

**SOMMAIRE**

<b>Présentation.....</b>	<b>1</b>
<b>Contexte.....</b>	<b>1</b>
<b>Existant.....</b>	<b>1</b>
<b>Langages.....</b>	<b>1</b>
<b>Technologies utilisées.....</b>	<b>1</b>
<b>Mission 1 : gérer les documents.....</b>	<b>2</b>
<b>Mission 2 : gérer les commandes.....</b>	<b>13</b>
Tâche 1.....	15
Tâche 2.....	30
<b>Mission 3 : gérer le suivi de l'état des exemplaires.....</b>	<b>36</b>
<b>Mission 4 : mettre en place les authentifications.....</b>	<b>45</b>
<b>Mission 5 : assurer la sécurité, la qualité et intégrer des logs.....</b>	<b>52</b>
Tâche 1.....	54
Tâche 2.....	56
Tâche 3.....	57
<b>Mission 6 : tester et documenter.....</b>	<b>58</b>
Tâche 1.....	59
Tâche 2.....	64
Tâche 3.....	66
<b>Mission 7 : déployer et gérer les sauvegardes de données.....</b>	<b>67</b>
Tâche 1.....	68
Tâche 2.....	69
<b>Bilan professionnalisation n°3.....</b>	<b>74</b>

# Présentation

## Contexte

Travaillant en tant que technicien développeur junior pour l'ESN InfoTech Services 86, ce travail portera sur le développement de l'application bureau pour les médiathèques du réseau "Mediatek86" afin de permettre la gestion des documents (livres, DVD et revues) de celles-ci. Les personnels cibles de l'application sont les employés des services "administratifs" et "prêts".

## Existant

Un premier développeur s'est occupé de la construction de la base de données et du développement de certaines fonctionnalités de l'application. C'est une application de bureau, prévue d'être installée sur plusieurs postes de la médiathèque et qui accèdent à la même base de données.

L'application exploite une API REST afin d'accéder à la BDD MySQL, codée partiellement également.

## Langages

Langage de développement applicatif :	C#
Langage de développement REST :	PHP
Langage d'exploitation du SGBD :	SQL

## Technologies utilisées

Serveur local:	WampServer
Système de gestion de base de données :	MySQL
Application Web de gestion SBGB :	phpMyAdmin
Logiciel de gestion de versions :	Git
Service d'hébergement du développement :	GitHub
Hébergeur Web :	OVHCloud.
Client FTP :	FileZilla.
Application de test API :	Postman.

Environnement de développement intégré :

- Visual Studio Code 2019 Community pour l'application bureau C#.
- Apache Netbeans IDE 12.0 pour l'API REST.

# Production

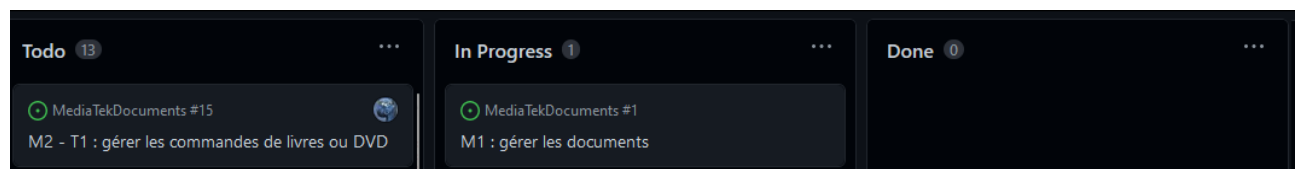
## Mission 1 : gérer les documents

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

Temps de travail estimé  
réel  
8 heures

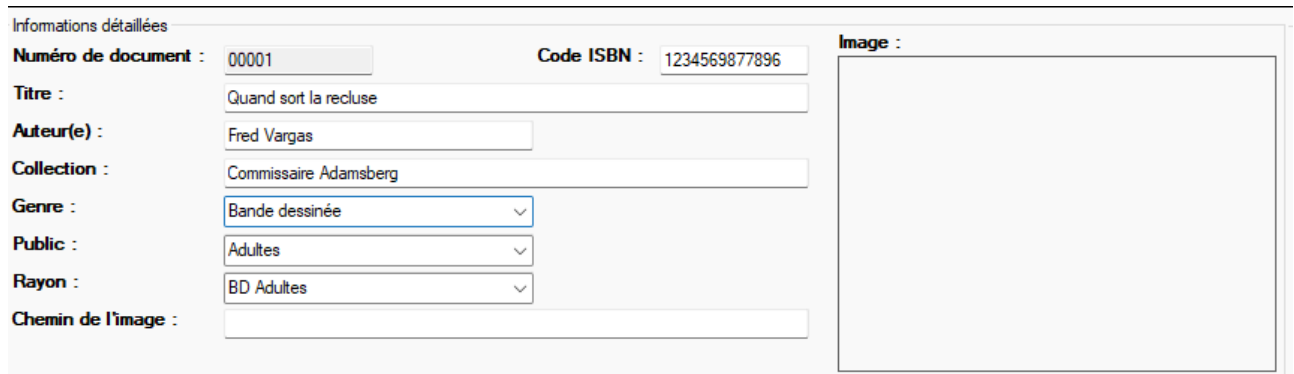
Temps de travail  
  
20 heures

### Kanban de la tâche actuelle "In Progress"



## Processus

Afin de pouvoir ajouter, modifier ou supprimer un document, de nombreuses modifications et ajouts ont été ajoutés dans la classe *FrmMediatek.cs*. Il a fallu notamment récupérer des valeurs qui ne sont pas disponibles dans le code d'origine. Celles-ci correspondent aux idGenre, idRayon et idPublic. Pour ce faire, nous avons ajouté trois nouveaux combobox dans les onglets Livre, Dvd et Revues.



