

COMPTE RENDU MEDIATEKDOCUMENTS (DONOVAN BLOCUS)

SOMMAIRE

MISSION 1 Gérer les documents	4
A : Ajouter un document	6
B : Modifier un document.....	7
C : Supprimer un document	8
MISSION 2 Gérer les commandes.....	9
Tâche 2.1 : Gérer les commandes de Livre et DVD	9
Tâche 2 : gérer les commandes de revues	12
MISSION 3 Gérer le suivi de l'état des exemplaires	15
MISSION 4 Mettre en place des authentifications	16
A : Gérer les utilisateurs et les services	18
B : Gérer les alertes de fin d'abonnement.....	19
MISSION 5 assurer la sécurité, la qualité et intégrer des logs	21
Tâche 1 : corriger des problèmes de sécurité.....	21
Tâche 2 : contrôler la qualité	22
Tâche 2 : intégrer des logs.....	23
MISSION 6 tester et documenter	25
Tâche 1 : gérer les tests	25
Tâche 2 : créer les documentations techniques.....	27
Tâche 2 : créer la documentation utilisateur en vidéo.....	28
MISSION 7 tester et documenter	29
Tâche 1 : déployer le projet	29
Tâche 1 : gérer les sauvegardes des données	30
BILAN.....	31

CONTEXTE

Le réseau de médiathèques Médiaték86 a sollicité l'expertise de l'entreprise de services numériques InfoTech Services 86 afin de faire évoluer son application de bureau, devenue obsolète au regard de ses besoins actuels. Dans ce cadre, j'occupe le poste de technicien développeur junior au sein d'InfoTech Services 86. Ma mission consiste à participer activement à la refonte de l'application, en apportant des solutions techniques adaptées aux attentes du client, tout en respectant les contraintes fonctionnelles.

EXISTANT

MediatekDocuments est une application de bureau développée en C# à l'aide de l'environnement de développement Visual Studio 2019. Elle est conçue pour gérer les différents types de documents d'une médiathèque, tels que les livres, les DVD et les revues. L'application communique avec une base de données MySQL par l'intermédiaire d'une API REST, ce qui permet une interaction fluide entre l'interface utilisateur et les données stockées. À ce jour, le développement de l'application est partiellement achevé. Elle intègre déjà certaines fonctionnalités clés, notamment la recherche et l'affichage des informations relatives aux documents, ainsi que la gestion de la réception des nouveaux numéros de revues.

LANGAGES ET TECHNOLOGIES UTILISEES

Langages : C#, PHP, SQL

IDE : Visual Studio 2022, Apache NetBeans IDE 21

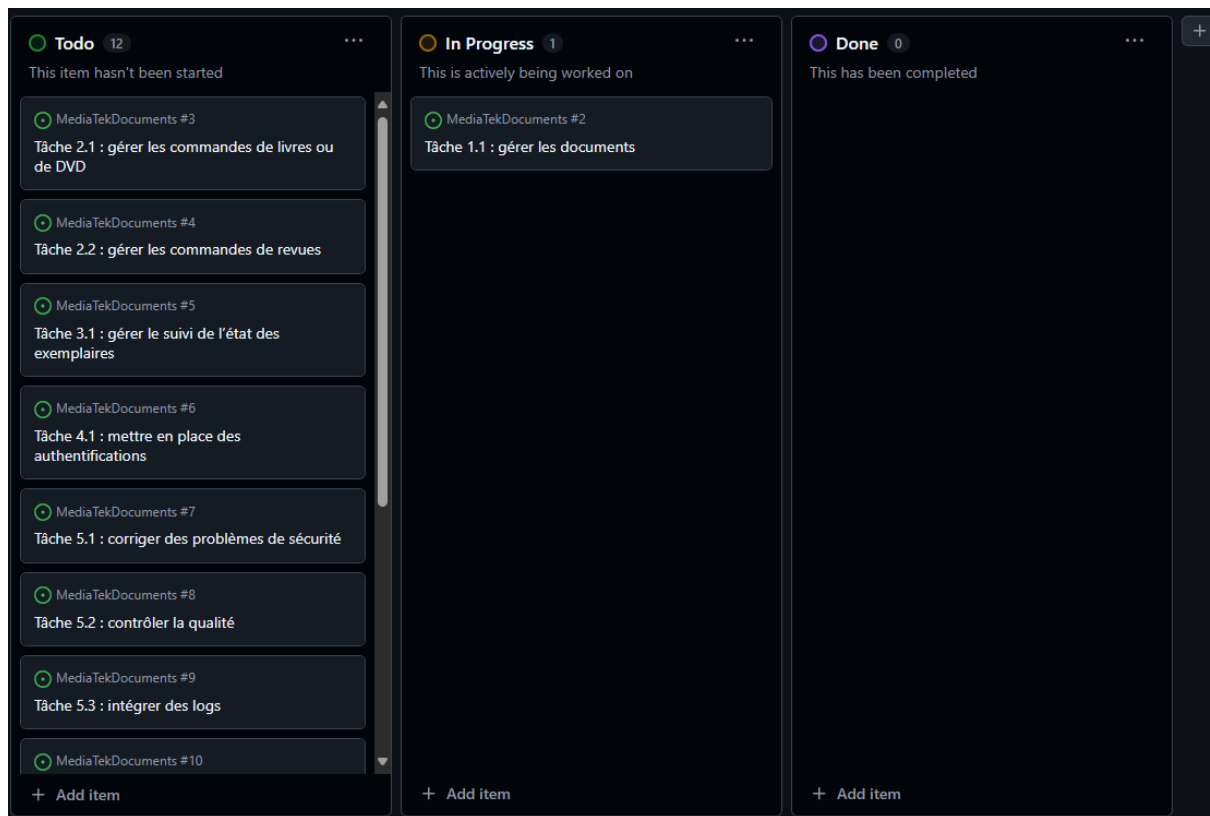
Gestion de version et Suivi de projet : GitHub

Hébergement : Hostinger

Autres outils : Postman (Test de requêtes vers API)

Serveur Local : Wampserver64

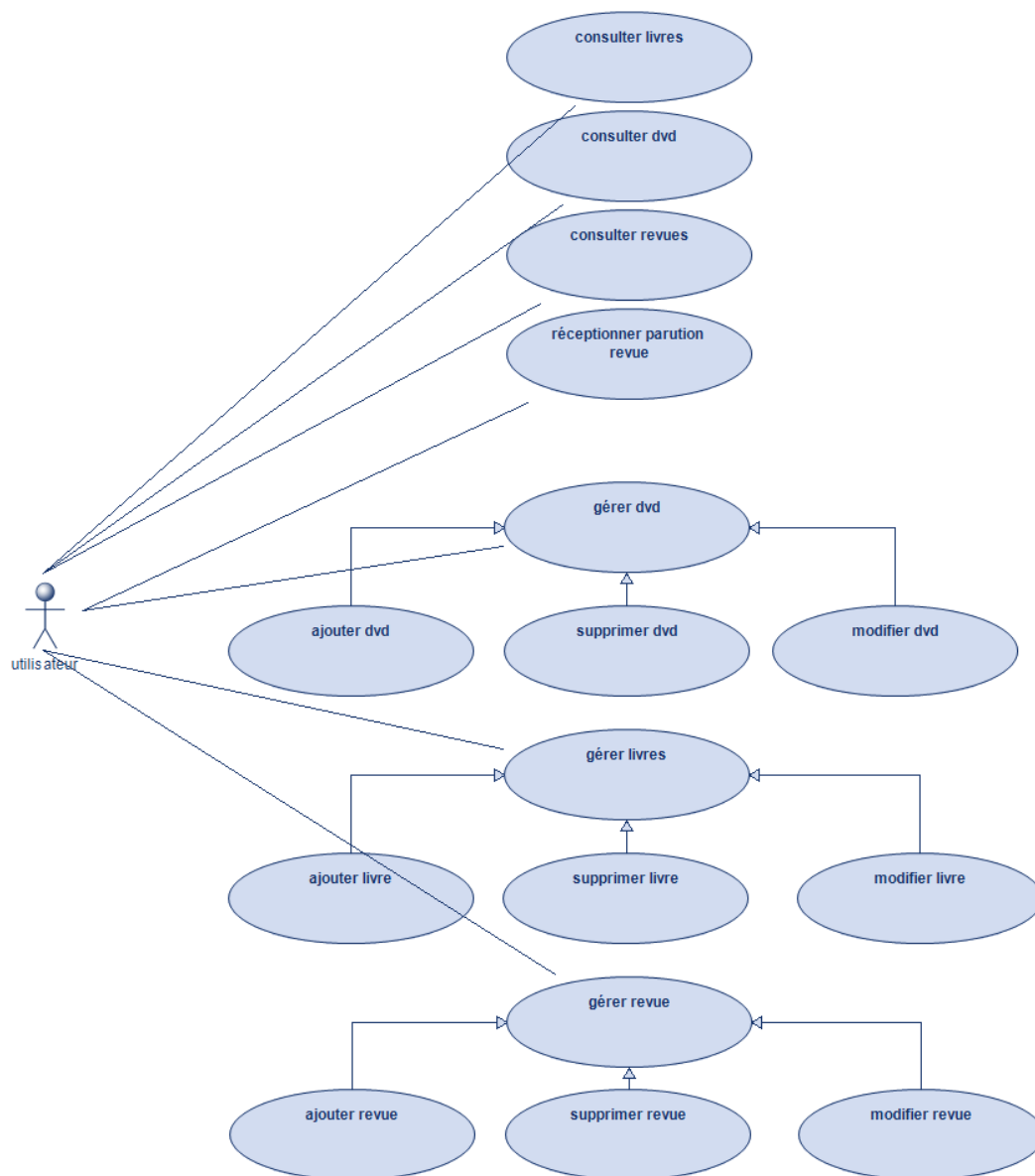
MISSION 1 Gérer les documents



Temps estimé : 8 heures

Temps réel : 8heures

Diagramme de cas d'utilisation :



Les étapes pour cette tâche est similaire pour chaque document, c'est pourquoi je documenterais seulement les modifications liées au livres.

