEST_METHOD} =DELETE
-RewriteRule ^([a-zA-Z0-9_]+)$ src/index.php?table=$1 [B,L]
+RewriteRule ^([a-zA-Z0-9_]+)$ src/index.php?table=$1 [B,END]
 RewriteCond %{REQUEST_METHOD} =DELETE
-RewriteRule ^([a-zA-Z0-9_]+)/({.*})$ src/index.php?table=$1&champs=$2 [B,L]
+RewriteRule ^([a-zA-Z0-9_]+)/({.*})$ src/index.php?table=$1&champs=$2 [B,END]
+
+RewriteRule ^$ src/index.php?notfound=1 [B,END]
+RewriteRule . src/index.php?notfound=1 [B,END]
```

src/Controle.php +4

```
@@ -39,6 +39,10 @@ public function demande(string $methodeHTTP, string $table, ?string $id, ?array
         $this->controleResult($result);
     }

+    public function reponseNotFound() {
+        $this->reponse(404, "ce point d'entrée de l'API n'existe pas");
+    }
+
     /**
      * réponse renvoyée (affichée) au client au format json
      * @param int $code code standard HTTP (200, 500, ...)
```

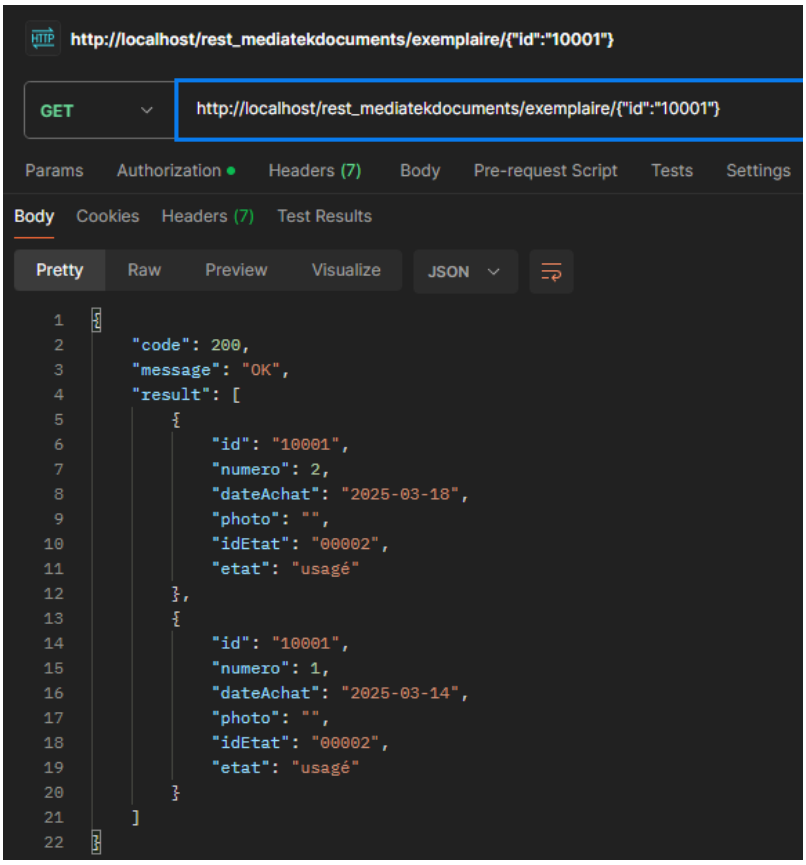
src/index.php +5

```
@@ -14,6 +14,11 @@
 // crée l'objet d'accès au contrôleur
 $controle = new Controle();