Ceux-ci vont nous permettre de récupérer l'id de chaque champ cité via une fonction d'écoute d'évènement sur le changement de sélection. On crée également un champ caché qui permet de sauvegarder l'id récupéré.

```
private void CbxLivresPublicEdit_SelectedIndexChanged(object sender, EventArgs e)
{
    if (cbxLivresPublicEdit.SelectedIndex >= 0)
    {
        Public lePublic = (Public)cbxLivresPublicEdit.SelectedItem;
        txbLivresIdPublic.Text = lePublic.Id;
    }
}
```

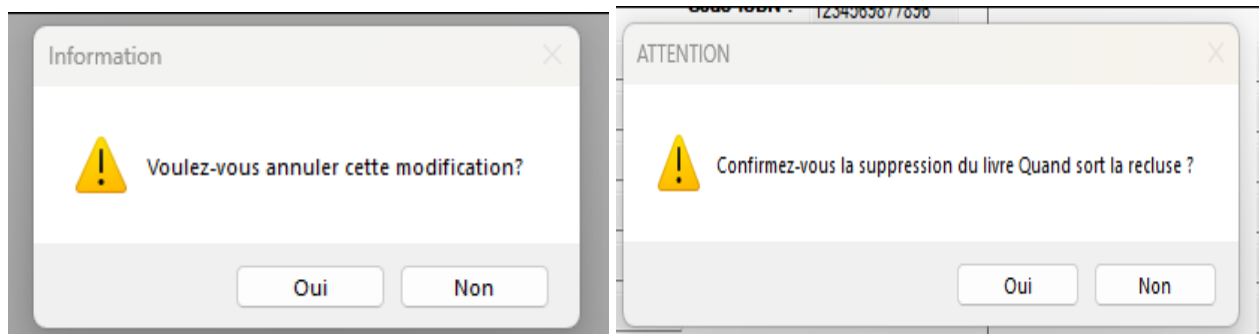
La fonction suivante dans le cas de l'ajout d'un livre en base de données :

```
private void BtnValiderAjoutLivre_Click(object sender, EventArgs e)
{
    if (!txbLivresNumero.Text.Equals("") && cbxLivresGenreEdit.SelectedIndex != -1 && cbxLivresPublicEdit.SelectedIndex != -1 && cbxLi
    {
        try
        {
            string id = txbLivresNumero.Text;
            string titre = txbLivresTitre.Text;
            string image = txbLivresImage.Text;
            string isbn = txbLivresIsbn.Text;
            string auteur = txbLivresAuteur.Text;
            string collection = txbLivresCollection.Text;
            string idGenre = txbLivresIdGenre.Text;
            string genre = cbxLivresGenreEdit.SelectedItem.ToString();
            string idPublic = txbLivresIdPublic.Text;
            string lePublic = cbxLivresPublicEdit.SelectedItem.ToString();
            string idRayon = txbLivresIdRayon.Text;
            string rayon = cbxLivresRayonEdit.SelectedItem.ToString();
            Livre livre = new Livre(id, titre, image, isbn, auteur, collection, idGenre, genre, idPublic, lePublic, idRayon, rayon);
            if (controller.CreerLivre(livre))
            {
                MessageBox.Show("Le livre " + titre + " vient d'être ajouté.");
            }
        }
    }
}
```

Les autres valeurs n'ont pas posé de problème car déjà initialisées et récupérables.

Des boutons ajouter, modifier et supprimer ont été ajoutés dans chaque onglet, ainsi que des boutons optionnels afin de valider ou annuler l'action en cours.

Différents champs de vérification permettent de vérifier que l'utilisateur soit bien conscient des actions qu'il effectue sur l'application. Par exemple :



Plusieurs fonctions permettant une meilleure expérience utilisateur ont été ajoutées afin d'afficher les éléments liés à l'action demandée. Par exemple dans l'onglet Livre, afin d'afficher les boutons nécessaires et informations nécessaires à la modification d'un livre :

```

/// <summary>
/// Active les champs nécessaires à la modification de l'état d'un exemplaire livre
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void BtnModifierExemplairesLivre_Click(object sender, EventArgs e)
{
    // Liste des contrôles à afficher
    Control[] show = { btnModifierExemplairesLivreOk, btnAnnulerExemplairesLivre };
    foreach (Control control in show)
    {
        control.Show();
    }
    // Liste des contrôles à masquer
    Control[] hide = { btnSupprimerExemplairesLivre, btnModifierExemplairesLivre };
    foreach (Control control in hide)
    {
        control.Hide();
    }
    cbxExemplairesLivreEtat.Enabled = true;
}