A : Ajouter un document

J'ai d'abord commencé à modifier l'interface en ajoutant les boutons modifier, supprimer en dessous de la liste ou apparaissent les livres et j'ai également modifié le group box "information du livre" pour qu'il permette d'ajouter ou modifier un livre avec un bouton enregistrer et un bouton annuler l'ajout ou la modification.

Gestion des documents de la médiathèque

Livres DVD Revue Parutions des revues Commandes Livres Commandes Dvd Commandes Revues

Recherches

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

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

Ou sélectionner le rayon : X

Modifier Supprimer

Ajouter un livre

Numéro de document : Code ISBN : Image :

Titre :

Auteur(e) :

Collection :

Genre :

Public :

Rayon :

Chemin de l'image :

Enregistrer Annuler

Dans la classe Access j'ai ajouté la fonction "CreerLivre" qui insère un livre dans la base de données :

```
1 reference  
public bool CreerLivre(Livre livre)
```

J'ai également créé une fonction intermédiaire dans la classe "FrmMediatekController" pour accéder à cette fonction :

```
1 reference  
public bool CreerLivre(Livre livre)
```

Ensuite j'ai ajouté la méthode événementiel "btnEnregLivre_Click" pour le clic du bouton "Enregistrer" dans la classe du formulaire "FrmMediatek" qui vérifie si tous les champs sont remplis, et crée un objet livre via les informations saisies et appelle la fonction "CreerLivre" du controller en passant l'objet livre en paramètre :

```
!reference  
private void btnEnregLivre_Click(object sender, EventArgs e)
```

J'ai également modifié la fonction "insertLivre" de la classe MyAccessBDD.php de l'API REST pour y ajouter une transaction afin d'ajouter un tuple dans les tables document, livredvd et livre. De ce fait si l'un des inserts échoue, les insertions précédentes sont annulées à l'aide d'un rollback.

B : Modifier un document

Dans la classe Access j'ai ajouté la fonction "ModifierLivre" qui udpate un livre dans la base de données :

```
!reference  
public bool ModifierLivre(Livre livre)
```

Avec la fonction intermédiaire dans la classe controller :

```
public bool ModifierLivre(Livre livre)
```

J'ai ajouté la méthode événementiel "BtnDemandeModifLivre_Click" lors du clic sur le bouton modifier qui vérifie si un livre est sélectionné dans la liste et qui modifie une valeur booléenne de la classe "encoursdemodification". J'ai également modifié la méthode événementiel "btnEnregLivre_Click" qui vérifie s'il s'agit d'une modification et appelle la fonction "ModifierLivre" du controller dans le cas échéant :

```
!reference  
private void BtnDemandeModifLivre_Click(object sender, EventArgs e)
```

J'ai également modifié la fonction "updateLivre" de la classe MyAccessBDD.php de l'API REST pour y ajouter une transaction afin d'update un tuple dans les tables document, livredvd et livre. De ce fait si l'un des updates échoue, les updates précédentes sont annulées à l'aide d'un rollback.

Enfin, s'il s'agit d'une modification, alors l'élément id est passé en mode read-only et disable.

C : Supprimer un document

Dans la classe Access j'ai ajouté la fonction "SupprimerLivre" qui supprime un livre dans la base de données :

```
public bool SupprimerLivre(Livre livre)
```

Avec la fonction intermédiaire dans la classe controller :

```
public bool SupprimerLivre(Livre livre)
```

J'ai ajouté les fonctions "GetExemplaireDocument" et "GetCommandes" qui récupère les exemplaires et commandes associés à un document :

```
public List<Exemplaire> GetExemplairesDocument(string idDocument)
```

J'ai ajouté la méthode événementiel "btnDemandeSupprLivre_Click_Click" lors du clic sur le bouton supprimer qui vérifie si un livre est sélectionné dans la liste et si le compte d'exemplaire et de commande rattaché est de 0, puis appelle la fonction "SupprimerLivre" du controller :

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

J'ai également modifié la fonction "deleteLivre" de la classe MyAccessBDD.php de l'API REST pour y ajouter une transaction afin de supprimer un tuple dans les tables document, livredvd et livre. De ce fait si l'un des delete échoue, les delete précédent sont annulé à l'aide d'un rollback.

MISSION 2 Gérer les commandes

Temps estimé : 12 heures

Temps réel : 11 heures

Tâche 2.1 : Gérer les commandes de Livre et DVD

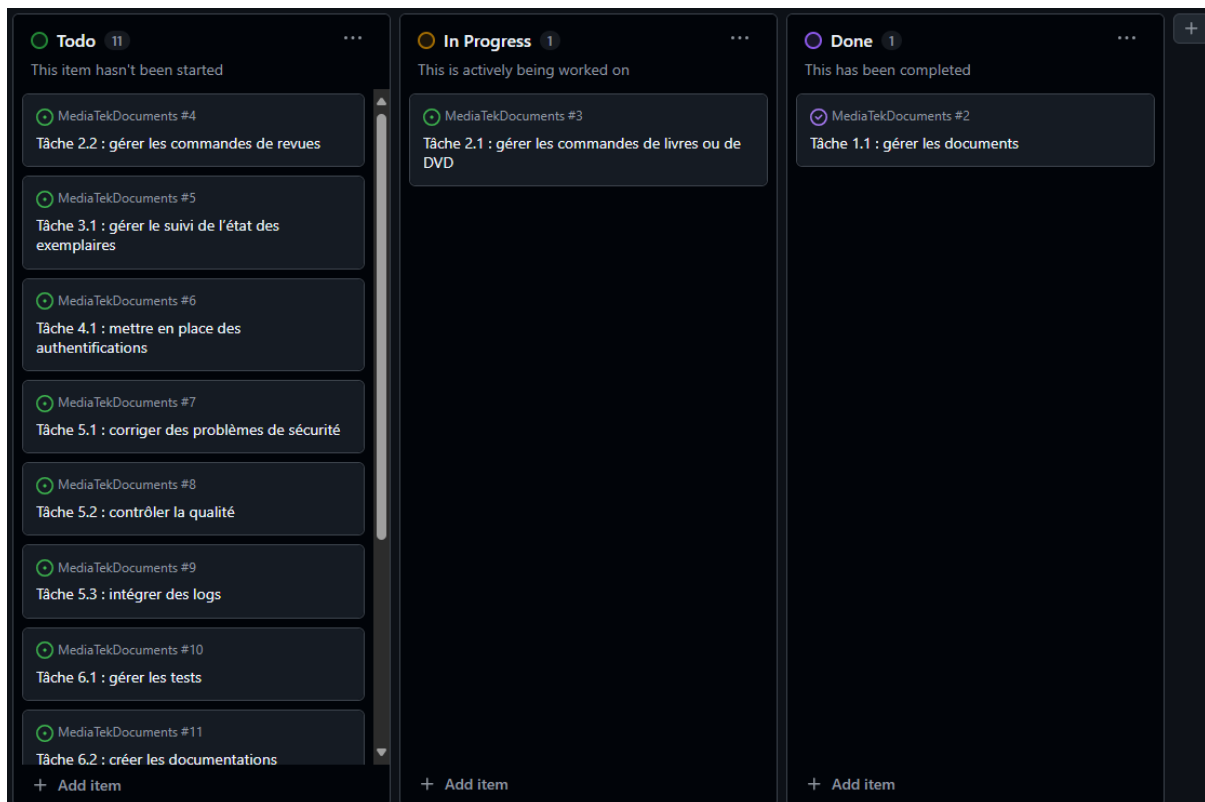
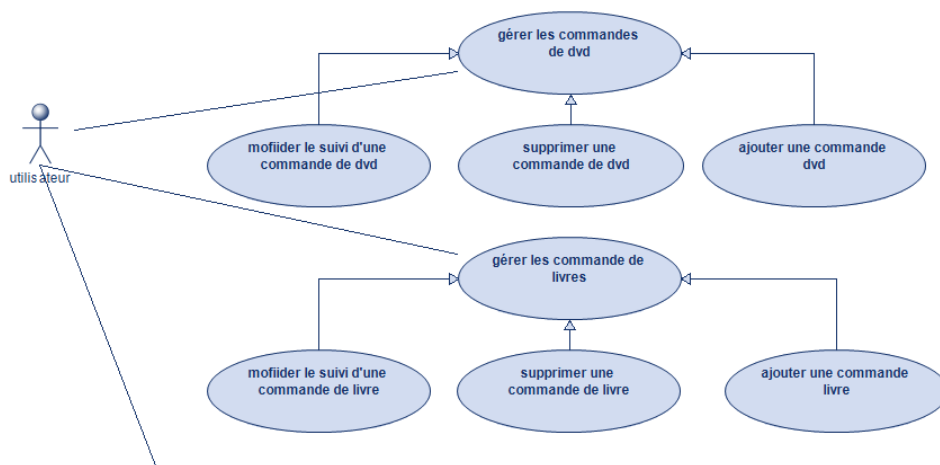


Diagramme de cas d'utilisation :



Les étapes pour cette tâche est similaire pour les livres et dvd, c'est pourquoi je documenterais seulement les modifications liées au livres.