+if (isset($_REQUEST['notfound']) && $_REQUEST['notfound'] == 1) {
+    $controle->reponseNotFound();
+    return;
+}
+
 // vérifie l'authentification
 if (!$url->authentification()){
     // l'authentification a échoué
```

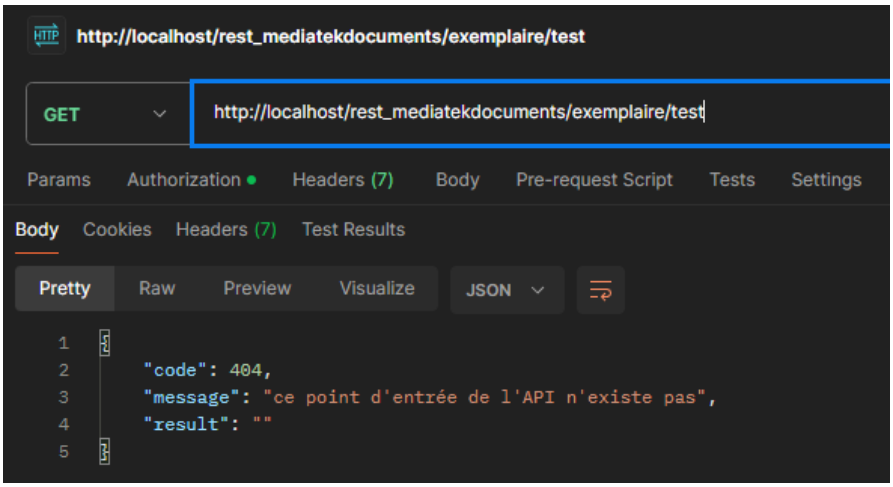


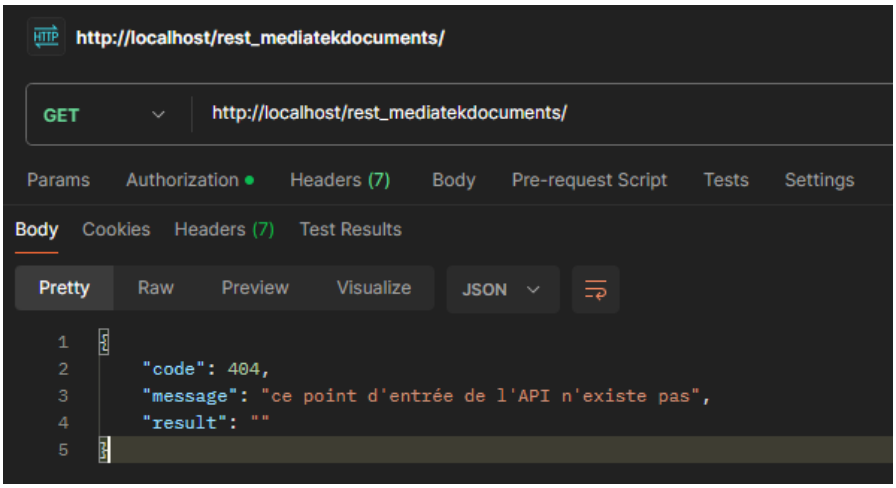
Voici le résultat :

Requête classique marchant toujours :



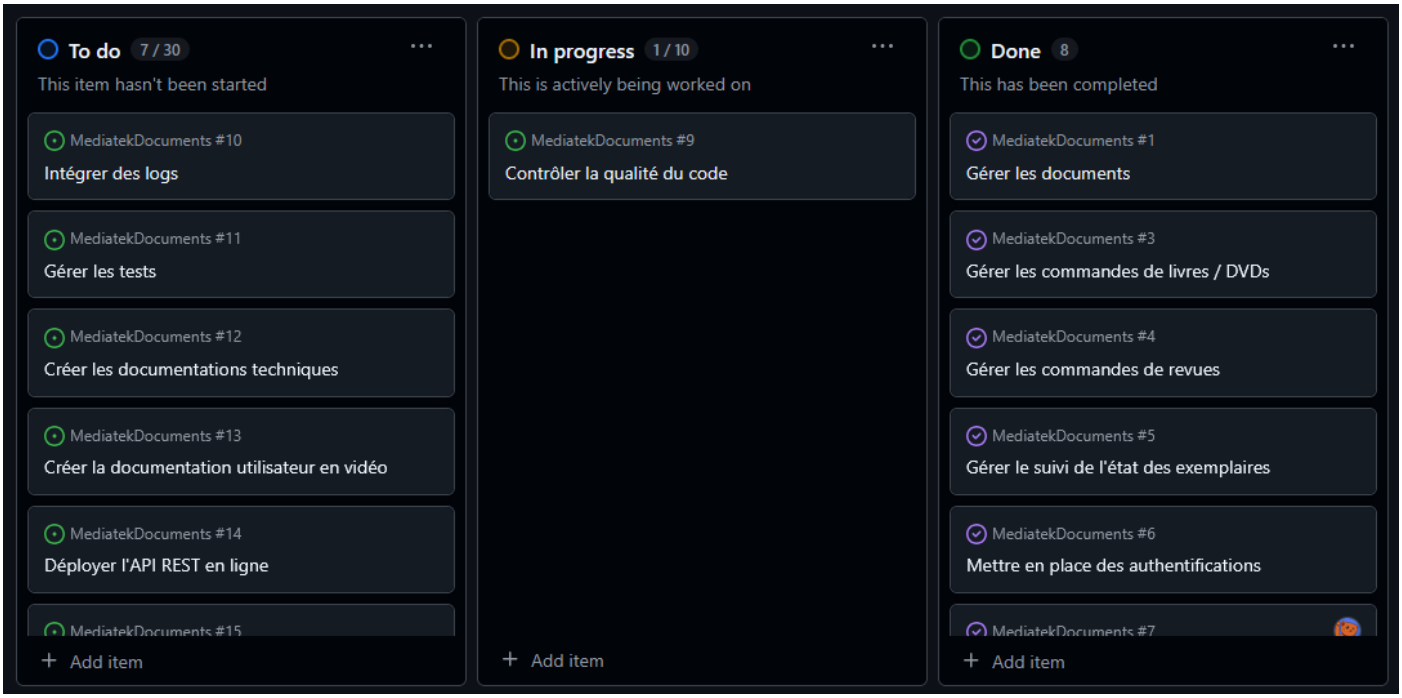
Requêtes avec URLs invalides ne fonctionnant plus :





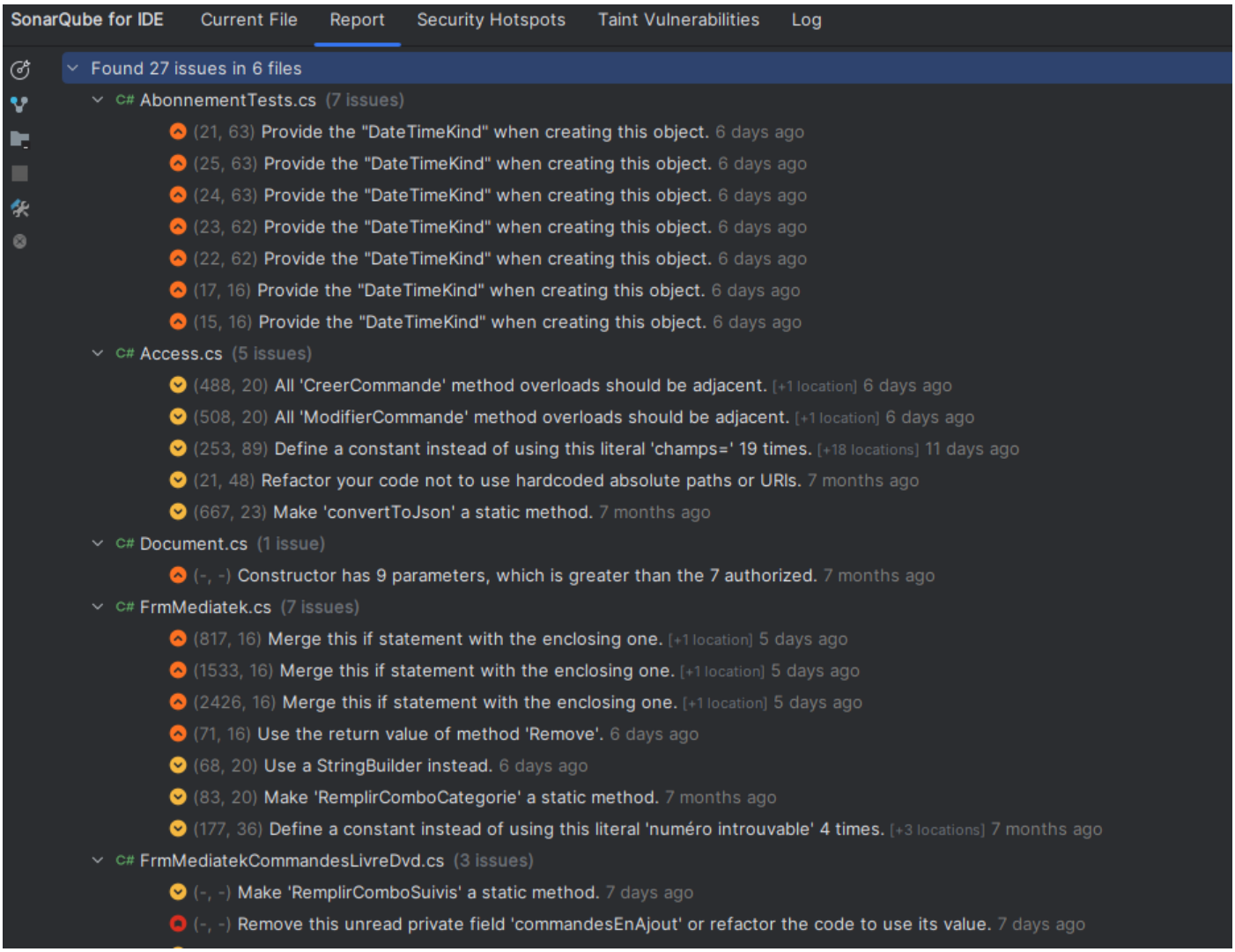
Tâche 2 : Contrôler la qualité

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

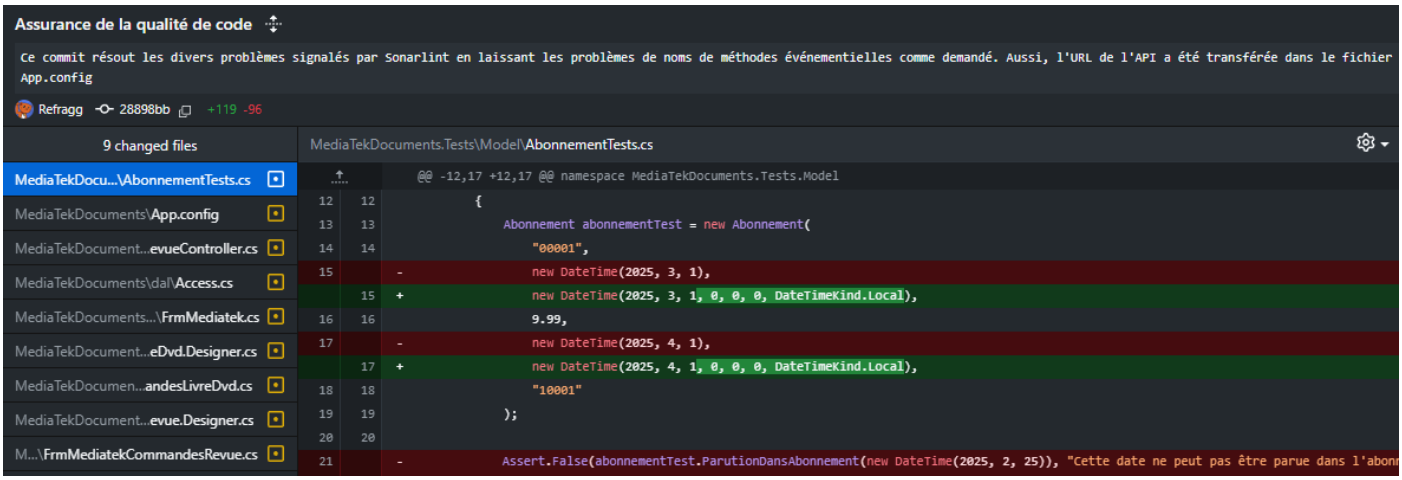


En utilisant l'outil de gestion de qualité de code Sonarlint, il nous est demandé de régler les problèmes rapportés quand cela est approprié.

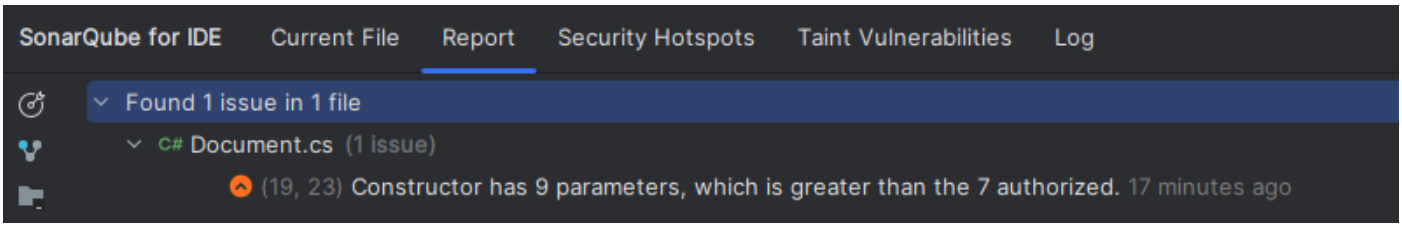
Voici les problèmes rapportés par Sonarlint :



Les problèmes rapportés sont des problèmes que l'on peut régler rapidement, voici la liste des fichiers modifiés ainsi que le commit pour cette tâche (nous avons aussi au passage déplacé l'URL d'accès à l'API dans le fichier App.config similairement aux identifiants de la tâche précédente) :



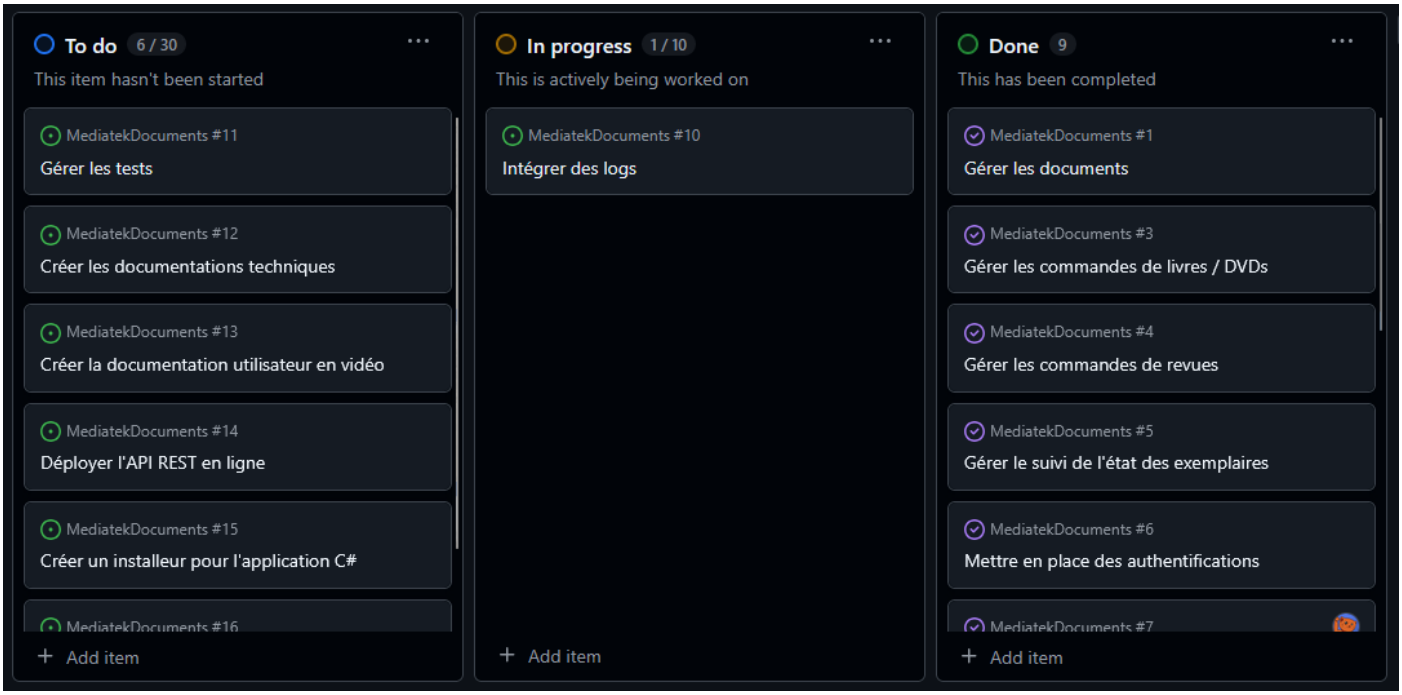
Liste des problèmes rapportés par Sonarlint après le commit :



Nous n'avons pas réglé ce dernier problème car la désérialisation JSON a besoin de tous ces paramètres dans le constructeur pour créer l'objet Document.

Tâche 3 : Intégrer des logs

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



Dans cette tâche, il nous est demandé de remplacer les appels simples à Console.WriteLine() par un système de journalisation centralisé qui permet une meilleure gestion des journaux (logs).

Pour ce faire, nous allons utiliser le package NuGet Serilog.

On configure le Logger du package Serilog depuis la classe principale Program afin de l'initialiser dès le début :

```
13          [STAThread]
            CNED SLAM +1 *
14          static void Main()
15          {
16              Log.Logger = new LoggerConfiguration()
17                  .MinimumLevel.Information()
18                  .WriteTo.Console()
19                  .WriteTo.File(path: "logs/MediaTekDocuments.log", rollingInterval: RollingInterval.Day) // LoggerConfiguration
20                  .CreateLogger(); // Logger
21
22              Application.EnableVisualStyles();
23              Application.SetCompatibleTextRenderingDefault(false);
24              Application.Run(new FrmAuth());
25          }
26      }
27  }
```

Cette configuration permet d'afficher seulement les logs à partir du niveau information, d'écrire les logs dans la console et de les écrire aussi dans un fichier dans le dossier logs. Un nouveau fichier de log sera créé chaque jour.

Ensuite, nous pouvons utiliser le logger configuré pour remplacer les appels à Console.WriteLine dans la classe Access :