```

Mais aussi afin de restaurer l'onglet Livres après une action d'édition :

```
/// <summary>
/// Restaure l'onglet Livres à son état initial
/// </summary>
7 références
private void RestaurationConfigLivres()
{
    // Activer le lien avec le grid
    DefaultLivre = true;
    // Liste des contrôles à masquer
    Control[] hide = { txbLivresIdGenre, txbLivresIdPublic, txbLivresIdRayon,
        lblLivresIdGenre, lblLivresIdPublic, lblLivresIdRayon,
        cbxLivresGenreEdit, cbxLivresPublicEdit, cbxLivresRayonEdit,
        btnAnnulerLivre, btnValiderAjouterLivre, btnAnnulerExemplairesLivre, btnModifierExemplairesLivreOk
    };
    foreach (Control control in hide)
    {
        control.Hide();
    }

    // Liste des contrôles à afficher
    Control[] show = {
        btnAjouterLivre, btnModifierLivre, btnSupprimerLivre,
        btnModifierExemplairesLivre, btnSupprimerExemplairesLivre,
        cbxLivresGenres, cbxLivresPublics, cbxLivresRayons
    };
    foreach (Control control in show)
    {
        control.Show();
    }
    DesactiverChampsInfosLivre();
}
```

Le controller *FrmMediatekController.cs* a été codé par toutes les méthodes liées à la création, modification et suppression dans le cas d'un livre, dvd ou revue. Un extrait de ces ajouts :

```
/// <summary>
/// Crée un DVD dans la BDD
/// </summary>
/// <param name="dvd">l'objet DVD concerné</param>
/// <returns>true si la création a pu se faire</returns>
1 référence
public bool CreerDvd(Dvd dvd) => access.CreerDvd(dvd);
```

```
/// <summary>
/// Modifie un livre dans la BDD
/// </summary>
/// <param name="livre">l'objet Livre concerné</param>
/// <returns>true si la modification a pu se faire</returns>
public bool ModifierLivre(Livre livre) => access.ModifierLivre(livre);
```

```
/// <summary>
/// Supprimer une revue dans la BDD
/// </summary>
/// <param name="revue">l'objet revue concerné</param>
/// <returns>true si la suppression a pu se faire</returns>
1 référence
public bool SupprimerRevue(Revue revue) => access.SupprimerRevue(revue);
```

Et de ce fait dans la classe `Access.cs`. Nous sérialisons la requête au format json afin qu'elle puisse être lue par l'api (que nous allons voir à la suite de ces extraits) ainsi que la fonction `TraitementRecup` qui formalise l'envoi de la requête vers l'api :

```
private List<I> TraitementRecup<I>(String methode, String message)
{
    List<T> liste = new List<T>();
    try
    {
        JObject retour = api.RecupDistant(methode, message);
        // extraction du code retourné
        String code = (String)retour["code"];
        if (code.Equals("200"))
        {
            // dans le cas du GET (select), récupération de la liste d'objets
            if (methode.Equals(GET))
            {
                String resultString = JsonConvert.SerializeObject(retour["result"]);
                // construction de la liste d'objets à partir du retour de l'api
                liste = JsonConvert.DeserializeObject<List<T>>(resultString, new CustomBooleanJsonConverter());
            }
        }
    }
}
```

Un exemple des méthodes `Access.cs` qui seront utilisées par la vue via le controller afin d'afficher la demande utilisateur :

```
public bool CreerRevue(Revue revue)
{
    String jsonRevue = JsonConvert.SerializeObject(revue);
    try
    {
        // récupération soit d'une liste vide (requête ok) soit de null (erreur)
        List<Revue> liste = TraitementRecup<Revue>(POST, "revue/" + jsonRevue);
        return (liste != null);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return false;
}
```

```
public bool ModifierDvd(Dvd dvd)
{
    String jsonDvd = JsonConvert.SerializeObject(dvd);
    try
    {
        // récupération soit d'une liste vide (requête ok) soit de null (erreur)
        List<Dvd> liste = TraitementRecup<Dvd>(PUT, "dvd/" + dvd.Id + "/" + jsonDvd);
        return (liste != null);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return false;
}
```

Cet appel au controller est visible dans *FrmMediatek* dès qu'une action est effectuée, par exemple ici dans la création d'un nouveau livre :

```
string rayon = cbxLivresRayonEdit.SelectedItem.ToString();
Livre livre = new Livre(id, titre, image, isbn, auteur, collection, idGenre, genre, idPublic, lePublic, idRayon, rayon);
if (controller.CreerLivre(livre))
{
```

Du côté de l'api rest, dans la classe *AccessBDD.php*, afin de répondre à la demande côté client, la requête est écrite en respectant les champs reçus. Ici pour l'insertion (création) d'un nouveau livre par exemple :

```
public function insertLivre($champs) {
    // tableau associatif des données document
    $champsDocument = [
        "Id" => $champs["Id"],
        "Titre" => $champs["Titre"],
        "Image" => $champs["Image"],
        "IdRayon" => $champs["IdRayon"],
        "IdPublic" => $champs["IdPublic"],
        "IdGenre" => $champs["IdGenre"]
    ];
    $resultDocument = $this->insertSimple("document", $champsDocument);

    // tableau associatif des données livres_dvd
    $champsLivresDvd = ["Id" => $champs["Id"]];
    $resultLivresDvd = $this->insertSimple("livres_dvd", $champsLivresDvd);

    // tableau associatif des données livre
    $champsLivre = [
        "Id" => $champs["Id"],
        "Isbn" => $champs["Isbn"],
        "Auteur" => $champs["Auteur"],
        "Collection" => $champs["Collection"]
    ];
    $resultLivre = $this->insertSimple("livre", $champsLivre);

    return $resultDocument && $resultLivresDvd && $resultLivre;
}
```