J'ai d'abord commencé par modifier l'interface en ajouter les onglets "Commandes Livre" et "Commandes Dvd". Dans ces onglets il y a un datagridview permettant d'afficher la liste dans commandes associé à ces documents après recherche du numéro.

Il y a également une section réservée à l'ajout d'une commande et à la modification du suivi de la commande en plus d'un bouton supprimer :

Dans la base de données, j'ai ajouté la table "suivie" et la table "commandedocument".

Et dans la classe Access.cs j'ai ajouté la fonction "CreerCommandeDocument" qui créer une commande pour un livre ou un dvd dans la base de données avec également la fonction intermédiaire dans le contrôleur :

```
public bool CreerCommandeDocument(CommandeDocument commandeDocument)
```

Ainsi que la fonction SupprimerCommandeDocument :

```
1 reference  
public bool SupprimerCommandeDocument(CommandeDocument commandeDocument)  
{
```

Et la fonction ModifierCommandeDocument :

```
1 reference  
public bool ModifierCommandeDocument(CommandeDocument commandeDocument)  
{
```

J'ai créé la fonction selectAllCommandeDocument dans la classe MyAccessBDD.php dans l'API REST qui récupère toutes les commandes de livre ou dvd associé :

```
private function selectAllCommandeDocument(?array $champs) : ?array {
```

Et j'ai aussi ajouté la fonction insertCommandeDocument qui effectue une transaction pour insérer à la fois un tuple dans la table commande et commande document pour gérer l'héritage. Si l'un des inserts échoue, un rollback est effectué :

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

Et en respectant le même principe de transaction j'ai ajouté la fonction deleteCommandeDocument :

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

J'ai notamment ajouté la méthode événementiel btnEnregCmdLivre_Click pour le clic sur le bouton enregistrer qui vérifie si un livre est sélectionné et que les informations de commandes sont toutes renseignées avant d'appeler la fonction "CreerCommandeDocument" du contrôleur :

```
1 reference  
private void btnEnregCmdLivre_Click(object sender, EventArgs e)  
{
```

Et la méthode événementielle pour le clic sur le bouton supprimer qui vérifie si une commande est sélectionnée et appelle la fonction SupprimerCommandeDocument du contrôleur :

```
1 reference  
private void btnSupprCmdLivre_Click(object sender, EventArgs e)  
{
```

Ainsi que la méthode événementielle pour le clic sur le bouton "modifier suivi" qui vérifie si une commande est sélectionnée et modifie son suivi avec la valeur renseignée dans le combobox. Il y a aussi des vérifications faites pour ne pas modifier le suivi vers une étape inférieure, et ne pas modifier le suivi d'une commande si la valeur du combobox correspond à "réglée" alors que la commande n'est pas livrée :

```
1 reference  
private void btnModSuiviCmdLivre_Click(object sender, EventArgs e)  
{
```

Tâche 2 : gérer les commandes de revues

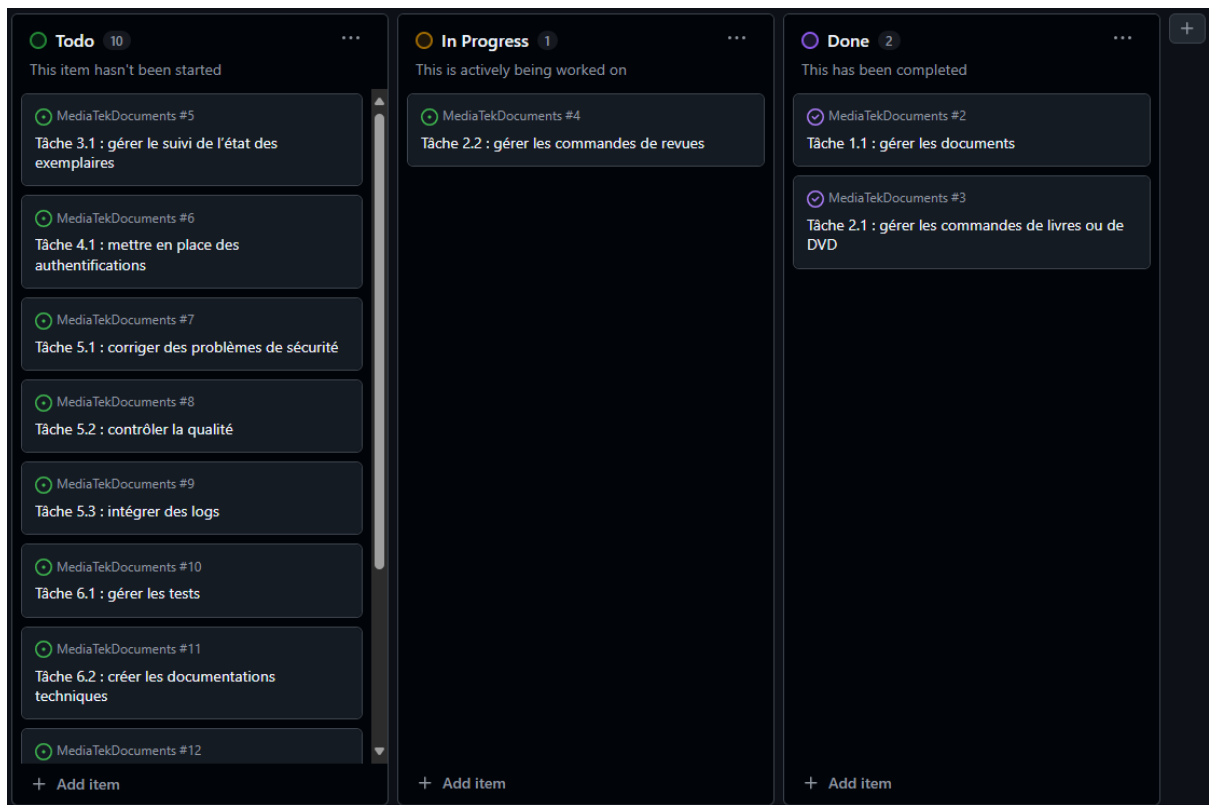
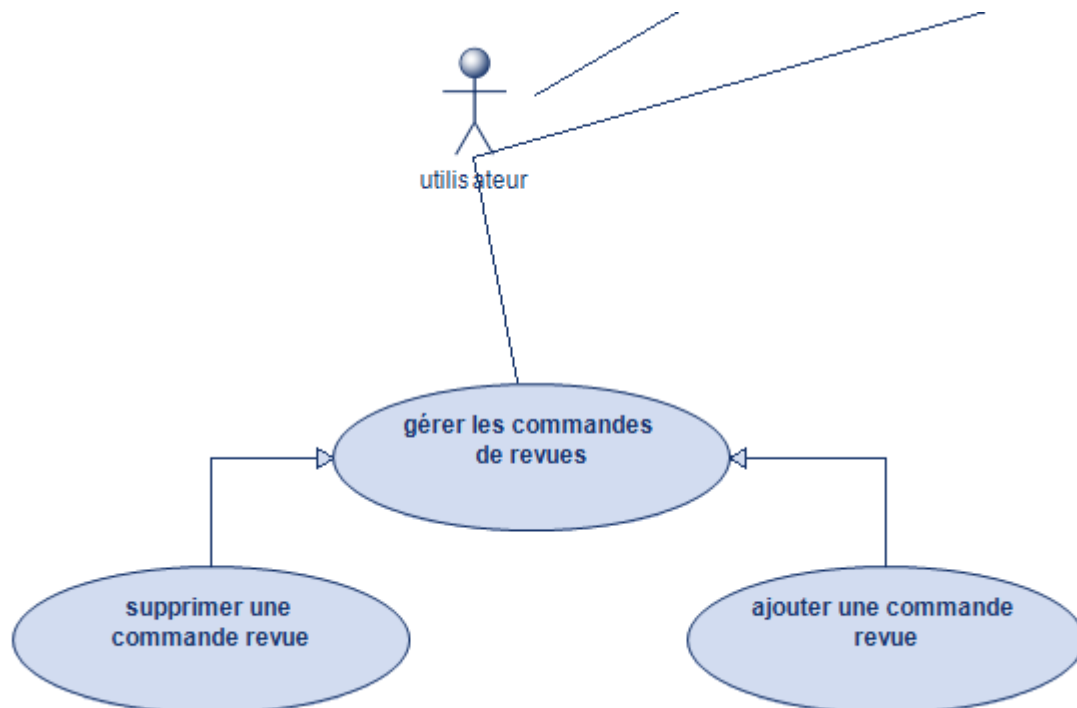
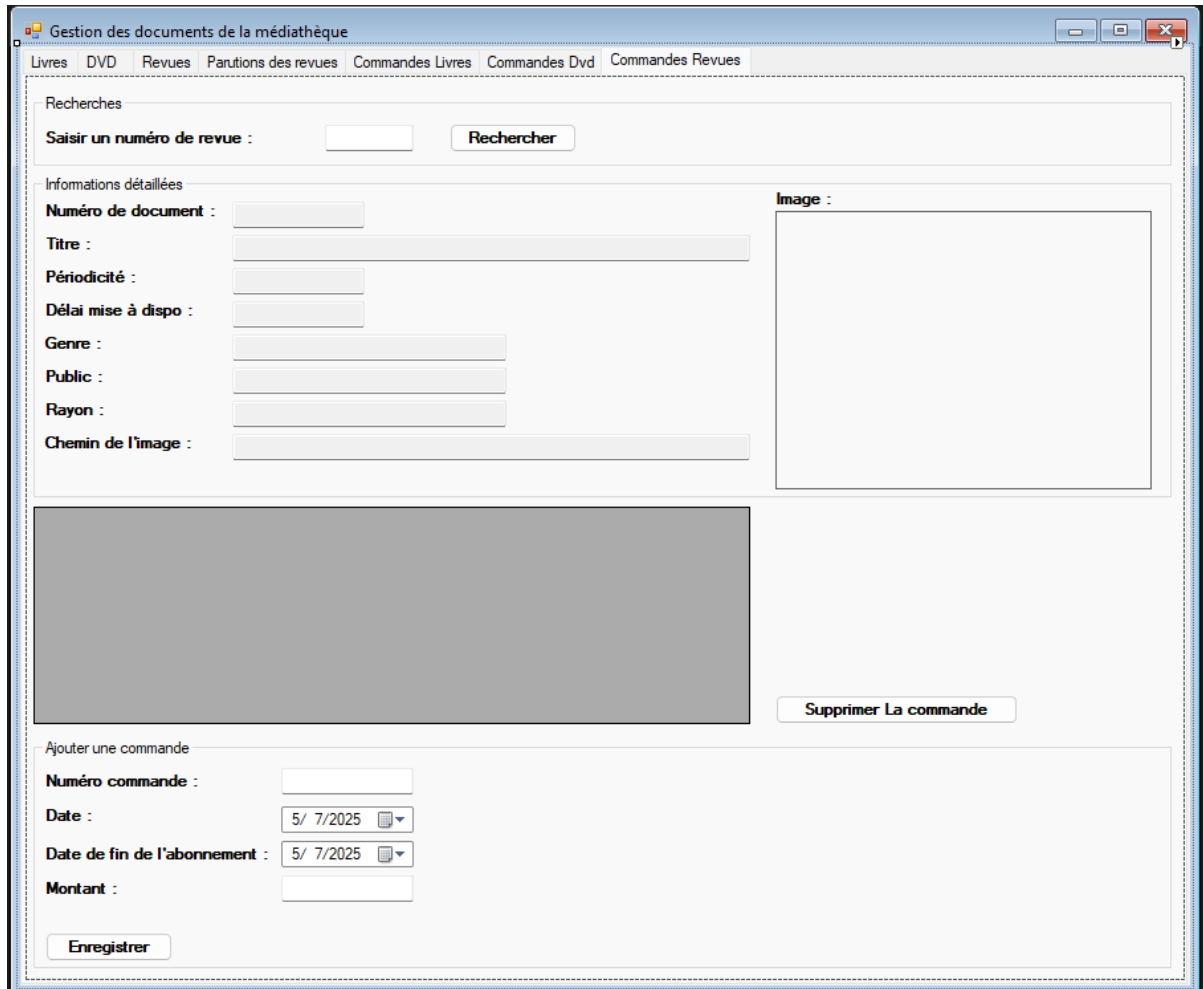