```
257     public bool CreerExemplaire(Exemplaire exemplaire)
258     {
259         String jsonExemplaire = JsonConvert.SerializeObject(exemplaire.ToRestApiObject(), new CustomDateTim
260         try
261         {
262             List<Exemplaire> liste = TraitementRecup<Exemplaire>( methode: POST,  message: "exemplaire",  paramet
263             return (liste != null);
264         }
265         catch (Exception ex)
266         {
267             Log.Error(ex,  messageTemplate: "Une erreur est survenue lors de la création d'un exemplaire");
268         }
269         return false;
270     }
```

On fait de même avec tous les autres affichages console puis on obtient les logs et dans la console, et dans un fichier :

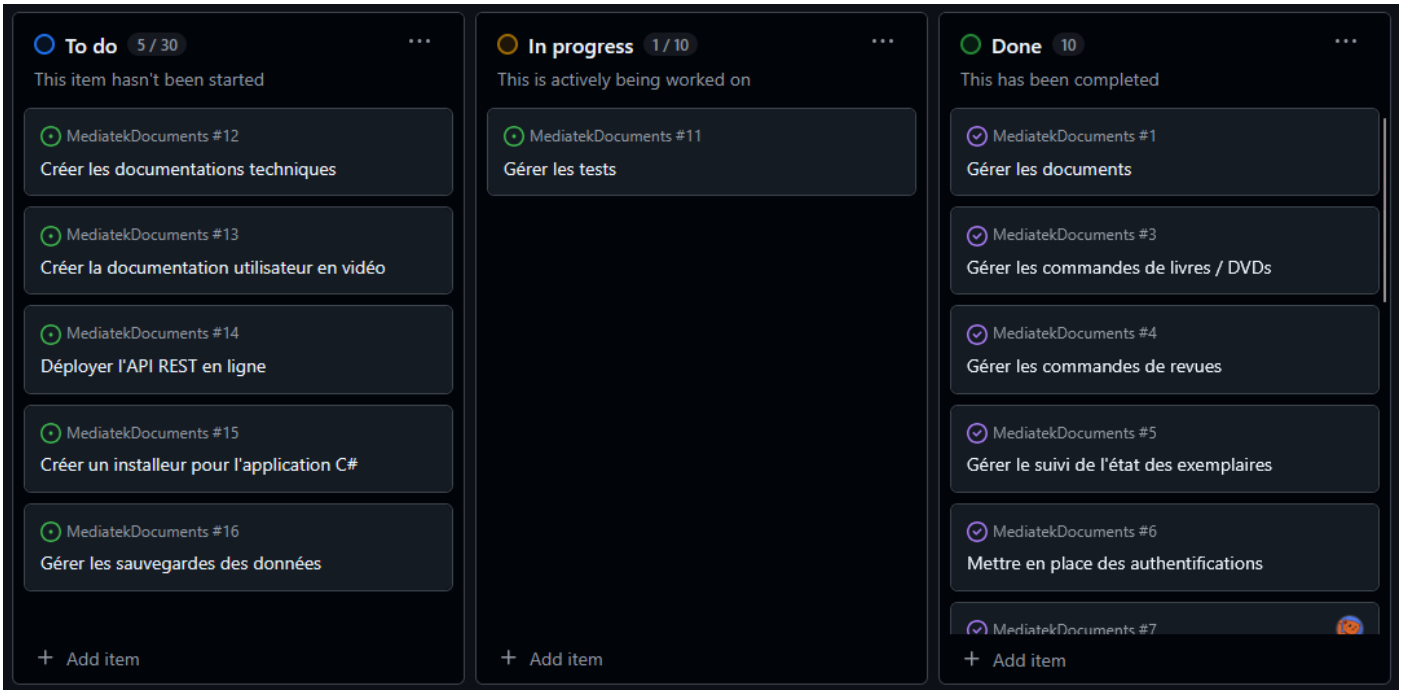
```
Run   MediaTekDocuments
D:/CNED/src/csharp/MediaTekDocuments/MediaTekDocuments/bin/Debug/MediaTekDocuments.exe
[14:26:57 ERR] L'API REST a retourn€ une erreur : code : '401', message : 'authentification incorrecte'
```

```
MediaTekDocuments20250319.log   App.config
Fichier   Modifier   Affichage
2025-03-19 14:26:57.629 +01:00 [ERR] L'API REST a retourné une erreur : code : '401', message : 'authentification incorrecte'
```

Étape 7 : Mise en place de tests et de la documentation

Tâche 1 : Gérer les tests

Temps de réalisation estimé : 5h - Temps de réalisation réel : 8h



Pour cette tâche, il nous est demandé de réaliser des tests unitaires sur les classes du dossier Model, d'effectuer des tests fonctionnels de l'interface utilisateur de l'application avec Specflow et enfin, de réaliser des tests fonctionnels sur les fonctionnalités de l'API REST.

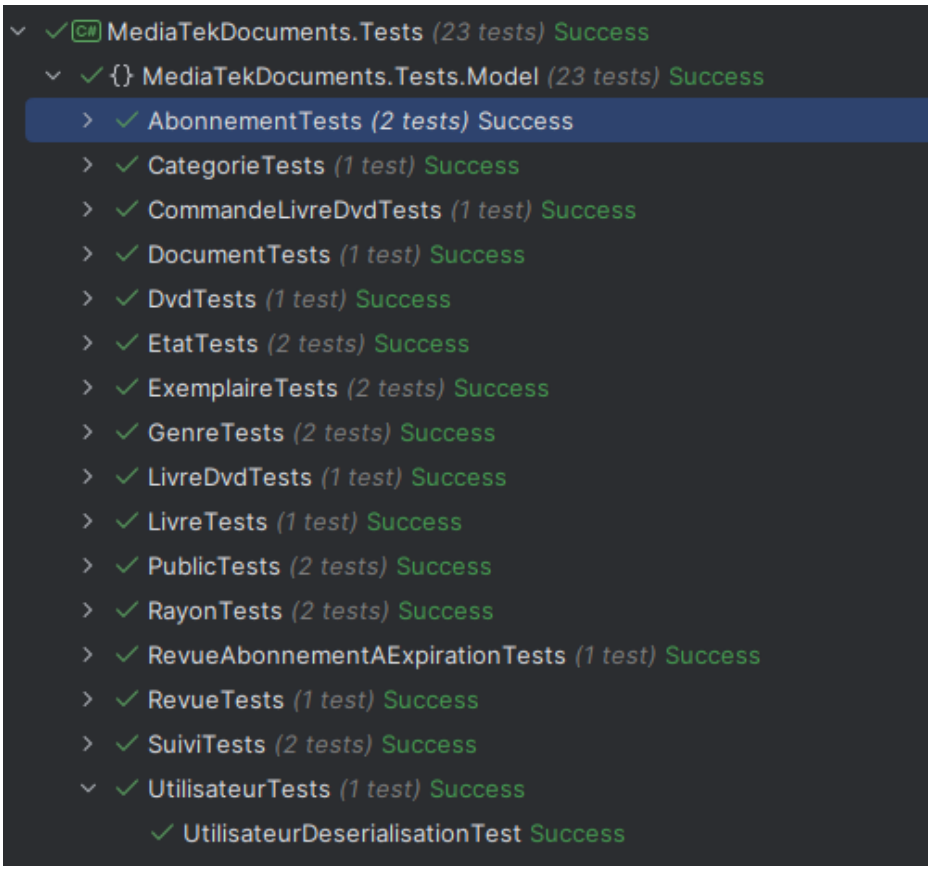
Commençons par les tests unitaires sur les classes du dossier Model.

De multiples tests ont été conçus afin de vérifier que la désérialisation des classes métiers se fait correctement, vérifier qu'une classe hérite bien d'une autre enfin, certains tests vérifient que la méthode surchargée ToString d'un type retourne le contenu attendu.

Exemple :

```
30      [Test]
        Refragg
31      public void AbonnementDeserialisationTest()
32      {
33          string json = "{\"id\":\"00051\",\"dateCommande\":\"2025-03-12\",\"montant\":50.99,\"dateFinAbonnement\":\"202
34          Abonnement abonnement = JsonConvert.DeserializeObject<Abonnement>(json);
35
36          Assert.IsNotNull(abonnement);
37          Assert.AreEqual("00051", abonnement.Id);
38          Assert.AreEqual(new DateTime(year: 2025, month: 03, day: 12, hour: 0, minute: 0, second: 0, DateTimeKind.Local), abonnem
39          Assert.AreEqual(50.99, abonnement.Montant);
40          Assert.AreEqual(new DateTime(year: 2025, month: 03, day: 20, hour: 0, minute: 0, second: 0, DateTimeKind.Local), abonnem
41          Assert.AreEqual("10001", abonnement.IdRevue);
42      }
```

Résultat des tests :



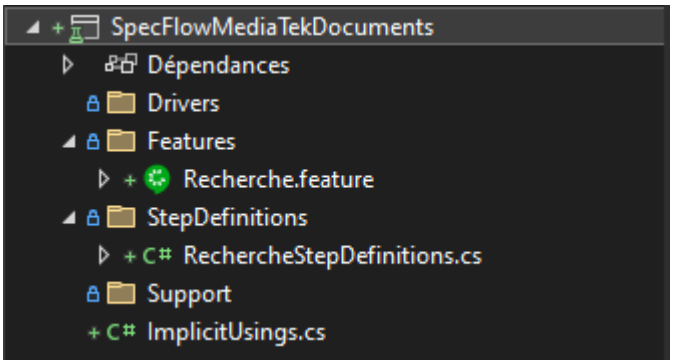
Ensuite, nous allons créer un test fonctionnel avec SpecFlow pour vérifier qu'une recherche dans l'onglet des livres retourne le bon résultat :

On commence par changer la visibilité du constructeur FrmMediatek en public afin de pouvoir y accéder depuis le projet SpecFlow.

Ensuite, on modifie la récupération des chaines de connexion pour toujours utiliser le fichier de configuration du projet MediaTekDocuments (au lieu d'utiliser la configuration du projet SpecFlow) :

```
2 références
private static string GetConnectionString(string name)
{
    string cheminExe = Assembly.GetExecutingAssembly().Location;
    Configuration config = ConfigurationManager.OpenExeConfiguration(cheminExe);
    return config.ConnectionStrings.ConnectionStrings[name]?.ConnectionString;
}
```

Création du projet SpecFlow :



Test décrit en langage humain :

```
Feature: Recherche

Scenario: Recherche livre
Given je suis dans longlet 'Livres'
When je selectionne le genre 'Bande dessinée'
Then le nombre de livres obtenu est de 5
```

Définition du code pour le test :

```
[Given(@"je suis dans longlet '([^']*)'")]
0 références
public void GivenJeSuisDansLonglet(string livres)
{
    TabControl tabOnglets = (TabControl)frmMediatek.Controls["tabOngletsApplication"];
    frmMediatek.Visible = true;
    tabOnglets.SelectedTab = GetTabPage("tabLivres");
    Assert.AreEqual("tabLivres", tabOnglets.SelectedTab.Name);
}

[When(@"je selectionne le genre '([^']*)'")]
0 références
public void WhenJeSelectionneLeGenre(string p0)
{
    ComboBox cbxLivresGenres = (ComboBox)GetTabPage("tabLivres").Controls["grpLivresRecherche"].Controls["cbxLivresGenres"];
    int indiceLigne = cbxLivresGenres.FindStringExact(p0);
    cbxLivresGenres.SelectedIndex = indiceLigne;
    Assert.AreNotEqual(-1, cbxLivresGenres.SelectedIndex);
}