On peut noter l'appel de *insertSimple* qui permet, à condition que les champs et que la connection soit établie, d'exécuter la requête envoyée en la reformulant :

(voir page suivante)



```

public function insertSimple($table, $champs) {
    if ($this->conn != null && $champs != null) {
        // construction de la requête
        $req = "insert into $table (";
        foreach ($champs as $key => $value) {
            $req .= "$key,";
        }
        // (enlève la dernière virgule)
        $req = substr($req, 0, strlen($req) - 1);
        $req .= ") values (";
        foreach ($champs as $key => $value) {
            $req .= ":$key,";
        }
        // (enlève la dernière virgule)
        $req = substr($req, 0, strlen($req) - 1);
        $req .= ");";
        return $this->conn->execute($req, $champs);
    } else {
        return null;
    }
}

```

La fonction *insertOne* récupère les fonctions d'insertion et assigne grâce à un switch, à chaque demande côté client, l'insert correspondant :

```

/**
 * Ajout d'une ligne dans une table
 * @param string $table nom de la table
 * @param array $champs nom et valeur de chaque champs de la ligne
 * @return true si l'ajout a fonctionné
 */
public function insertOne($table, $champs) {
    if ($this->conn != null && $champs != null) {
        switch ($table) {
            case "livre" :
                return $this->insertLivre($champs);
        }
    }
}

```

La classe *Controle.php* récupère la fonction afin de la traiter et la valider ou non, le cas échéant :

```

/**
 * Requête arrivée en POST (insert)
 * @param string $table nom de la table
 * @param array $champs nom et valeur des champs
 */
public function post($table, $champs) {
    $result = $this->accessBDD->insertOne($table, $champs);
    if ($result === null || $result === false) {
        $this->reponse(400, "requete invalide");
    } else {
        $this->reponse(200, "OK");
    }
}

```

Enfin, la classe *mediatekdocuments.php* formalise la demande vers la base de données en s'y connectant au préalable et en fonction du verbe HTTP utilisé :

```
// traitement suivant le verbe HTTP utilisé
if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    if ($contenu !== "") {
        $controle->getUtilisateur($contenu);
    } else {
        $controle->get($table, $id);
    }
} else if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $controle->post($table, $contenu);
} else if ($_SERVER['REQUEST_METHOD'] === 'PUT') {
    $controle->put($table, $id, $contenu);
} else if ($_SERVER['REQUEST_METHOD'] === 'DELETE') {
    $controle->delete($table, $contenu);
}
```

Trigger contrôlant la contrainte de partition de l'héritage sur Document :

- lors d'un insert de type *livres\_dvd*

```
1 DROP TRIGGER IF EXISTS insert_livresdvd_to_document_by_id;
2 DELIMITER //
3 CREATE TRIGGER insert_livresdvd_to_document_by_id BEFORE INSERT ON livres_dvd
4 FOR EACH ROW BEGIN
5     IF NOT EXISTS (SELECT id FROM document WHERE id = NEW.id) THEN
6         INSERT INTO document (id) VALUES (NEW.id);
7     END IF;
8 END //
9 DELIMITER ;
```

- lors d'un delete de type *livres\_dvd*

```
1 CREATE TRIGGER `delete_livresdvd_to_document_by_id` AFTER DELETE ON `livres_dvd`
2 FOR EACH ROW BEGIN
3     DELETE FROM document WHERE id = OLD.id;
4 END
```

- lors d'un insert de type *revue*

```
1 DROP TRIGGER IF EXISTS insert_revue_to_document_by_id;
2 DELIMITER //
3 CREATE TRIGGER insert_revue_to_document_by_id BEFORE INSERT ON revue
4 FOR EACH ROW BEGIN
5     IF NOT EXISTS (SELECT id FROM document WHERE id = NEW.id) THEN
6         INSERT INTO document (id) VALUES (NEW.id);
7     END IF;
8 END //
9 DELIMITER ;
```

- lors d'un delete de type *revue*

```
1 DROP TRIGGER IF EXISTS delete_revue_to_document_by_id;
2 DELIMITER //
3 CREATE TRIGGER delete_revue_to_document_by_id AFTER DELETE ON revue
4 FOR EACH ROW BEGIN
5     DELETE FROM document WHERE id = OLD.id;
6 END //
7 DELIMITER ;
```

Trigger contrôlant la contrainte de partition de l'héritage sur LivresDvd :

- lors d'un insert de type *livre*

```
1 DROP TRIGGER IF EXISTS insert_livre_to_livresdvd_by_id;
2 DELIMITER //
3 CREATE TRIGGER insert_livre_to_livresdvd_by_id BEFORE INSERT ON livre
4 FOR EACH ROW BEGIN
5     IF NOT EXISTS (SELECT id FROM livres_dvd WHERE id = NEW.id) THEN
6         INSERT INTO livres_dvd (id) VALUES (NEW.id);
7     END IF;
8 END //
9 DELIMITER ;
```