Diagramme de cas d'utilisation :



J'ai d'abord commencé par modifier l'interface en ajoutant l'onglet "Commandes Revues". Dans cet onglet il y a un datagridview permettant d'afficher la liste des commandes associées à une revue après recherche du numéro.

Il y a également une section réservée à l'ajout d'une commande et un bouton supprimer :



Dans la classe Access.cs j'ai ajouté la fonction "CreerAbonnementRevue" qui crée un abonnement(commande) pour une revue et j'ai ajouté la fonction intermédiaire dans le contrôleur :

```
public bool CreerAbonnementRevue(Abonnement abonnement)
```

Dans la classe MyAccessBDD.php de l'API REST j'ai modifié la fonction insertAbonnement et deleteAbonnement pour qu'elle effectue une transaction en ajoutant/supprimant également un tuple dans la table commande (pour l'héritage). Cela permet de faire un rollback et annuler les ajout/suppression si l'insert/delete échoue.

Ensuite j'ai ajouté la fonction événementielle "btnEnregCmdRevue_Click" qui appelle la fonction "CreerAbonnementRevue" du contrôleur en passant en paramètre un objet abonnement qui a été créé via les champs renseignés :

```
1 reference  
private void btnEnregCmdRevue_Click(object sender, EventArgs e)  
{
```

Pour le bouton supprimer j'ai ajouté la fonction événementielle "btnSupprCmdRevue_Click" qui supprime l'abonnement sélectionné s'il n'est rattaché à aucun exemplaire à l'aide de la fonction "isAbonnementRattaché_a_Exemplaire" :

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

```
private bool isAbonnementRattaché_a_Exemplaire(Abonnement abonnement)
```

La fonction "isAbonnementRattaché_a_Exemplaire" récupère les exemplaires liés à cet abonnement et vérifie si la date de parution de l'exemplaire est comprise entre la date de début et de fin de l'abonnement.

MISSION 3 Gérer le suivi de l'état des exemplaires

Cette mission était une mission facultative pour les élèves travaillant de manière individuelle, ce qui était mon cas.

MISSION 4 Mettre en place des authentifications

Todo 9 This item hasn't been started

- MediaTekDocuments #5
Tâche 3.1 : gérer le suivi de l'état des exemplaires
- MediaTekDocuments #7
Tâche 5.1 : corriger des problèmes de sécurité
- MediaTekDocuments #8
Tâche 5.2 : contrôler la qualité
- MediaTekDocuments #9
Tâche 5.3 : intégrer des logs
- MediaTekDocuments #10
Tâche 6.1 : gérer les tests
- MediaTekDocuments #11
Tâche 6.2 : créer les documentations techniques
- MediaTekDocuments #12
Tâche 6.3 : créer les documentations utilisateur en vidéo
- MediaTekDocuments #13

+ Add item

In Progress 1 This is actively being worked on

- MediaTekDocuments #6
Tâche 4.1 : mettre en place des authentifications

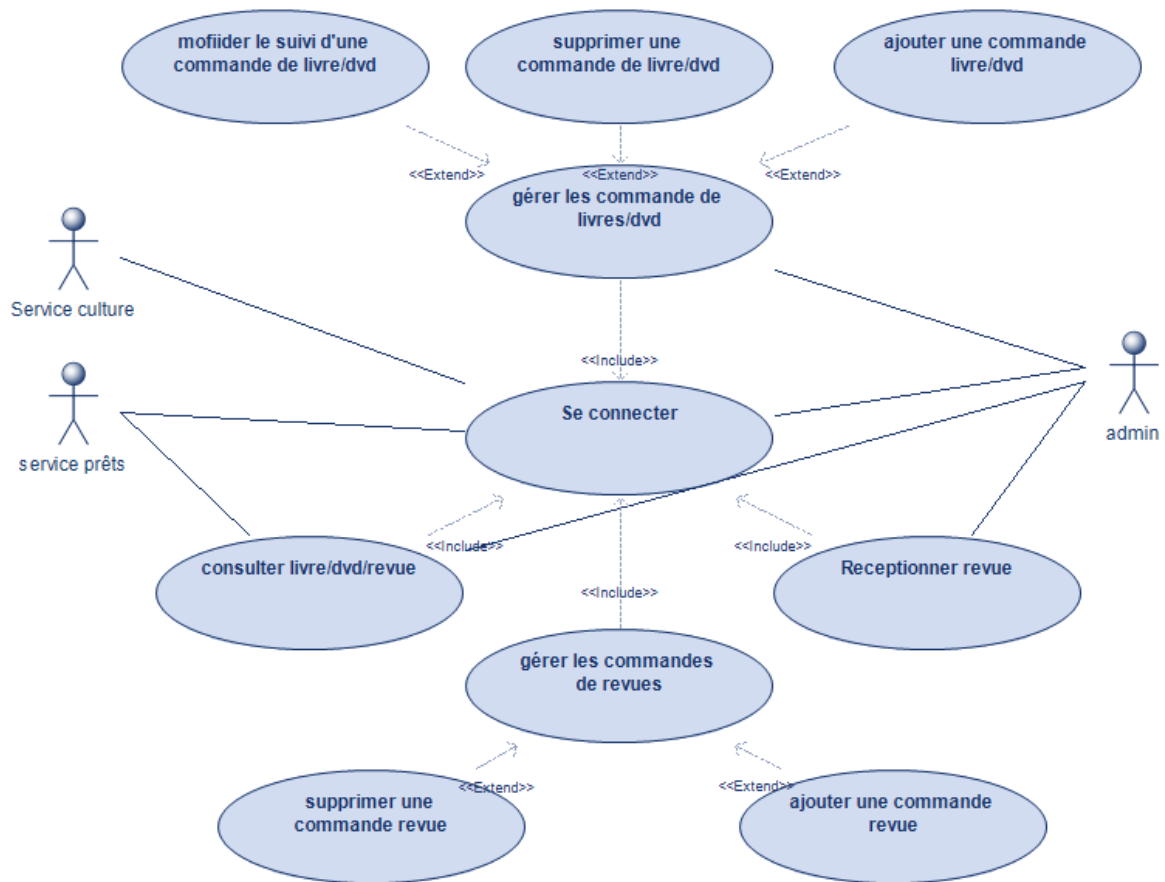
+ Add item

Done 3 This has been completed

- MediaTekDocuments #2
Tâche 1.1 : gérer les documents
- MediaTekDocuments #3
Tâche 2.1 : gérer les commandes de livres ou de DVD
- MediaTekDocuments #4
Tâche 2.2 : gérer les commandes de revues