[Then(@"le nombre de livres obtenu est de (.*)")]
0 références
public void ThenLeNombreDeLivresObtenuEstDe(int p0)
{
    DataGridView listeLivres = (DataGridView)GetTabPage("tabLivres").Controls["grpLivresRecherche"].Controls["dgvLivresListe"];
    Assert.AreEqual(5, listeLivres.Rows.Count);
}
```

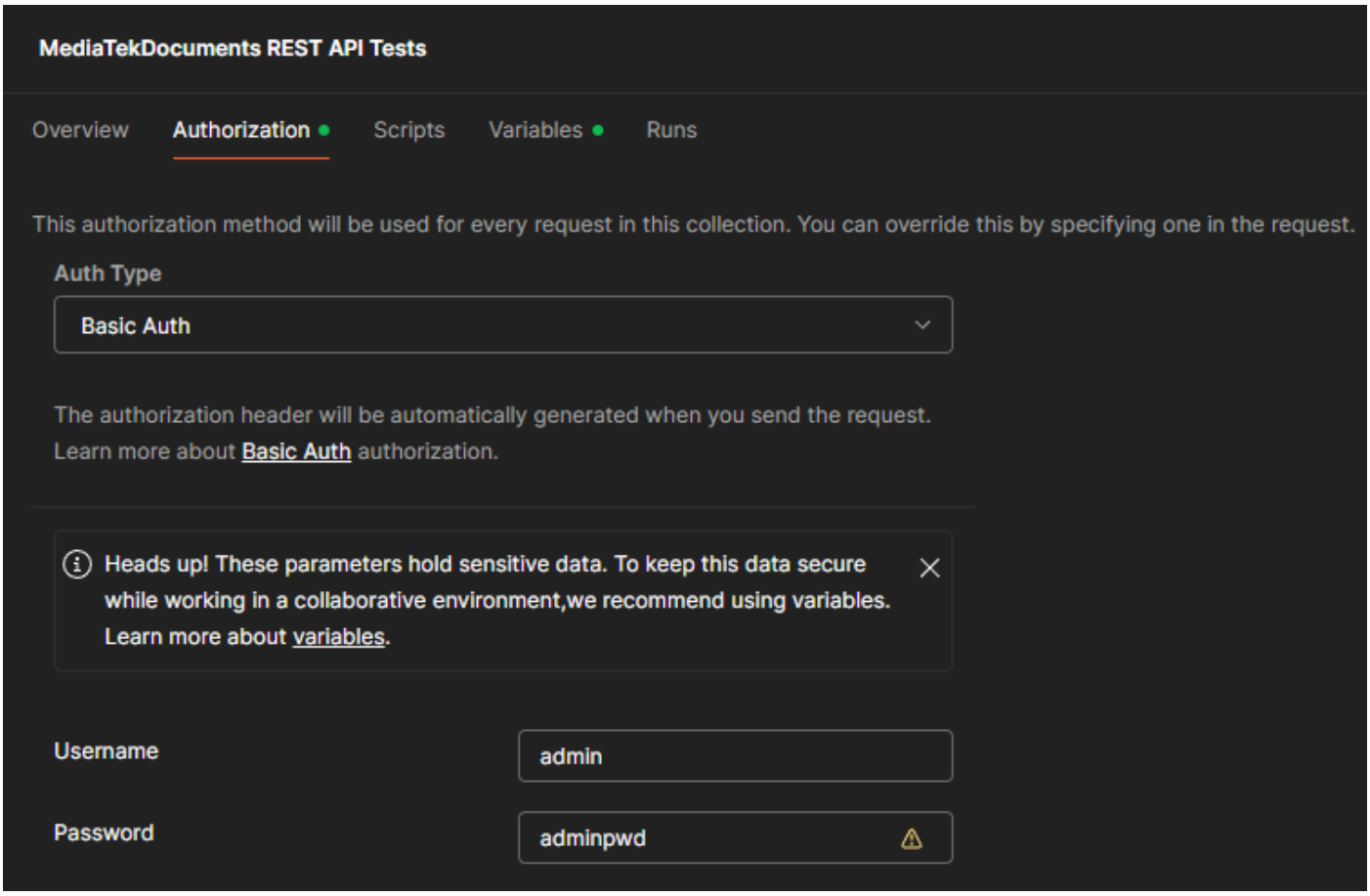
Résultat du test :

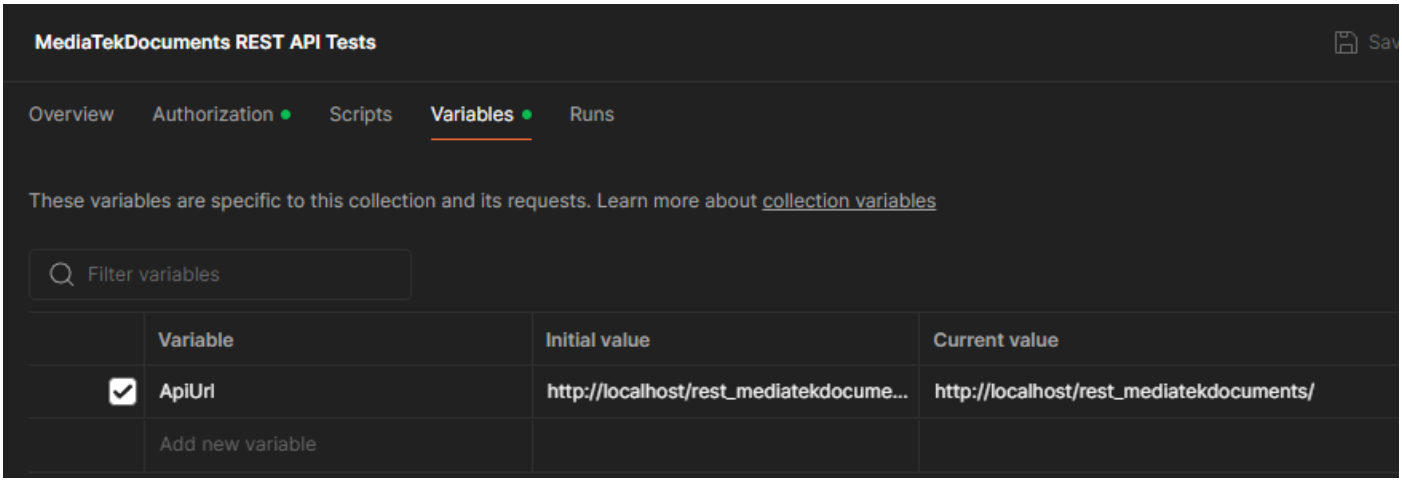
Série de tests achevée : 25 tests (25 réussi(s), 0 non réussi(s), 0 ignoré(s)) exécutés en 2,4 s

Test	Durée	Caractéris...	Message d'erreur
MediaTekDocuments.Tests (24)	370 ms		
SpecFlowMediaTekDocuments (1)	1,1 s		
SpecFlowMediaTekDocuments.Feat...	1,1 s		
RechercheFeature (1)	1,1 s		
RechercheLivre	1,1 s		

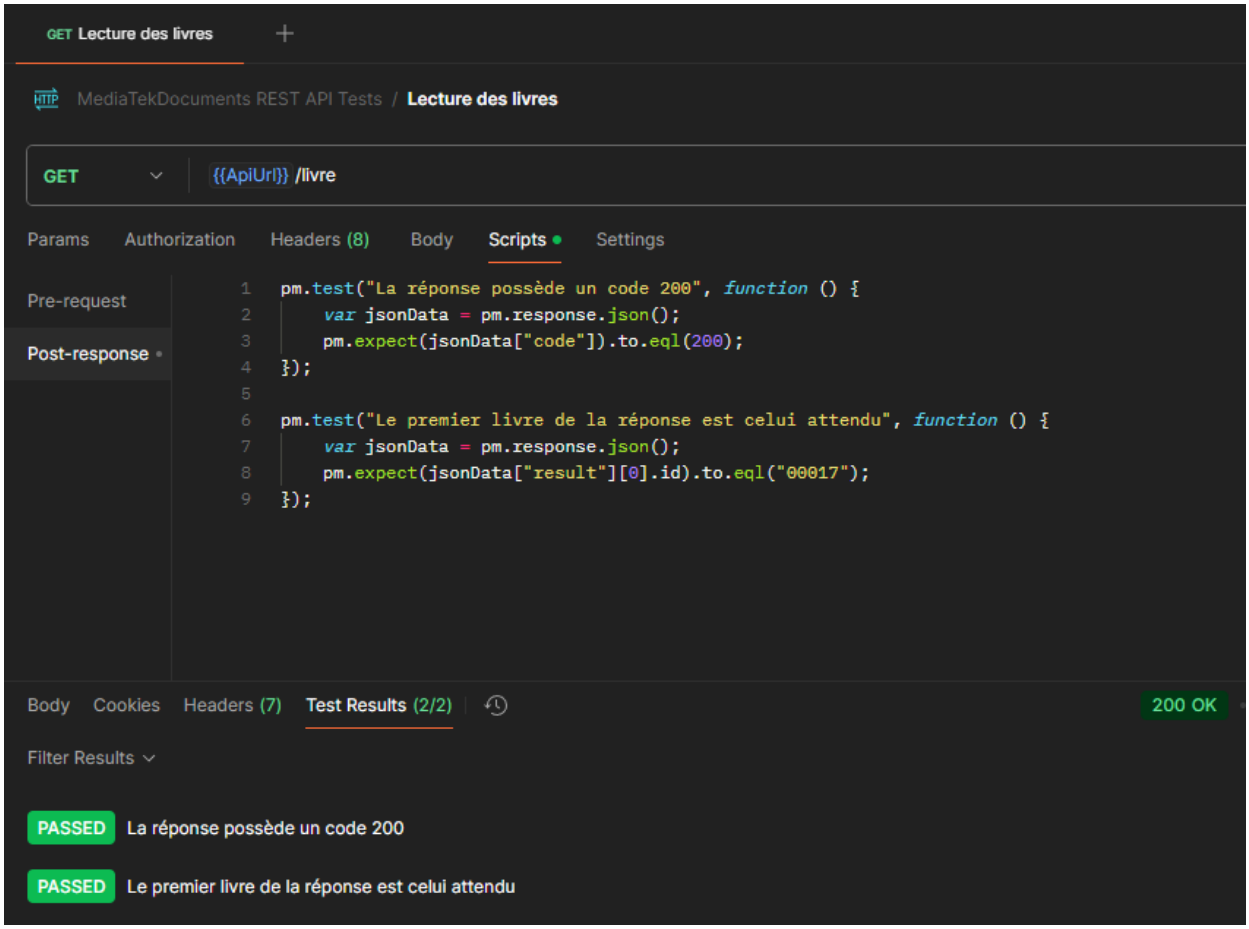
Enfin, nous allons créer des tests fonctionnels avec Postman pour contrôler toutes les fonctionnalités de l'API d'accès à la BDD :

On commence par créer une collection de tests et on configure la variable ApiUrl et l'authentification :



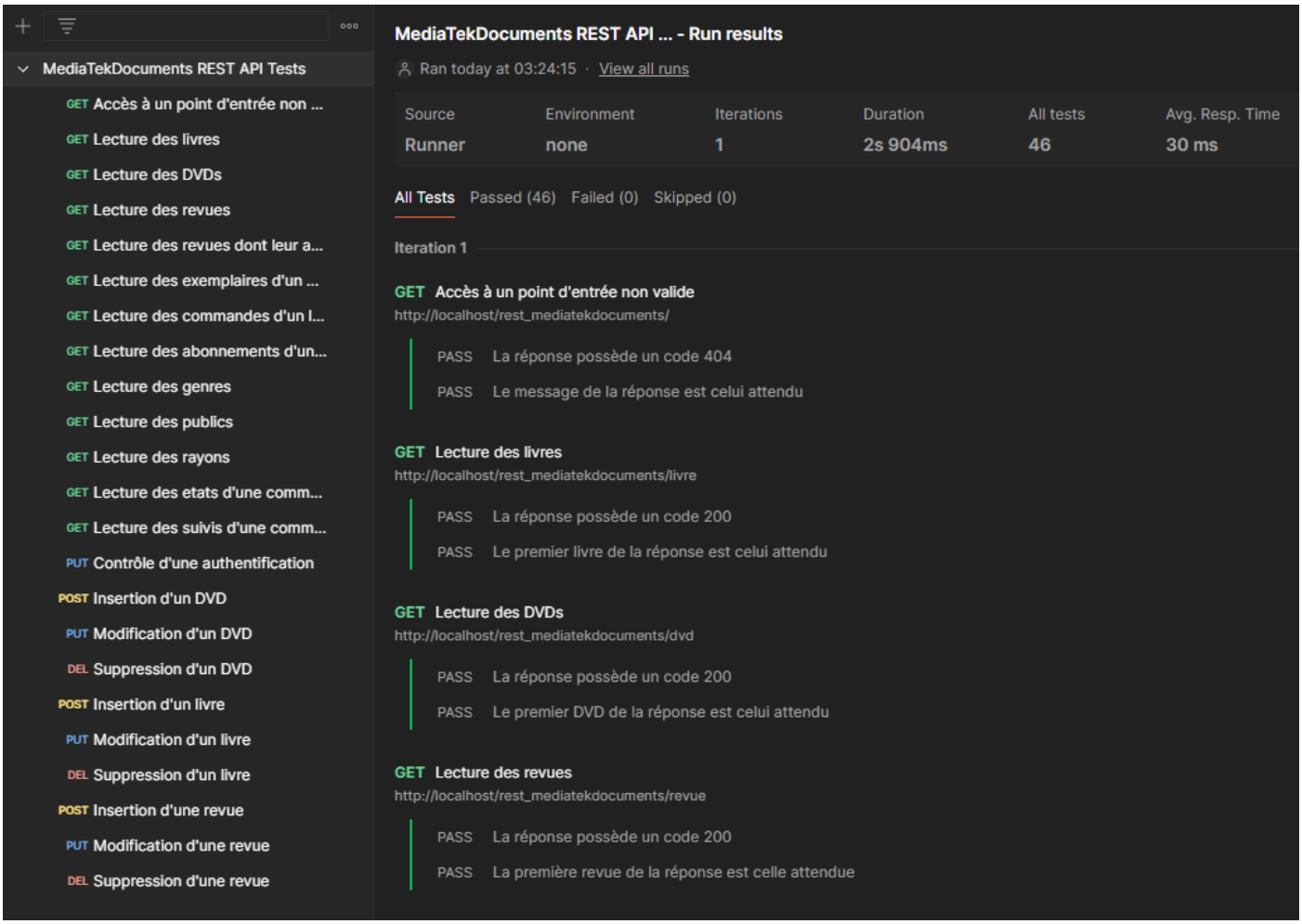


Ensuite, on crée des tests pour chaque fonctionnalité de l'API, voici par exemple le test pour la lecture des livres :



On effectue ici une requête GET classique mais on ajoute dans les scripts des tests. Le premier test vérifie que la réponse retourne un code 200 et le deuxième vérifie que l'identifiant du premier livre de la réponse correspond à celui attendu.