- lors d'un delete de type *livre*

```
1 CREATE TRIGGER `delete_livre_to_livresdvd_by_id` AFTER DELETE ON `livre`
2 FOR EACH ROW BEGIN
3     DELETE FROM livres_dvd WHERE id = OLD.id;
4 END
```

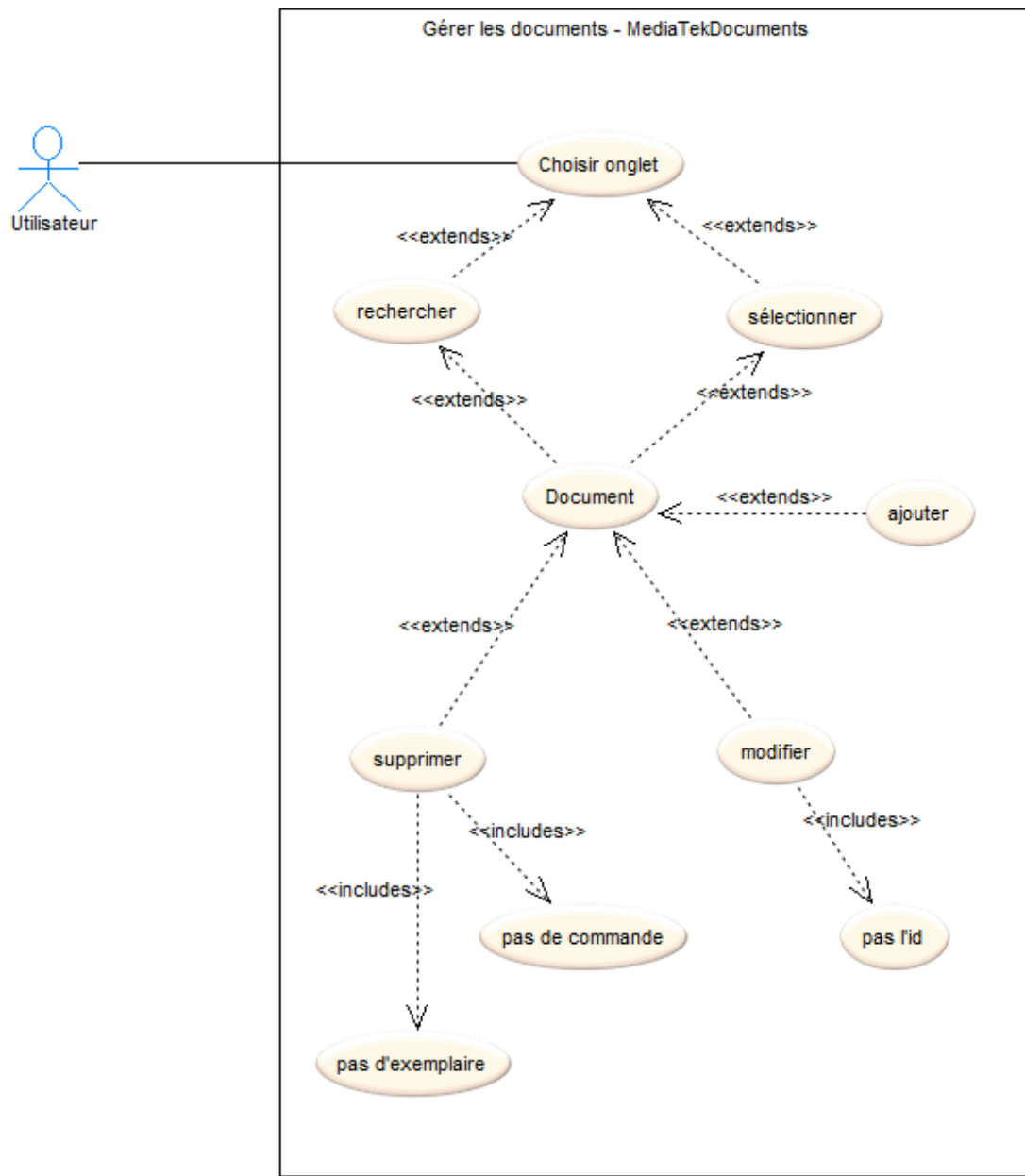
- lors d'un insert de type *dvd*

```
1 DROP TRIGGER IF EXISTS insert_dvd_to_livresdvd_by_id;
2 DELIMITER //
3 CREATE TRIGGER insert_dvd_to_livresdvd_by_id BEFORE INSERT ON dvd
4 FOR EACH ROW BEGIN
5     IF NOT EXISTS (SELECT id FROM livres_dvd WHERE id = NEW.id) THEN
6         INSERT INTO livres_dvd (id) VALUES (NEW.id);
7     END IF;
8 END //
9 DELIMITER ;
```

- lors d'un delete de type *dvd*

```
1 CREATE TRIGGER `delete_dvd_to_livresdvd_by_id` AFTER DELETE ON `dvd`
2 FOR EACH ROW BEGIN
3     DELETE FROM livres_dvd WHERE id = OLD.id;
4 END
```