+ Add item

Diagramme de cas d'utilisation :



A : Gérer les utilisateurs et les services

J'ai ajouté dans la BDD la table user avec quel exemple et en hachant les mots de passe, ainsi que la table service :

```
use mediatek86;

CREATE TABLE IF NOT EXISTS utilisateur (
  id INT PRIMARY KEY AUTO_INCREMENT,
  login varchar(50),
  pwd varchar(64),
  idService INT
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE IF NOT EXISTS service (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nom varchar(50)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

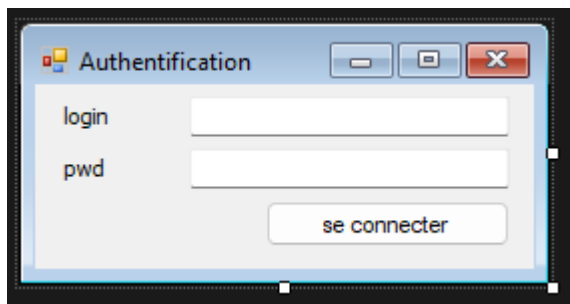
ALTER TABLE utilisateur
  ADD CONSTRAINT utilisateur_ibfk_1 FOREIGN KEY (idService) REFERENCES service(id);

INSERT INTO service (nom)
VALUES
  ('administratif'),
  ('prêts'),
  ('culture');

SELECT * FROM utilisateur;

INSERT INTO utilisateur (login, pwd, idService)
VALUES
  ('Eric', SHA2('Eric', 256), 1),
  ('Kyle', SHA2('Kyle', 256), 2),
  ('Stan', SHA2('Stan', 256), 2),
  ('Butters', SHA2('Butters', 256), 3);
```

Ensuite j'ai ajouté un nouveau formulaire d'authentification "FrmAuthentification.cs" avec son propre controller "FrmAuthentificationController.cs" :

A screenshot of a Windows-style application window titled "Authentification". The window has a standard title bar with minimize, maximize, and close buttons. Inside the window, there are two text input fields. The first field is labeled "login" and the second field is labeled "pwd". Below these fields is a button labeled "se connecter". The window has a light blue border and a white background.

Dans la classe Acces.cs, j'ai ajouté la fonction "GetUtilisateur" qui récupère un utilisateur par rapport au login et pwd renseigné :

```
1 reference
public Utilisateur GetUtilisateur(string login, string pwd)
{
```

Dans la classe MyAccessBDD.php de l'api rest j'ai ajouté la fonction selectUtilisateur qui récupère un utilisateur en hachant le pwd récupérer dans les champs et en le comparant à celui enregistré en bdd :

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

Ensuite dans la classe "FrmAuthentification.cs", j'ai ajouté la fonction événementielle btnConnect_Click pour le clic du bouton connecter, qui vérifie que les champs sont renseignés et ouvre le formulaire "FrmMediatek" si l'utilisateur est correctement authentifié et qu'il n'appartient pas au service culture :

```
1 reference
private void btnConnect_Click(object sender, EventArgs e)
{
```

Aussi, dans la fonction "Load" du formulaire FrmMediatek, j'ai ajouté une condition qui vérifie si l'utilisateur est un membre du service admin et bloque certaine fonctionnalité dans le cas contraire.

Dans la classe "Program.cs", j'ai remplacé dans la méthode run le formulaire de l'application par le formulaire d'authentification :

```
Application.Run(new FrmAuthentification());
```

B : Gérer les alertes de fin d'abonnement

Dans la classe MyAccessBDD.php de l'API REST, j'ai ajouté la fonction selectAllAbonnementsFin qui récupère tous les abonnements qui arrivent à échéance dans moins de 30 jours :

```
private function selectAllAbonnementsFin() : ?array {
```

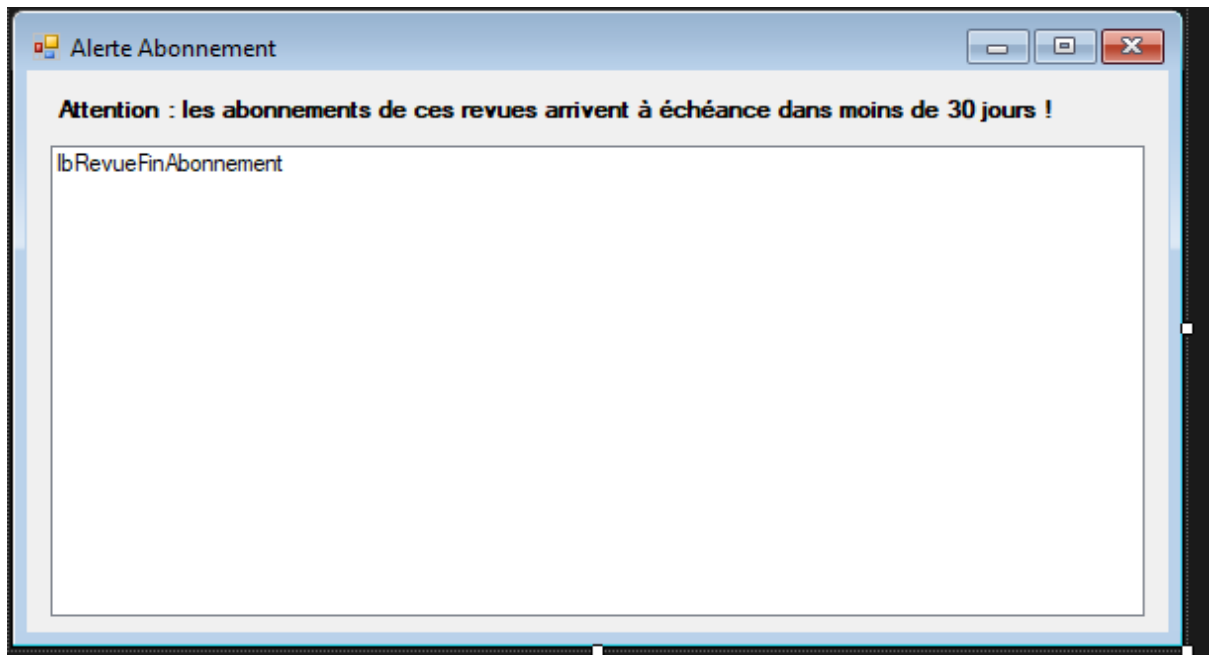
Après ça, j'ai ajouté dans la classe Acces.cs la fonction " GetAllAbonnementsFin" qui récupère tous les abonnements arrivent à échéance, toujours avec la fonction intermédiaire dans le contrôleur :

```
1 reference
public List<Abonnement> GetAllAbonnementsFin()
{
```

Ensuite, dans j'ai créé la classe FrmAlerteAbonnement.cs qui prend en paramètre (dans son constructeur) une liste d'abonnement :

```
1 reference
public FrmAlerteAbonnement(List<Abonnement> abonnements)
```

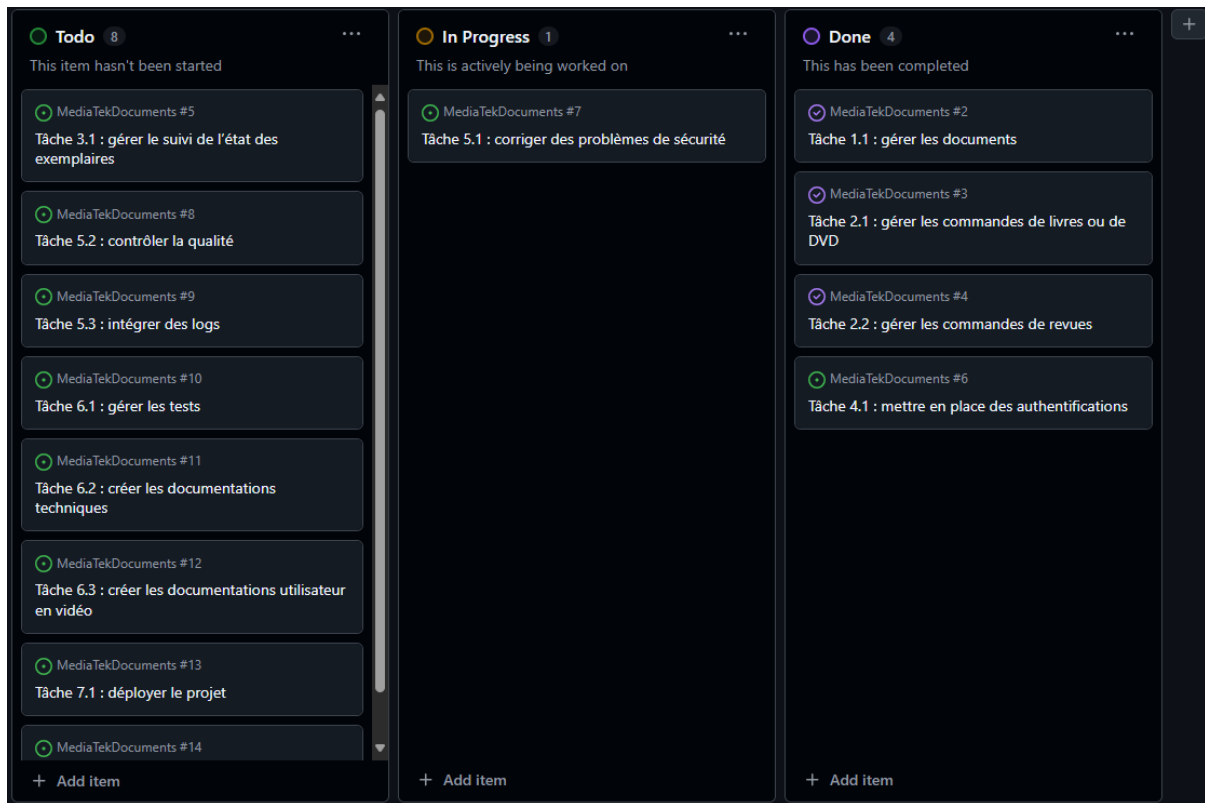
Et affiche ces abonnements dans une liste box :



Enfin, dans la classe FrmMediatek.cs, j'ai modifié la fonction "Load" pour qu'elle vérifie si l'utilisateur appartient au service admin, et si c'est le cas le formulaire d'alerte est affiché.