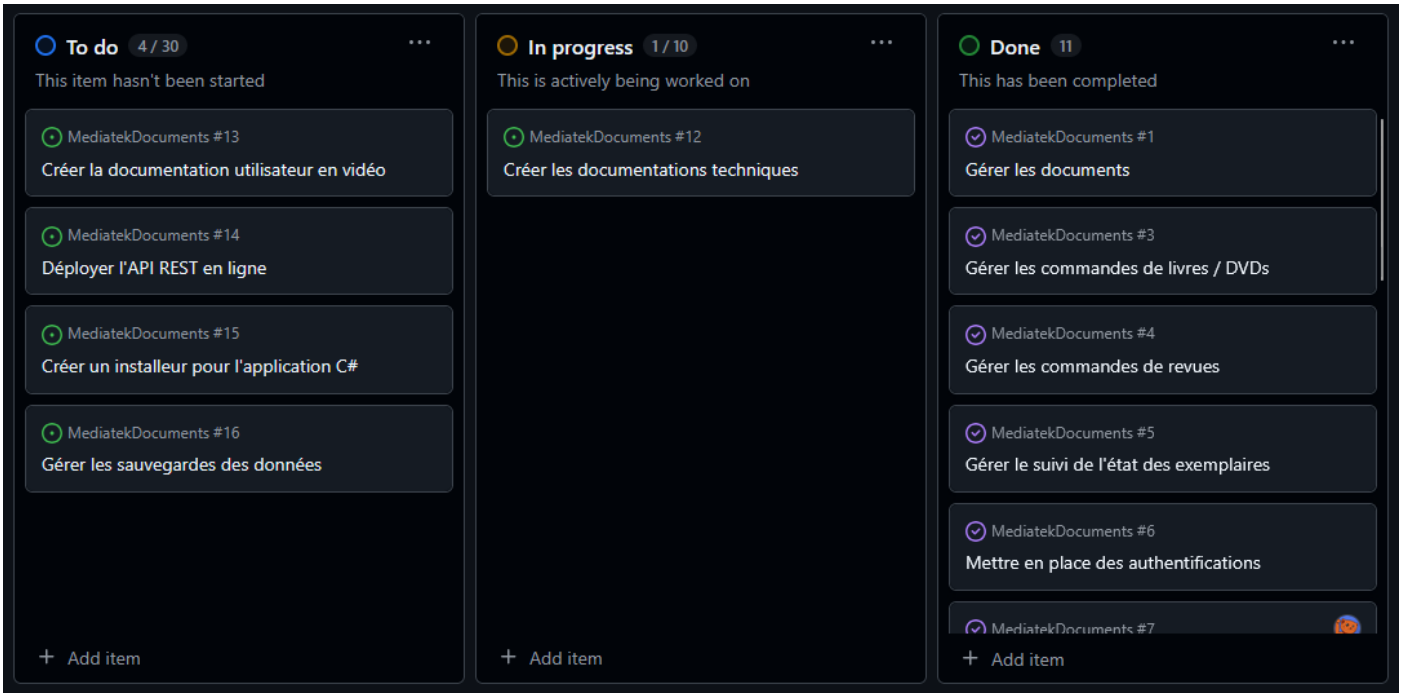
Une fois tous les tests construits, nous pouvons tous les lancer et obtenir le résultat :



Cette tâche a pris plus de temps que prévu car des problèmes ont été rencontrés à l'utilisation de SpecFlow. En effet, le projet a été abandonné donc il a fallu trouver des solutions pour quand même pouvoir l'installer et l'utiliser.

Tâche 2 : Créer les documentations techniques

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



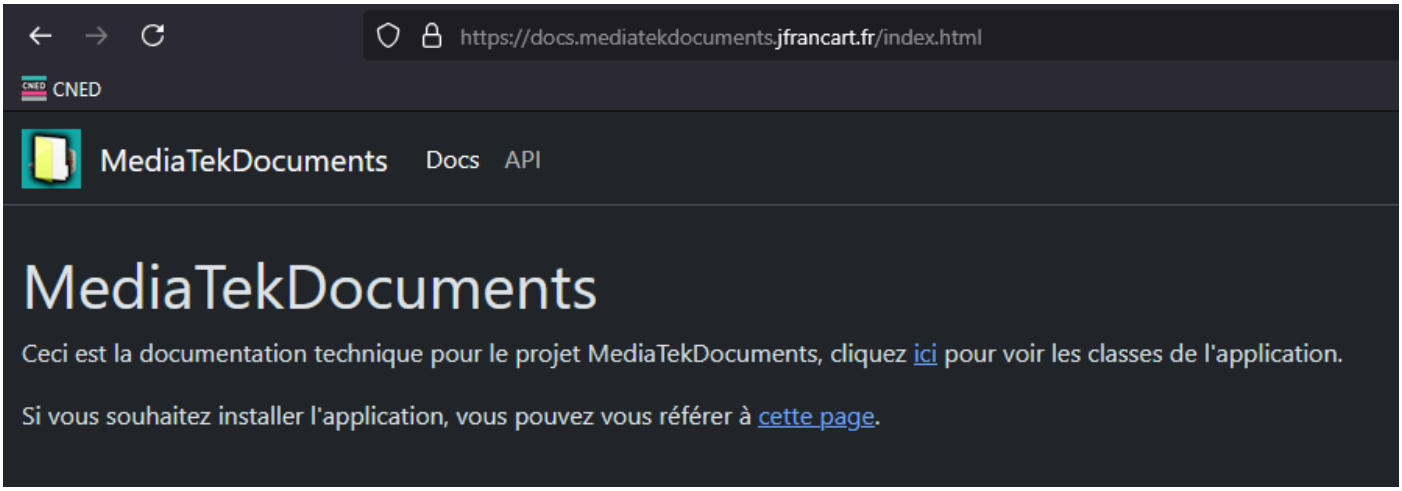
Dans cette tâche, il nous est demandé de générer et déployer les documentations techniques pour l'application C# ainsi que l'API REST. Les procédés utilisés vont être les mêmes que pour mes ateliers professionnels précédents.

Pour l'application C#, on utilise un projet de documentation DocFx qui permet d'avoir des pages avec les classes de l'application, leur signature et leur description ainsi que d'autres pages qui peuvent être utiles dans une documentation. Nous pouvons ensuite automatiquement déployer le projet DocFx vers une GitHub Pages à l'aide d'une action GitHub.

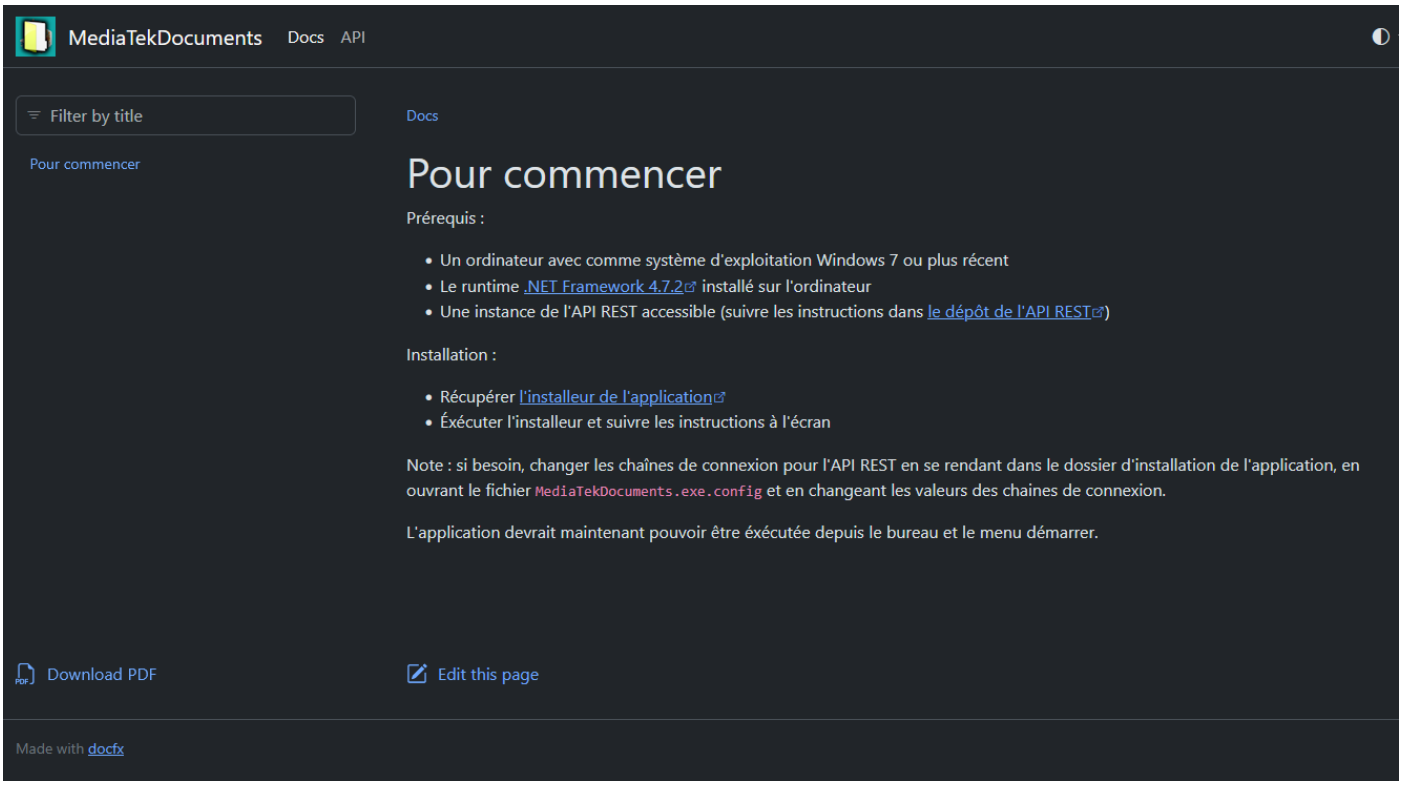
Pour plus de détails sur le fonctionnement, voir l'atelier professionnel PersonnelManager à la page 13

Voici le résultat :

Accueil de la documentation technique :

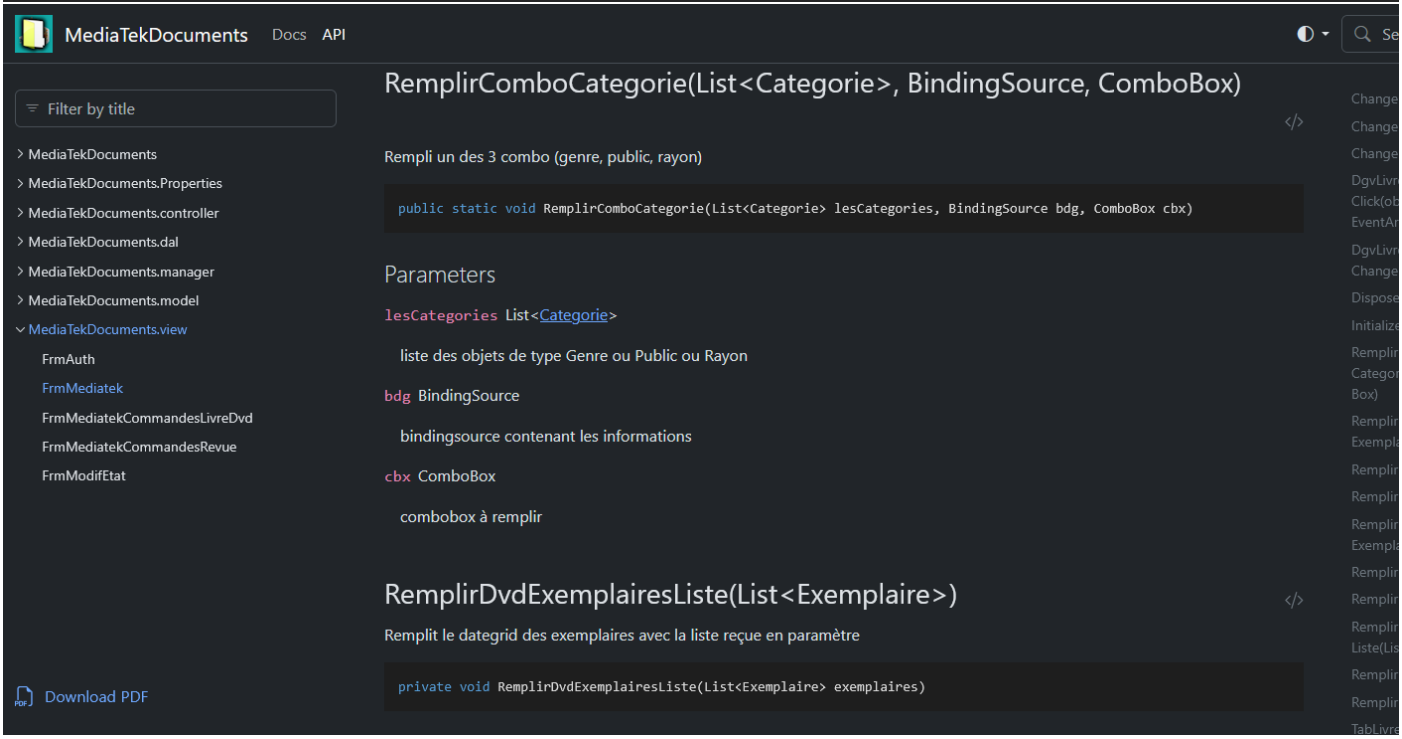


Article "Pour commencer"



Exemple d'affichage d'une méthode :