## Diagramme de cas d'utilisation



## Mission 2 : gérer les commandes

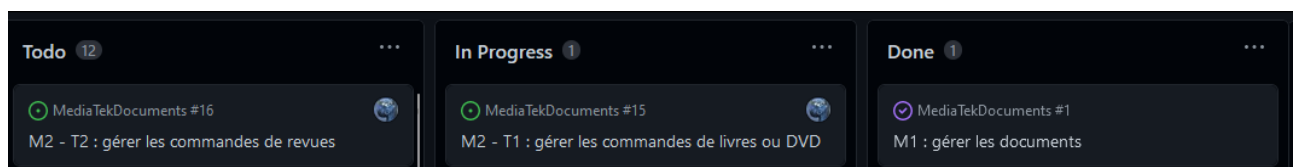
### Tâche 1 : gérer les commandes de livres ou de DVD

- Gérer les commandes de livres ou de DVD. Il doit être possible de commander un ou plusieurs exemplaires d'un livre ou d'un DVD et de suivre l'évolution d'une commande.
- L'application doit permettre de voir la liste des commandes et gérer le suivi.
- Lorsqu'une commande est livrée, il faut que les exemplaires concernés soient automatiquement générés dans la BDD avec un numéro séquentiel par rapport au document concerné.
- Une commande doit pouvoir être supprimée si elle n'est pas encore livrée.
- Créer le trigger qui supprime également une commande dans la classe fille.
- Créer le trigger qui contrôle la contrainte de partition de l'héritage sur Commande.
- Créer le trigger qui lors du passage d'une commande à l'étape "livrée", créer autant de tuples dans la table *exemplaires* que nécessaire, en valorisant la date d'achat avec la date de la commande et en mettant l'état de l'exemplaire à "neuf". Le numéro de l'exemplaire doit être séquentiel par rapport au livre concerné.

Temps de travail estimé  
réel  
8 heures

Temps de travail  
  
20 heures

### Kanban de la tâche actuelle "In Progress"



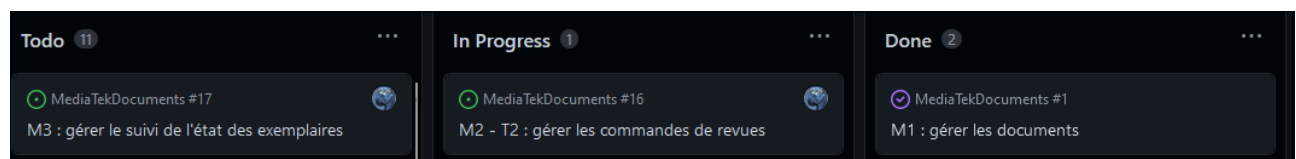
## Tâche 2 : gérer les commandes de revues

- Il doit être possible de commander une revue. Une commande de revue revient à réaliser un abonnement. L'application doit permettre de voir la liste des commandes, même pour les abonnements expirés.
- Une commande ne peut être supprimée que si aucun exemplaire lié à cette commande n'est enregistré.
- Au démarrage de l'application, une petite fenêtre doit s'ouvrir automatiquement pour afficher les revues dont l'abonnement se termine dans moins de 30 jours.

Temps de travail estimé  
réel  
4 heures

Temps de travail  
  
6 heures

## Kanban de la tâche actuelle "In Progress"



# Tâche 1

## Processus

Actualisation du design de l'application avec la création des onglets Commandes livres et DVD :

**Gestion des documents de la médiathèque**

Liens : Livres | DVD | Revues | Parutions des revues | **Commandes livres** | Commandes DVD | Commandes revues

Recherche d'une commande

Saisir un numéro de document :

Date commande	Montant	Nb exemplaires	Suivi
08/04/2023	1234	14	livrée
08/04/2023	100	10	livrée
07/04/2023	150	10	livrée
07/04/2023	250	5	livrée
07/04/2023	1000	10	livrée
06/04/2023	500	10	livrée
05/04/2023	3500	12	livrée
05/04/2023	1200	12	livrée

Informations détaillées

Número de document :  Code ISBN :

Titre :

Auteur(e) :

Collection :

Genre :

Public :

Rayon :

Image :

Commandes

Date :

Número de commande :

Nombre d'exemplaires :

Montant :

Etape du suivi :

Gestion des commandes livres

Lors de l'insertion dans le champ "Saisir un numéro de document", les données sont récupérées d'une part dans la table Livre afin d'indiquer les informations détaillées du document correspondant, et le datagrid de commande indique les commandes issues de ce numéro de livre. En bas, les informations de commande sont également renseignées en fonction de la commande sélectionnée dans le datagrid.

Les boutons "commander", "modifier" et "supprimer" sont également implémentés afin de répondre à la demande.