MISSION 5 assurer la sécurité, la qualité et intégrer des logs

Tâche 1 : corriger des problèmes de sécurité



Le premier problème de sécurité est dû au fait que 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 cela, j'ai transféré les informations sensibles dans le fichier App.config dans la variable <connectionstring>

Après ça j'ai ajouté la fonction "GetConnectionStringByName" qui récupère cette la chaîne d'authentification qui se trouve désormais dans le fichier App.config :

```
private static string GetConnectionStringByName(string name)
```

```
authenticationString = GetConnectionStringByName(connectionName);  
api = ApiRest.GetInstance(uriApi, authenticationString);
```

Le second problème résulte du fait que 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, alors que le but est d'avoir un retour d'erreur.

Pour corriger cela, j'ai ajouté dans le fichier `.htaccess` de l'API une règle de réécriture pour prendre en compte les url qui sont passées sans paramètres. Cela redirige vers une page qui affiche l'erreur 404 :

```
RewriteCond %{REQUEST_URI} ^/rest_mediatekdocuments/?$
RewriteRule ^ - [R=400,L]
```

Tâche 2 : contrôler la qualité

The screenshot shows a Kanban board with three columns: **Todo** (7 items), **In Progress** (1 item), and **Done** (5 items). The **In Progress** column is highlighted with a blue border.

- Todo (7 items):**
 - MediaTekDocuments #5
Tâche 3.1 : gérer le suivi de l'état des exemplaires
 - MediaTekDocuments #9
Tâche 5.3 : intégrer des logs
 - MediaTekDocuments #10
Tâche 6.1 : gérer les tests
 - MediaTekDocuments #11
Tâche 6.2 : créer les documentations techniques
 - MediaTekDocuments #12
Tâche 6.3 : créer les documentations utilisateur en vidéo
 - MediaTekDocuments #13
Tâche 7.1 : déployer le projet
 - MediaTekDocuments #14
Tâche 7.2 : gérer les sauvegardes des données
- In Progress (1 item):**
 - MediaTekDocuments #8 ***
Tâche 5.2 : contrôler la qualité
- Done (5 items):**
 - MediaTekDocuments #2
Tâche 1.1 : gérer les documents
 - MediaTekDocuments #3
Tâche 2.1 : gérer les commandes de livres ou de DVD
 - MediaTekDocuments #4
Tâche 2.2 : gérer les commandes de revues
 - MediaTekDocuments #6
Tâche 4.1 : mettre en place des authentifications
 - MediaTekDocuments #7
Tâche 5.1 : corriger des problèmes de sécurité

Après avoir configuré Sonarlint voici une partie de la liste des problèmes relevé sur mon projet :

The screenshot shows the SonarQube web interface. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and More. A warning banner at the top states: "Embedded database should be used for evaluation purposes only. It doesn't support scaling, upgrading to a new SonarQube Server version, or migration to another database engine. [Learn more](#)".

The main content area is titled "MediaTekDocuments" and shows a list of issues. The left sidebar contains filters for "My Issues" (All) and "Issues in new code". The "Software Quality" section shows 0 Security, 0 Reliability, and 40 Maintainability issues. The "Severity" section shows 0 Blocker issues.

The main list of issues includes:

- Refactor your code not to use hardcoded absolute paths or URIs.** (Maintainability, L22 - 20min effort - 7 days ago)
- Update this method so that its implementation is not identical to 'GetExemplesDocument'.** (Maintainability, L192 - 15min effort - 7 days ago)
- Define a constant instead of using this literal 'champs' 10 times.** (Maintainability, L257 - 4min effort - 7 days ago)

Tâche 2 : intégrer des logs

The screenshot shows a Kanban board with three columns: "Todo", "In Progress", and "Done". Each column has a header with a status icon and a count of items.

- Todo (6 items):** This item hasn't been started.
 - MediaTekDocuments #5: Tâche 3.1 : gérer le suivi de l'état des exemplaires
 - MediaTekDocuments #10: Tâche 6.1 : gérer les tests
 - MediaTekDocuments #11: Tâche 6.2 : créer les documentations techniques
 - MediaTekDocuments #12: Tâche 6.3 : créer les documentations utilisateur en vidéo
 - MediaTekDocuments #13: Tâche 7.1 : déployer le projet
 - MediaTekDocuments #14: Tâche 7.2 : gérer les sauvegardes des données
- In Progress (1 item):** This is actively being worked on.
 - MediaTekDocuments #9: Tâche 5.3 : intégrer des logs
- Done (6 items):** This has been completed.
 - MediaTekDocuments #2: Tâche 1.1 : gérer les documents
 - MediaTekDocuments #3: Tâche 2.1 : gérer les commandes de livres ou de DVD
 - MediaTekDocuments #4: Tâche 2.2 : gérer les commandes de revues
 - MediaTekDocuments #6: Tâche 4.1 : mettre en place des authentifications
 - MediaTekDocuments #7: Tâche 5.1 : corriger des problèmes de sécurité
 - MediaTekDocuments #8: Tâche 5.2 : contrôler la qualité

En utilisant le package “Serilog”, j’ai modifié le constructeur de la classe Access pour configurer les logs (console / fichier au format JSON à enregistrer dans un nouveau fichier chaque jour) :

```
private Access()
{
    String authenticationString = null;
    try
    {
        Log.Logger = new LoggerConfiguration()
            .MinimumLevel
            .Verbose()
            .WriteTo.Console()
            .WriteTo.File(new JsonFormatter(), "logs/log.txt", rollingInterval: RollingInterval.Day)
            .CreateLogger();

        authenticationString = GetConnectionStringByName(connectionName);
        api = ApiRest.GetInstance(uriApi, authenticationString);
    }
    catch (Exception e)
    {
        Log.Fatal("Access.Access catch authenticationString={0} erreur={1}", authenticationString, e.Message);
        Environment.Exit(0);
    }
}
```

Et voici un exemple d’une partie du code qui utilise un log :

```
public bool CreerLivre(Livre livre)
{
    String jsonLivre = JsonConvert.SerializeObject(livre);
    try
    {
        List<Livre> liste = TraitementRecup<Livre>(POST, "livre", "champs=" + jsonLivre);
        return (liste != null);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Log.Error("Access.CreerLivre catch jsonLivre={0} erreur={1}", jsonLivre, ex.Message);
    }

    return false;
}
```