```
/// <summary>
/// Rempli un des 3 combo (genre, public, rayon)
/// </summary>
/// <param name="lesCategories">liste des objets de type Genre ou Public ou Rayon</param>
/// <param name="bdg">bindingsource contenant les informations</param>
/// <param name="cbx">combobox à remplir</param>
18 usages  CNED SLAM +1
public static void RemplirComboCategorie(List<Categorie> lesCategories, BindingSource bdg, ComboBox cbx)
```



Passons maintenant à la génération de la documentation technique de l'API REST.

Pour générer la documentation technique de l'API REST, on utilise l'outil phpDocumentor. Là encore, on utilise GitHub Actions et GitHub Pages pour ensuite déployer la documentation technique vers un site.

Voici l'exemple de l'affichage d'une méthode dans la documentation technique :

```php
/**
 * Retourne toutes les commandes pour un livre / DVD
 * @param array|null $champs Les champs contenant un champ 'id' avec l'identifiant du document
 * @return array|null La liste des commandes du document ou null en cas d'erreur
 */
1 usage  Refragg
private function selectCommandesDocument(?array $champs) : ?array{
```

selectCommandesDocument()

MyAccessBDD.php : 944

Retourne toutes les commandes pour un livre / DVD

```
private selectCommandesDocument(array<string|int, mixed>|null $champs) :
array<string|int, mixed>|null
```

Parameters

$champs : array<string|int, mixed>|null

Les champs contenant un champ 'id' avec l'identifiant du document

Return values

array<string|int, mixed>|null —
La liste des commandes du document ou null en cas d'erreur

selectExemplairesDocument()

MyAccessBDD.php : 924

récupère tous les exemplaires d'un document