Dans un premier temps le datagrid est créé dans le frame et codé dans la classe *FrmMediatek.cs* en assignant la liste reçue en paramètre à la bindingsource *bdgCommandesLivres* :

```
private void RemplirCmdLivreListe(List<CommandeDocument> lesCommandesDocument)
{
    bdgCommandesLivres.DataSource = lesCommandesDocument;
    dgvCmdLivreListe.DataSource = bdgCommandesLivres;
    dgvCmdLivreListe.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
    dgvCmdLivreListe.Columns["IdLivreDvd"].Visible = false;
    dgvCmdLivreListe.Columns["IdSuivi"].Visible = false;
    dgvCmdLivreListe.Columns["Id"].Visible = false;
    dgvCmdLivreListe.Columns["DateCommande"].DisplayIndex = 0;
    dgvCmdLivreListe.Columns["DateCommande"].HeaderText = "Date commande";
    dgvCmdLivreListe.Columns["Montant"].DisplayIndex = 1;
    dgvCmdLivreListe.Columns["NbExemplaire"].DisplayIndex = 2;
    dgvCmdLivreListe.Columns["NbExemplaire"].HeaderText = "Nb exemplaires";
    dgvCmdLivreListe.Columns["LibelleSuivi"].HeaderText = "Suivi";
}
```

Cette méthode est initialisée lors de l'évènement click sur le bouton de recherche. Des contrôles sont mis en place afin de guider l'utilisateur :

```
private void BtnCmdLivreNumRecherche_Click(object sender, EventArgs e)
{
    if (DefaultCmdLivre && !txbCmdLivreNumRecherche.Text.Equals(""))
    {
        Livre livre = lesLivres.Find(x => x.Id.Equals(txbCmdLivreNumRecherche.Text.ToString()));
        if (livre != null)
        {
            AfficherCmdLivreInfos(livre);
            RemplirCmdLivreListe(lesCommandesDocuments);
        }
        else
        {
            MessageBox.Show("Numéro introuvable");
        }
    }
    else
    {
        MessageBox.Show("Veuillez entrer un numéro de livre valide");
        TabCommandesLivres_Enter(sender, e);
    }
}
```

La méthode qui permet d'afficher les informations du livre reçoit également les informations de commande concernant le livre recherché. Voici la nouvelle méthode qui fait appel au controller pour récupérer en base de données le document correspondant :

```
public void AfficherCmdLivreInfoCmdGrid()
{
    string idDocument = txbCmdLivreNumRecherche.Text;
    lesCommandesDocuments = controller.GetCommandesDocument(idDocument);
    RemplirCmdLivreListe(lesCommandesDocuments);
}
```

Puis en appelant cette méthode dans *AfficherCmdLivresInfos* :

```
1 référence
private void AfficherCmdLivresInfos(Livre livre)
{
    txbCmdLivreIdLivreDvd.Text = livre.Id;
    txbCmdLivreIsbn.Text = livre.Isbn;
    txbCmdLivreTitre.Text = livre.Titre;
    txbCmdLivreAuteur.Text = livre.Auteur;
    txbCmdLivreCollection.Text = livre.Collection;
    txbCmdLivreGenre.Text = livre.Genre;
    txbCmdLivrePublic.Text = livre.Public;
    txbCmdLivreRayon.Text = livre.Rayon;
    string image = livre.Image;
    try...
    AfficherCmdLivreInfoCmdGrid();
}
```

Du côté du controller *FrmMediatekController.cs*, la demande est envoyée à la classe *Access.cs* :

```
/// <summary>
/// Récupère les commandes d'un document de type livre ou DVD
/// </summary>
/// <param name="idDocument">id du document concerné</param>
/// <returns>Liste d'objets Commande</returns>
2 références
public List<CommandeDocument> GetCommandesDocument(string idDocument) => access.GetCommandesDocuments(idDocument);
```

*Access.cs* traite la demande et formule la requête envoyée à l'api sous la forme d'une liste, en reprenant en paramètre l'id du document concerné par la demande utilisateur côté vue :

```
/// <summary>
/// Retourne les commandes document à partir de la BDD
/// </summary>
/// <param name="idDocument">id du document concerné</param>
/// <returns>Liste d'objets CommandeDocument</returns>
1 référence
public List<CommandeDocument> GetCommandesDocuments(string idDocument)
{
    List<CommandeDocument> lesCommandesDocuments = TraitementRecup<CommandeDocument>(GET, "commandesdocuments/" + idDocument);
    return lesCommandesDocuments;
}
```

Les informations de commande récupérées, les informations détaillées du livre sont récupérées en faisant l'appel au controller dès l'initialisation de l'onglet :

```
private void TabCommandesLivres_Enter(object sender, EventArgs e)
{
    if (DefaultCmdLivre)
    {
        lesLivres = controller.GetAllLivres();
    }
}
```

Et liées au livre cible en fonction du numéro de recherche dans la méthode *BtnCmdLivreNumRecherche\_Click* que nous avons déjà vu précédemment :

```
Livre livre = lesLivres.Find(x => x.Id.Equals(txbCmdLivreNumRecherche.Text.ToString()));
if (livre != null)
{
    AfficherCmdLivresInfos(livre);
}
```

Si l'utilisateur désire commander un nouveau livre, il va donc déclencher l'évènement click sur le bouton "*commander*", une méthode permet d'afficher les champs nécessaires (qui ne sont pas éditables avant qu'un choix utilisateur soit effectué pour les commandes, modifications et suppressions) :

```
private void BtnCmdLivreCmd_Click(object sender, EventArgs e)
{
    // Désactive le datagrid
    DefaultCmdLivre = false;
    dgvCmdLivreListe.Enabled = false;
    // Active l'accès à la gestion de la commande
    AccesInfosCmdLivreGrpBox(true);
    ActiverChampsInfosCmdLivre();
    ViderCmdLivreInfosCmd();
    cbxCmdLivreSuivi.SelectedIndex = 0;
    // Liste des contrôles à masquer
    Control[] hide = { btnCmdLivreCmd, btnCmdLivreModifier, btnCmdLivreSupprimer };
    foreach (Control control in hide) { control.Hide(); }
    // Liste des contrôles à afficher
    Control[] show = { btnCmdLivreCmdOk, btnCmdLivreAnnuler };
    foreach (Control control in show) { control.Show(); }
}
```