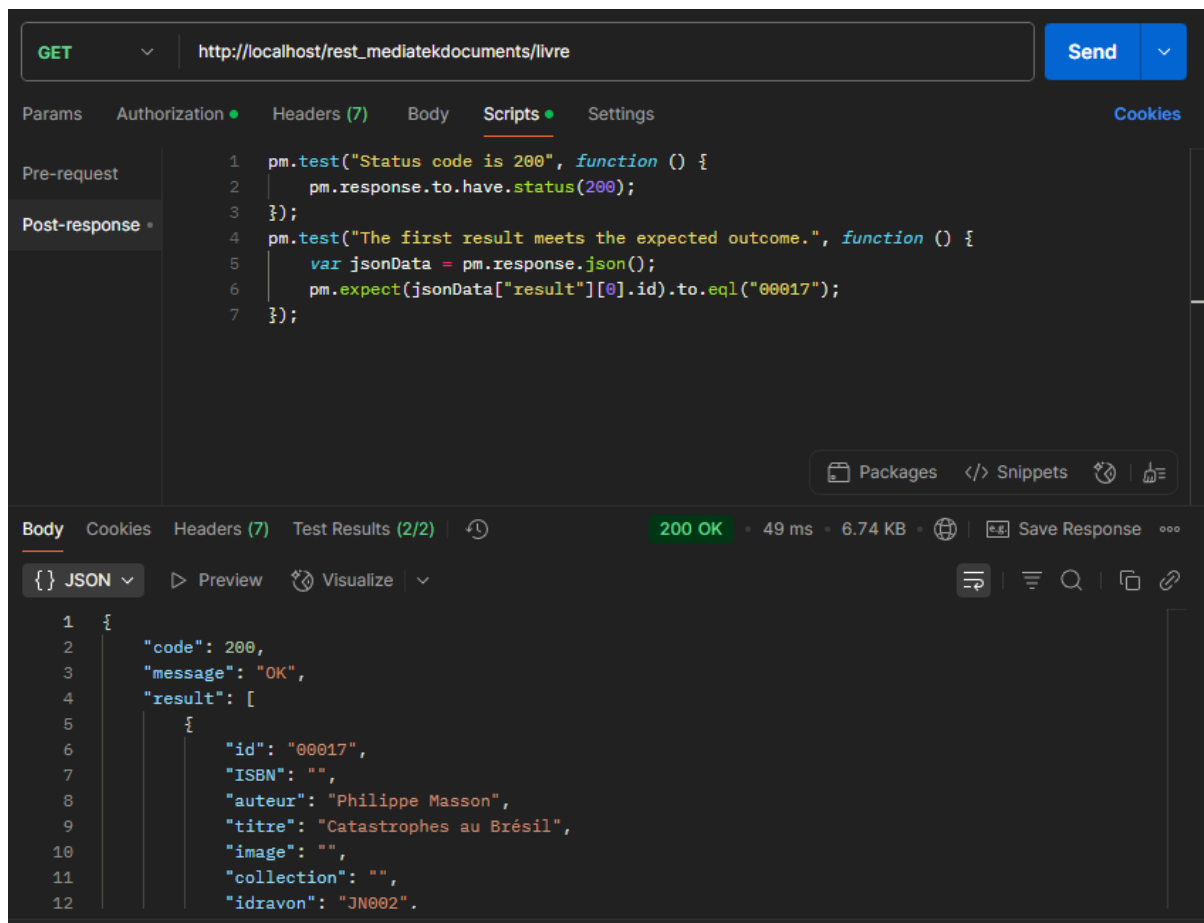

MISSION 6 tester et documenter

Tâche 1 : gérer les tests

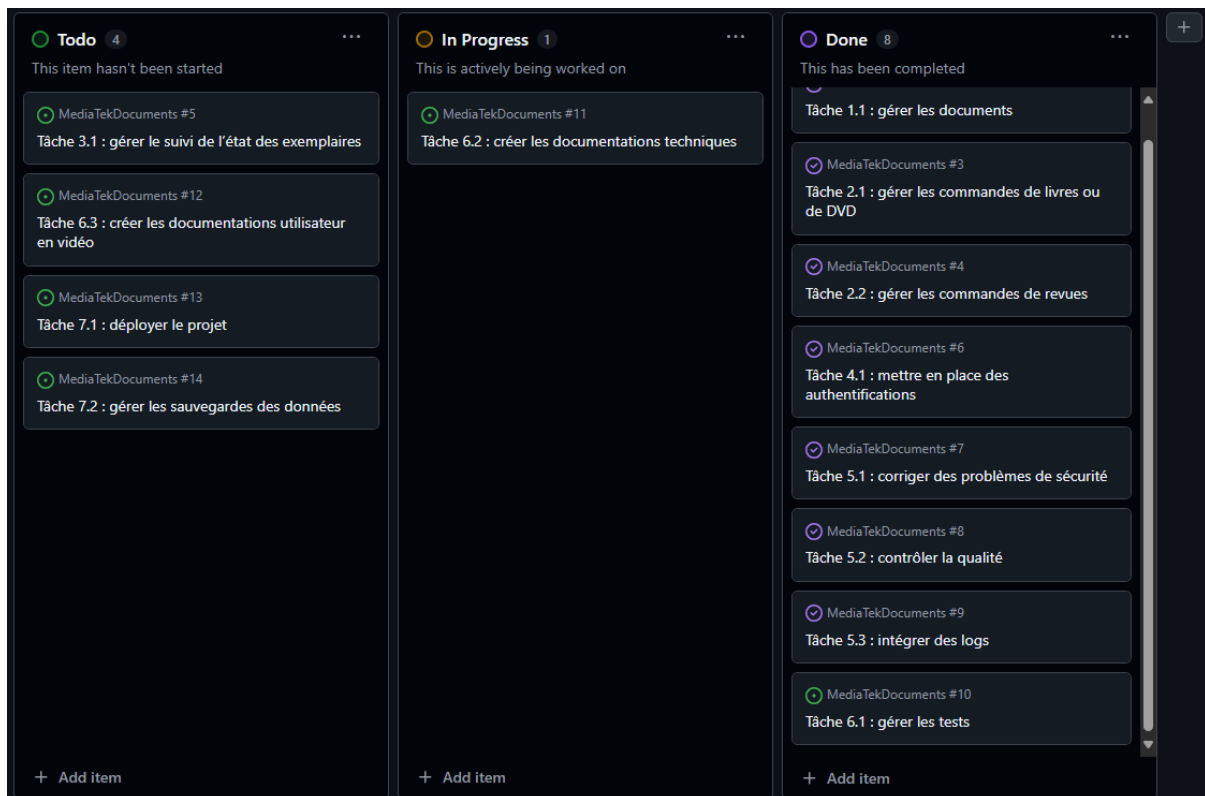
The image shows a Kanban board with three columns: **Todo** (5 items), **In Progress** (1 item), and **Done** (7 items). Each column has a description at the top and a '+ Add item' button at the bottom.

- Todo Column:**
 - MediaTekDocuments #5
Tâche 3.1 : gérer le suivi de l'état des exemplaires
 - MediaTekDocuments #11
Tâche 6.2 : créer les documentations techniques
 - MediaTekDocuments #12
Tâche 6.3 : créer les documentations utilisateur en vidéo
 - MediaTekDocuments #13
Tâche 7.1 : déployer le projet
 - MediaTekDocuments #14
Tâche 7.2 : gérer les sauvegardes des données
- In Progress Column:**
 - MediaTekDocuments #10
Tâche 6.1 : gérer les tests
- Done Column:**
 - MediaTekDocuments #2
Tâche 1.1 : gérer les documents
 - MediaTekDocuments #3
Tâche 2.1 : gérer les commandes de livres ou de DVD
 - MediaTekDocuments #4
Tâche 2.2 : gérer les commandes de revues
 - MediaTekDocuments #6
Tâche 4.1 : mettre en place des authentifications
 - MediaTekDocuments #7
Tâche 5.1 : corriger des problèmes de sécurité
 - MediaTekDocuments #8
Tâche 5.2 : contrôler la qualité
 - MediaTekDocuments #9
Tâche 5.3 : intégrer des logs

J'ai intégré des tests fonctionnels via Postman pour tous les select de l'API REST, voici un exemple :



Tâche 2 : créer les documentations techniques




Pour cette tâche chez ajouter des commentaires normalisés pour chaque fonctions et classe voici un exemple pour l'application :

```
/// <summary>
/// Retourne vrai si la date de parution est comprise entre deux dates
/// </summary>
/// <param name="dateCommande"></param>
/// <param name="dateFinAbo"></param>
/// <param name="dateParution"></param>
/// <returns></returns>
1 reference
private bool ParutionDansAbonnement(DateTime dateCommande, DateTime dateFinAbo, DateTime dateParution)
```

Et pour l'API REST :

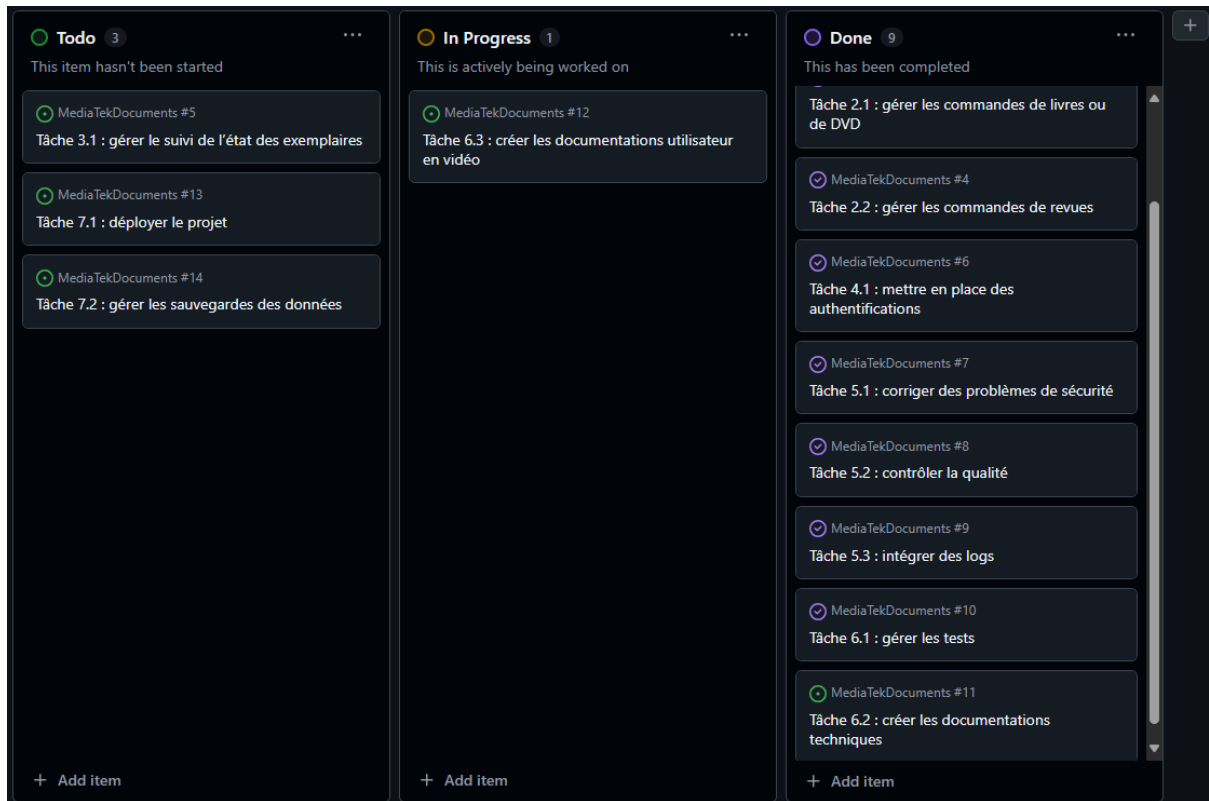
```
/**
 * récupère tous les abonemnts arrivant bientôt à échéance
 * @return array|null
 */
private function selectAllAbonnementsFin() : ?array {
```