```
private selectExemplairesDocument(array<string|int, mixed>|null $champs) :
array<string|int, mixed>|null
```

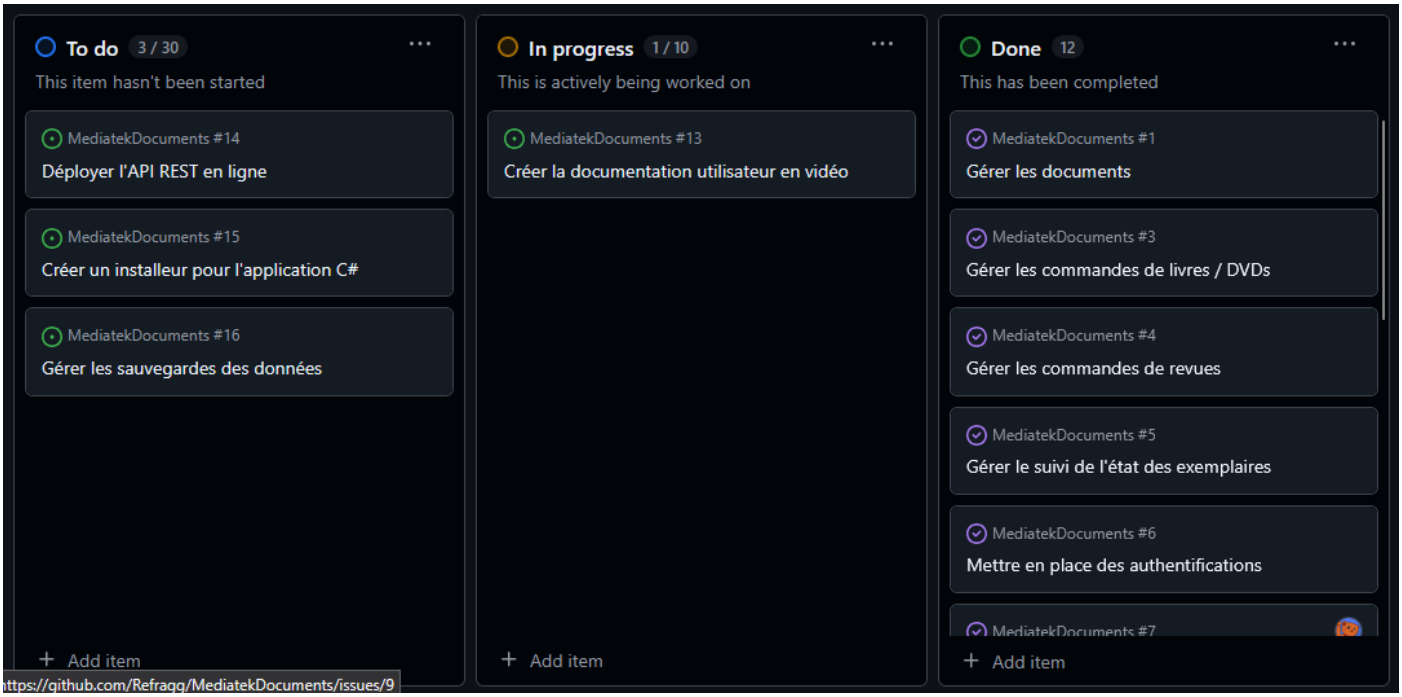
Parameters

$champs : array<string|int, mixed>|null

insertCommandeDocument()
insertDocument()
insertDvd()
insertLivre()
insertOneTupleOneTable()
insertRevue()
selectAbonnements()
selectAllDvd()
selectAllLivres()
selectAllRevues()
selectCommandesDocument()
selectExemplairesDocument()
selectRevuesAbonnementAExpirationProchaine()
selectTableSimple()
selectTuplesOneTable()
supprimerDocument()
supprimerDvd()
supprimerLivre()
supprimerLivreDvd()
supprimerRevue()
updateAbonnement()
updateCommande()
updateCommandeDocument()
updateDocument()
updateDvd()
updateExemplaire()
updateLivre()
updateOneTupleOneTable()
updateRevue()

Tâche 2 : Créer la documentation utilisateur en vidéo

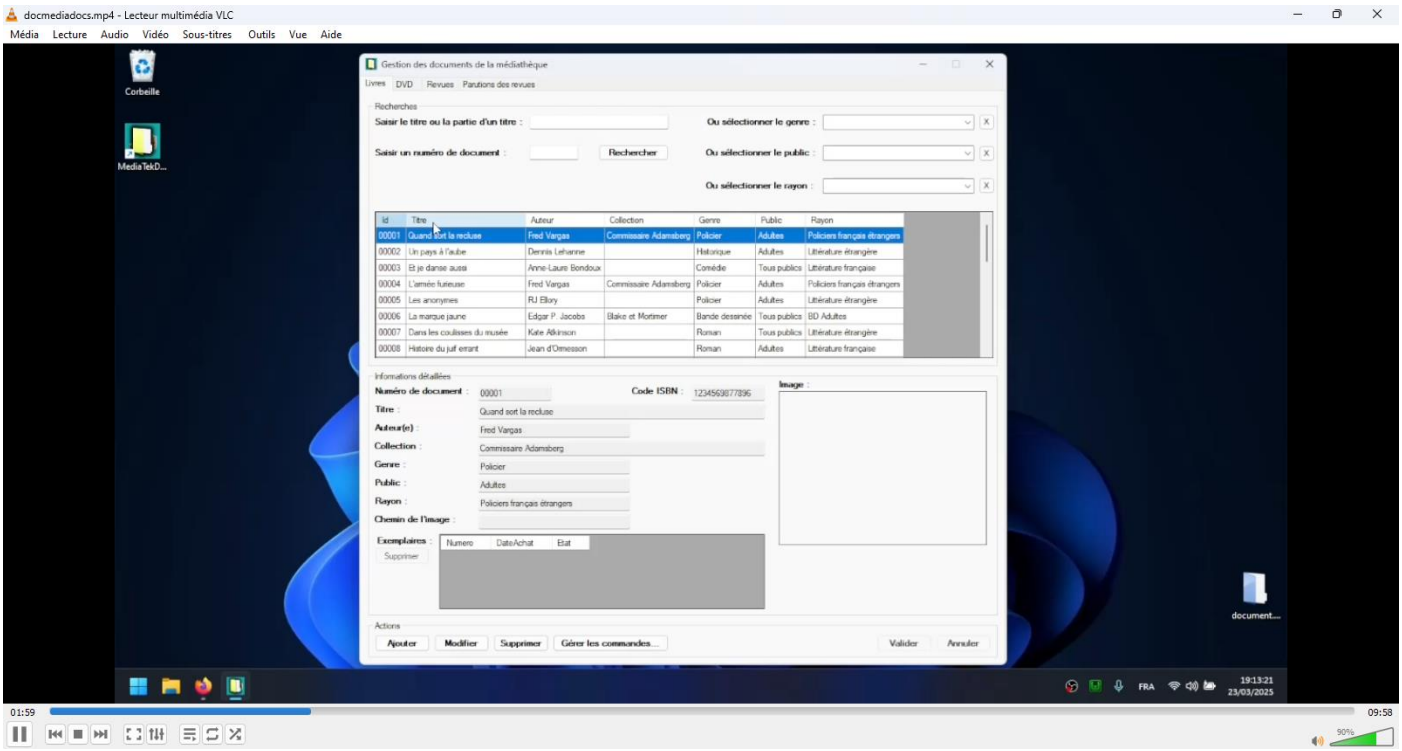
Temps de réalisation estimé : 2h - Temps de réalisation réel : 3h



Pour la réalisation de cette tâche, il nous est demandé de créer une vidéo de 10 minutes maximum présentant l'ensemble des fonctionnalités de l'application C#. Cette vidéo devra être accompagnée d'explications claires quant à l'usage de l'application.

J'ai commencé par créer une liste de points à aborder dans la vidéo. Ensuite, j'ai préparé l'outil pour enregistrer la vidéo : OBS Studio. Une fois la vidéo enregistrée, j'ai réalisé un léger montage avec Shotcut afin de rendre la vidéo plus agréable, fluide et de la raccourcir à moins de 10 minutes.

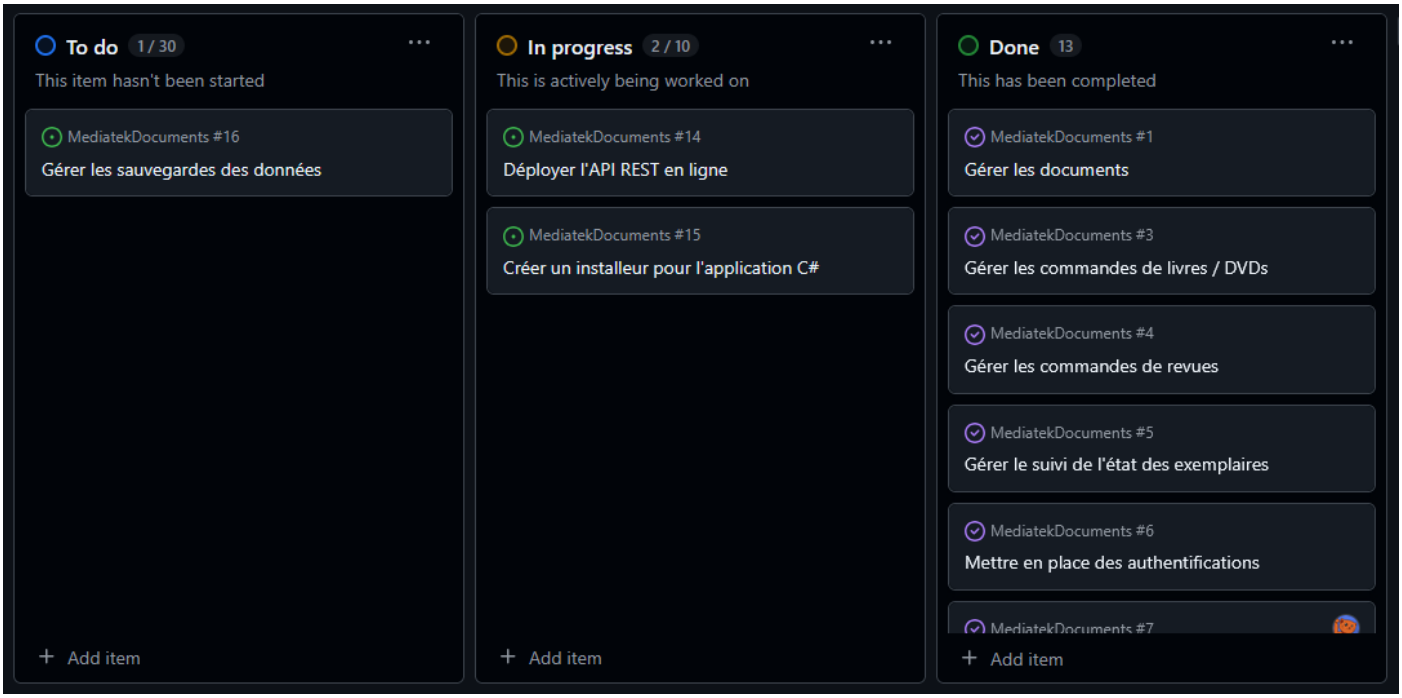
Au final, le durée de la vidéo est de 9:58 :



Étape 8 : Déploiement et gestion de la sauvegarde des données

Tâche 1 : Déployer le projet

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



Commençons par déployer l'API REST en ligne, elle sera déployée sur le même serveur utilisé que pour l'application Mediatekformations. La procédure de déploiement est d'ailleurs identique à quelques choses près :

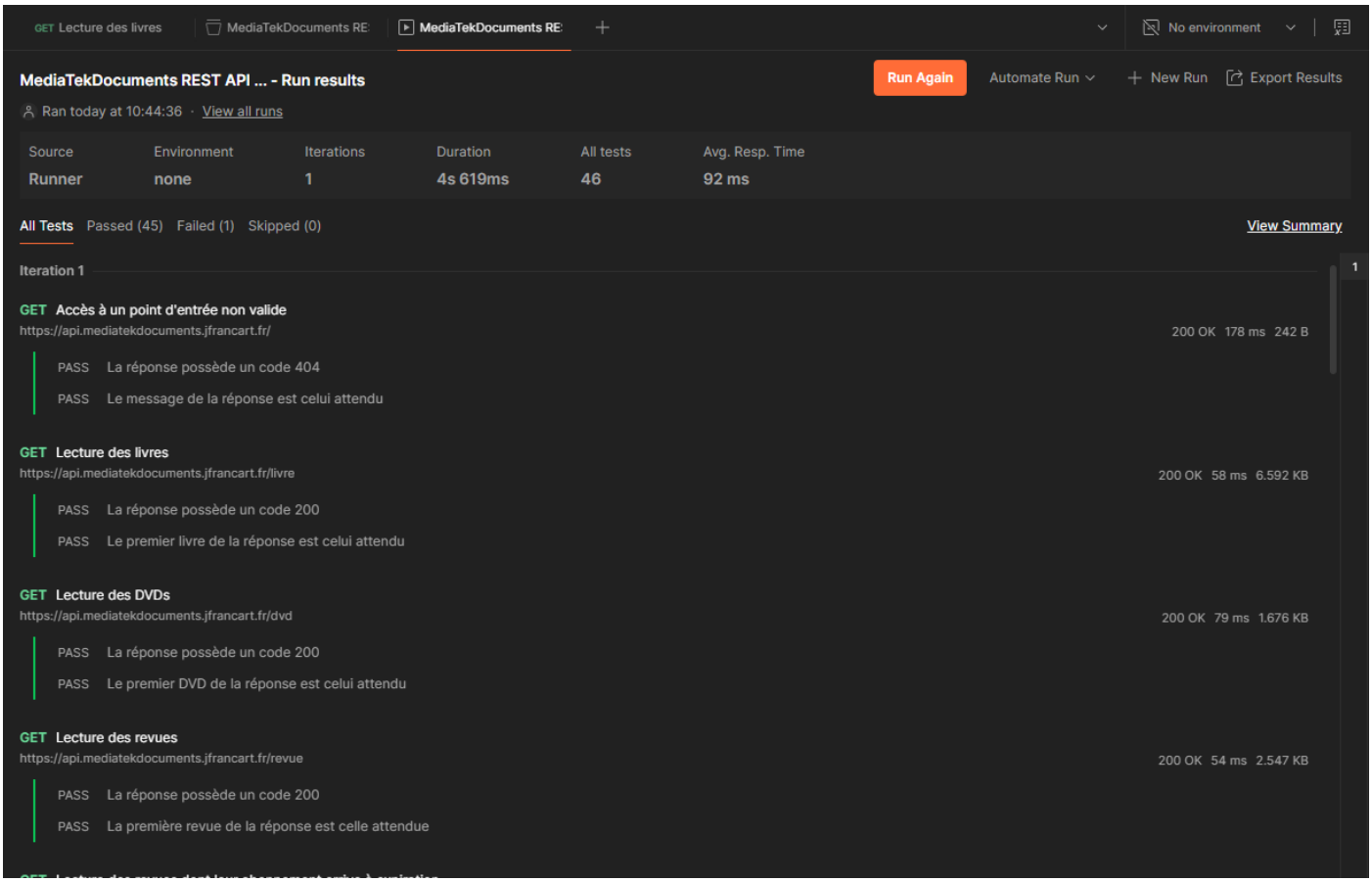
Après connexion sur le serveur, j'ai commencé par cloner et paramétrer un projet utilisant docker-compose-lamp. Ce dépôt contient une base pour déployer un stack LAMP (Linux, Apache, MySQL, PHP) facilement avec Docker compose.

Pour déployer le site correctement, plusieurs étapes ont été réalisées :

- Clonage du code source de rest_mediatekdocuments
- Configuration du DNS du domaine pour rediriger api.mediatekdocuments.jfrancart.fr vers le serveur
- Paramétrage de la base docker-compose-lamp pour l'adapter au projet
- Ajout de la base de données depuis le panel phpMyAdmin
- Configuration du proxy inversé de test utilisé (permet la gestion centralisée de plusieurs sites sur un serveur)
- Modification des paramètres d'accès à la base de données dans le fichier .env
- Changement des identifiants de l'accès à l'API REST
- Installation des bundles utilisés dans l'application avec `composer install`
- Une fois que la configuration fonctionne sur le proxy inversé de test, déploiement sur celui de production
- Création de services systemd pour automatiser le démarrage du site lors du redémarrage du serveur
- Ajout du domaine *.mediatekdocuments.jfrancart.fr aux domaines couverts par le certificat SSL utilisé par le serveur

Une fois ces étapes réalisées, l'API REST est déployée et disponible sur https://api.mediatekdocuments.jfrancart.fr

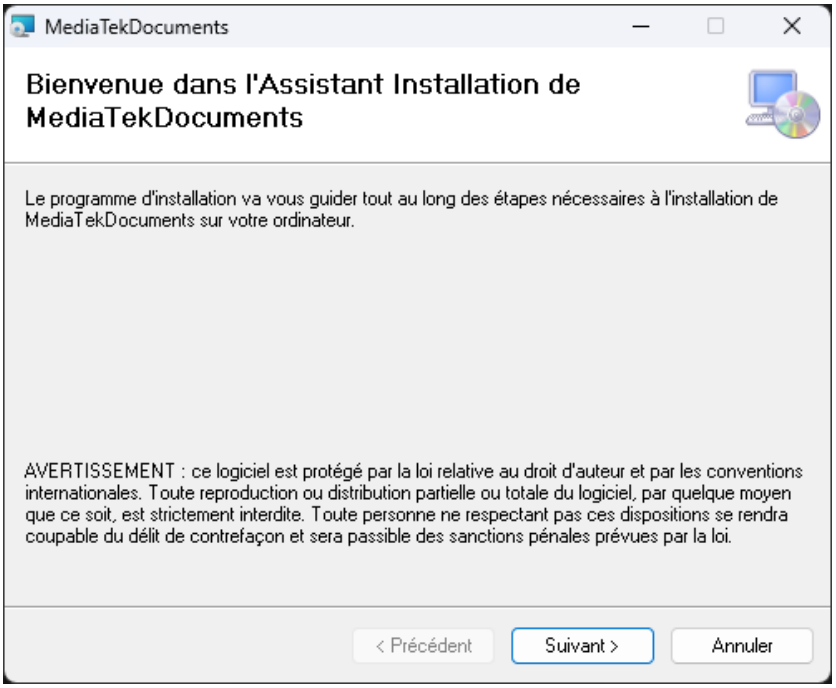
On peut ensuite tester l'API en lançant la collection de tests Postman :

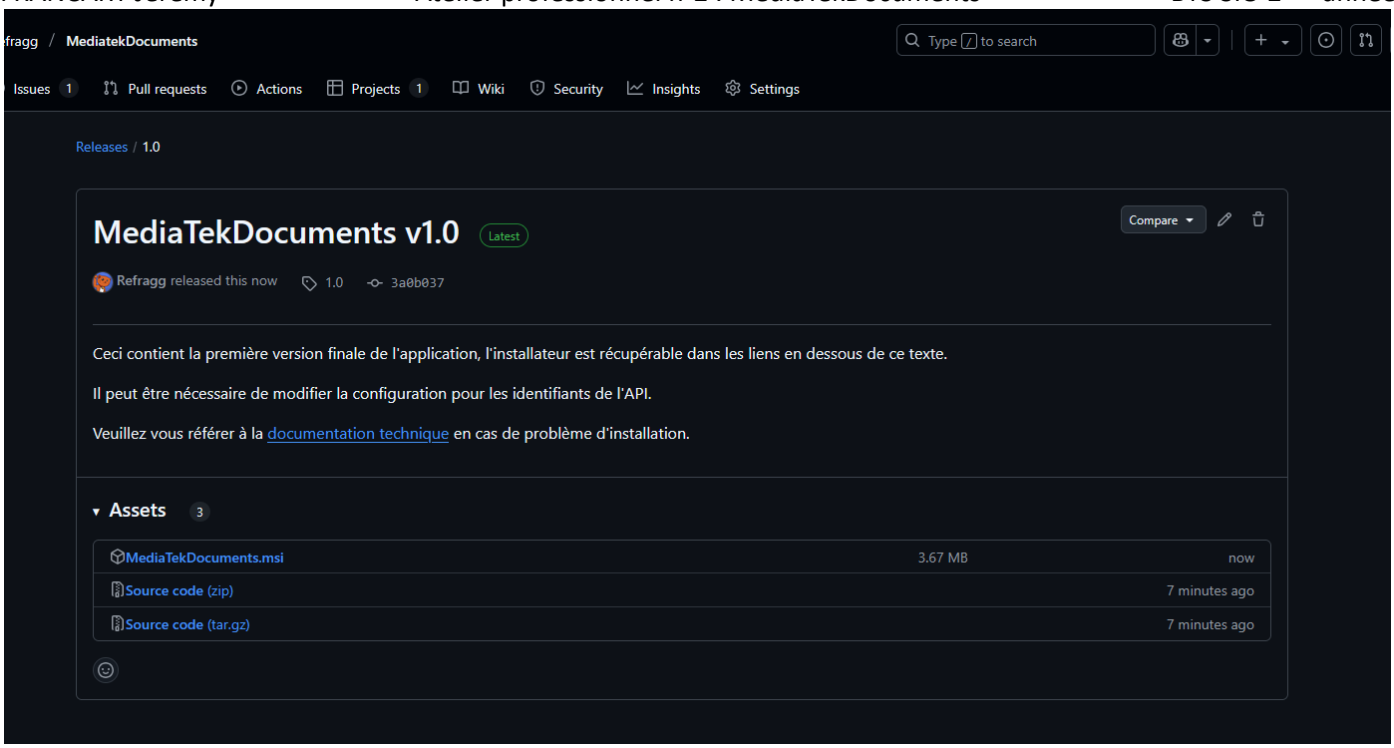


Un test ne passe pas, c'est le test de la lecture des revues dont leur abonnement arrive à expiration, on peut l'ignorer, c'est juste que les revues arrivant à expiration ont changé entre temps.

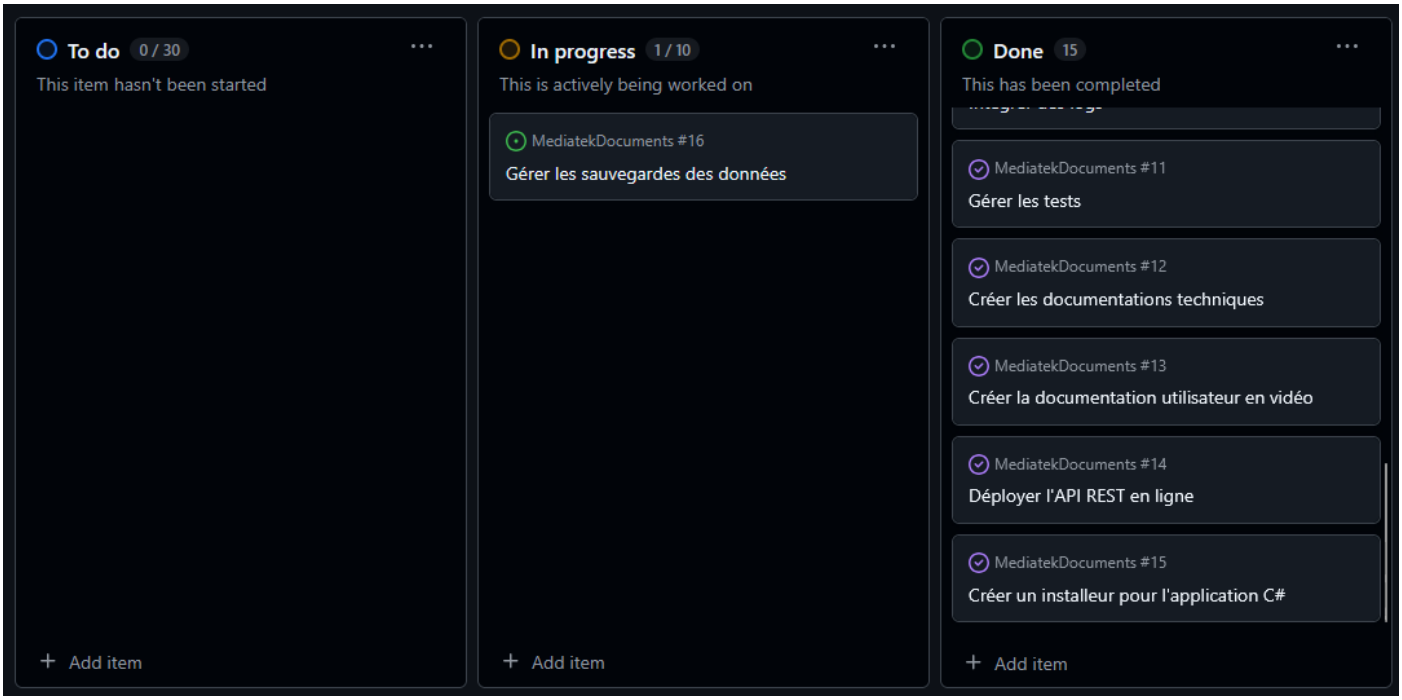
Désormais, nous allons créer un installeur pour l'application C#. La démarche est la même que pour l'application PersonnelManager, plus de détails sont disponibles dans le compte rendu de l'atelier professionnel PersonnelManager à la page 21.

Voici le résultat avec l'installeur et son accessibilité depuis le dépôt de l'application :





Tâche 2 : Gérer les sauvegardes des données
Temps de réalisation estimé : 1h - Temps de réalisation réel : 1h



Dans cette tâche, il nous est demandé de gérer la sauvegarde de la base de données.

Au niveau de l'organisation de la sauvegarde : on va effectuer une sauvegarde complète tous les jours. Sachant que la base de données est légère, nous pouvons effectuer une sauvegarde tous les jours sans grand impact sur les performances de l'API et sans grand impact sur le stockage du serveur.

Le procédé pour sauvegarder la base de données est le même que pour le site internet Mediatekformation. Cependant, un script de restauration a été ajouté.

Voici le script de sauvegarde de la base de données :

```
#!/bin/sh
DATE=`date -I`

# Suppression de l'ancien fichier de sauvegarde de BDD
#find /root/rest_mediatekdocuments-bdd-bkps/bdd* -mtime -1 -exec rm {} \;

# Sauvegarde de la BDD dans /root/rest_mediatekdocuments-bdd-bkps/
docker exec rest_mediatekdocuments-mariadb106 /usr/bin/mysqldump -u root --password=tiger --databases mediatek86
--single-transaction | gzip > /root/rest_mediatekdocuments-bdd-bkps/bddbackup_${DATE}.sql.gz
```

Puis, voici le script de restauration de la base de données :

```
#!/bin/sh

if [ ! -f $1 ]; then
    echo "Le fichier de backup n'existe pas"
fi

gunzip -c $1 > rest_mediatekdocuments_bkp-tmp.sql

docker exec -i rest_mediatekdocuments-mariadb106 /bin/bash -c "echo 'DROP DATABASE mediatek86;' | mysql -u root -ptiger"
docker exec -i rest_mediatekdocuments-mariadb106 /usr/bin/mysql -u root --password=tiger < rest_mediatekdocuments_bkp-tmp.sql

rm rest_mediatekdocuments_bkp-tmp.sql
```

Pour l'utiliser, on l'invoque avec en argument le fichier de backup choisi :

```
:~# ./rest_mediatekdocuments-bdd-restore rest_mediatekdocuments-bdd-bkps/bddbackup_2025-03-24.sql.gz
```

Enfin, on ajoute cette ligne dans le crontab en effectuant crontab -e :

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
00 12 * * * /root/mediatekformation-bdd-bkp
00 12 * * * /root/rest_mediatekdocuments-bdd-bkp
```

Cette ligne indique qu'il faut exécuter le script de sauvegarde de la base de données chaque jour à 12h00.

Bilan

Le développement et le déploiement de cette solution applicative comprenant l'application et l'API REST m'a permis de mettre mes compétences à l'épreuve notamment sur l'exploitation d'une API, la construction d'interfaces, la gestion des données d'une application, l'authentification ainsi que la gestion de projet pour réaliser les tâches à temps et autres.

Les objectifs de développement, documentation et déploiement ont pu être menés selon moi à bien malgré le fait que certaines libertés ont été prises ainsi qu'une importante contrainte de temps ce qui a mené à certaines adaptations nécessaires.