Et voilà ce à quoi cela ressemble dans la documentation technique de l'api :

 selectAllAbonnementsFin() : array<string|int, mixed> | null
récupère tous les abonemnts arrivant bientôt à échéance

Et la documentation technique de l'application :

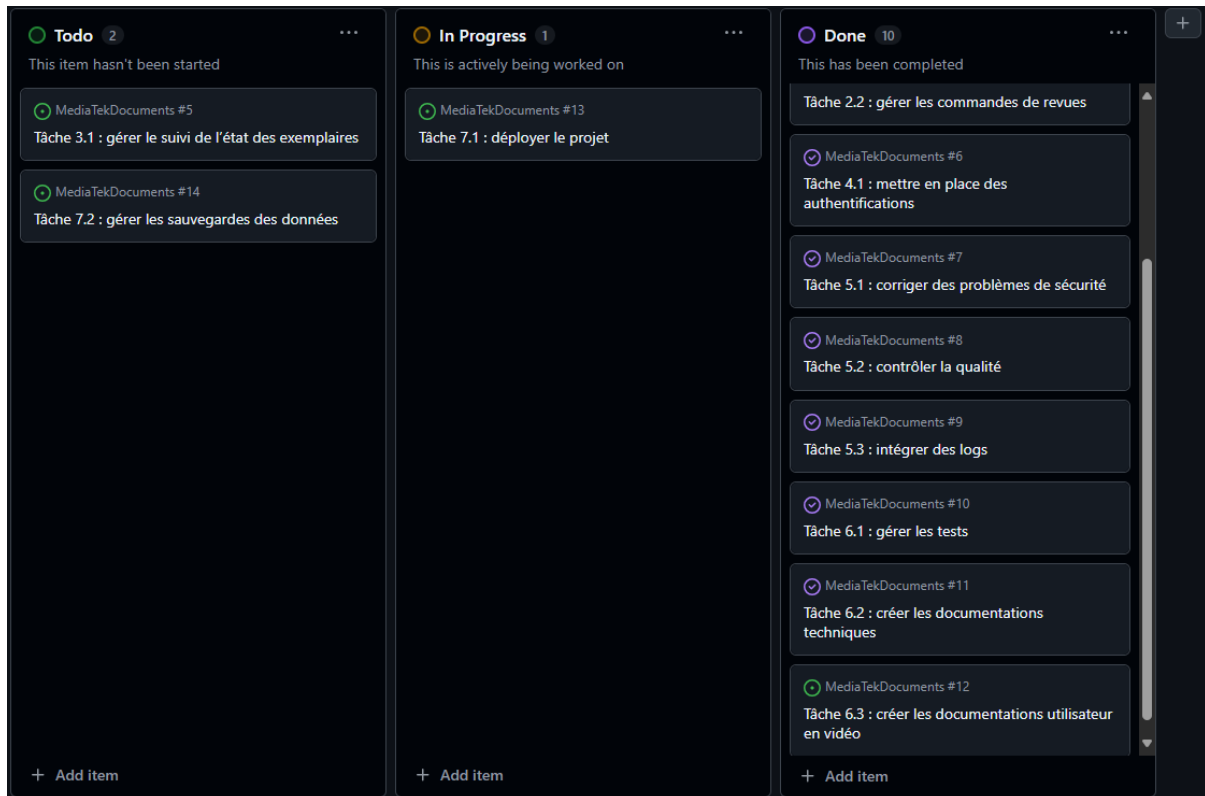
Tâche 2 : créer la documentation utilisateur en vidéo



Pour cette tâche chez réalisé une vidéo utilisateur de 10 min pour expliquer chaque fonctionnalité de l'application à l'aide du logiciel OBS studio.

MISSION 7 tester et documenter

Tâche 1 : déployer le projet

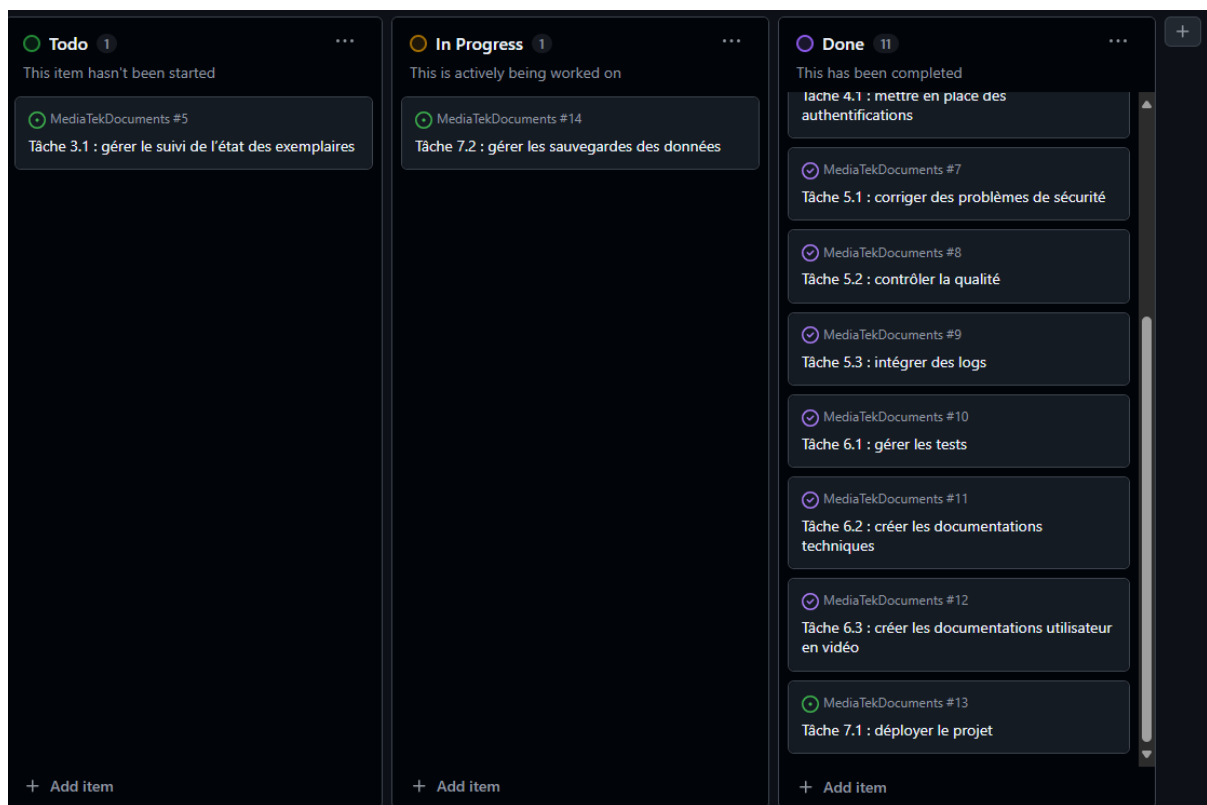


Pour déployer l'api en ligne j'ai pris un abonnement premium chez l'hébergeur Hostinger. J'ai utilisé WinSCP pour transférer les fichiers de l'API.

Ensuite j'ai importé le script SQL sur le phpMyAdmin de l'hébergeur, et à la suite de cela j'ai modifié les informations de la bdd dans le fichiers .env avec celle de l'hébergeur (login bdd, pwd bdd, nom de la bdd, le serveur et le port) et j'ai également changé les identifiant d'authentifications. Pour accéder à l'api j'ai également modifié la chaine d'authenfication dans le fichier App.config et l'url de l'API dans la classe Access.cs

Concernant l'application, j'ai créé un installateur. Pour ce faire j'ai créé un projet MediateDocuemntSetUp (Setup Wizard) que j'ai générer. J'ai également ajouté une icône pour l'application. Après cela il suffit d'exécuter MediateDocuemntSetUp et l'application s'installe.

Tâche 1 : gérer les sauvegardes des données



Pour cette tâche chez créer une script bash qui récupère le scrip sql de la bdd et l'enregistre dans le dossier savebdd(en racine de l'api). J'ai transféré le script chez l'hébergeur et j'ai créé j'ai planifié une tâche CRON quotidienne pour exécuter le script bash.

Pour restaurer les données, il suffit alors de récupérer le fichier sql enregistrer dans le dossier savedd et de l'exécuter dans le PhpMyAdmin de l'hébergeur.

BILAN

Toutes les missions de ce projet ont été accomplies excepté la mission 3 qui était une mission facultative pour les élèves individuels. Certains problèmes ont été rencontrés comme l'exécution du fichier bash pour la sauvegarde des données ou encore l'implémentation des transactions, mais j'ai réussi à tous les résoudre. Pour conclure, je dirais que ce projet m'a permis de mettre en pratique mes compétences et de les renforcer.