On retrouve alors les boutons permettant d'enregistrer ou d'annuler cette commande :

Commandes

Date : mercredi 12 avril 2023

Numéro de commande:

Nombre d'exemplaires :

Montant :

Etape du suivi :

Gestion des commandes livres

**Annuler** **Enregistrer**

On remarque que les champs de commande sont vides et cette fois-ci accessibles. Les étapes du suivi ne sont pas modifiables, car une nouvelle commande aura toujours ce statut "*en cours*" lors de sa création.

Si l'évènement click sur le bouton "*enregistrer*" est déclenché, on récupère les informations indiquées dans les champs liés à la commande, en y ajoutant un champ caché qui permet de récupérer l'id de l'étape du suivi ainsi que le champ de l'id du livre concerné. Des conditions sont codées afin que l'utilisateur renseigne les données attendues et qu'en cas de refus celui-ci en comprenne la cause. On remarque à la suite des conditions l'appel à deux méthodes. La première *RestaurerConfigCmdLivres* qui comme son nom l'indique regroupe les éléments visuels et évènements à restaurer suite à l'appel de *TabCommandesLivre\_Enter* :

(voir page suivante)

```

private void BtnCmdLivreCmdOk_Click(object sender, EventArgs e)
{
    if (!txbCmdLivreNumRecherche.Equals(""))
    {
        try
        {
            string id = txbCmdLivreIdCmd.Text;
            DateTime dateCommande = dtpCmdLivre.Value;
            double montant = double.Parse(txbCmdLivreMontant.Text);
            int nbExemplaire = int.Parse(txbCmdLivreNbExemplaires.Text);
            string idSuivi = txbCmdLivreIdSuivi.Text;
            string libelleSuivi = cbxCmdLivreSuivi.SelectedItem.ToString();
            string idLivreDvd = txbCmdLivreIdLivreDvd.Text;
            CommandeDocument commandeDocument = new CommandeDocument(id, dateCommande, montant, nbExemplaire, idLivreDvd, idSuivi, libelleSuivi);
            Console.WriteLine(commandeDocument);
            if (controller.CreerCommandeDocument(commandeDocument))
            {
                MessageBox.Show("Commande " + id + " effectuée", "Information");
                AfficherCmdLivreInfosCmd(commandeDocument);
            }
            else
            {
                MessageBox.Show("Commande déjà existante", "Erreur");
            }
        }
        catch
        {
            MessageBox.Show("Veuillez-vérifier que tous les champs soient correctement renseignés", "Information");
        }
    }
    else
    {
        MessageBox.Show("Aucun livre sélectionné");
    }
    RestaureConfigCmdLivres();
    TabCommandesLivres_Enter(sender, e);
}

```

Nous remarquons l'appel au controller afin de créer une commande de document :

```

/// <summary>
/// Créer une commande document dans la BDD
/// </summary>
/// <param name="commandeDocument">le document commandé</param>
/// <returns>true si la création a pu se faire</returns>
public bool CreerCommandeDocument(CommandeDocument commandeDocument) => access.CreerCommandeDocument(commandeDocument);

```

Et à nouveau une demande côté Access.cs qui s'occupe de formuler la requête pour l'api sous la forme d'une liste :

```

/// <summary>
/// Ajout d'une commande document en BDD
/// </summary>
/// <param name="commandeDocument">objet commande document</param>
/// <returns>true si l'insertion a pu se faire (retour != null)</returns>
1 référence
public bool CreerCommandeDocument(CommandeDocument commandeDocument)
{
    String jsonCreerCommandeDocument = JsonConvert.SerializeObject(commandeDocument, new CustomDateTimeConverter());
    try
    {
        // récupération soit d'une liste vide (requête ok) soit de null (erreur)
        List<CommandeDocument> liste = TraitementRecup<CommandeDocument>(POST, "commandedocument/" + jsonCreerCommandeDocument);
        return (liste != null);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Log.Error("Access.CreerCommandeDocument catch jsonCreerDocument={0} error={1}", jsonCreerCommandeDocument, ex);
    }
    return false;
}

```

Du côté de l'api, la ligne de la requête est lue et formulée vers la base de données :

```
/**
 * Ajout d'une commande de type livre
 * @param type $champs non et valeur de chaque champs
 */
public function insertCommandeDocument($champs) {
    // tableau associatif des données de commande
    $champsCommande = [
        "Id" => $champs["Id"],
        "DateCommande" => $champs["DateCommande"],
        "Montant" => $champs["Montant"]
    ];
    $resultCommande = $this->insertSimple("commande", $champsCommande);

    // tableau associatif des données commande document
    $champsCommandeDocument = [
        "Id" => $champs["Id"],
        "NbExemplaire" => $champs["NbExemplaire"],
        "IdLivreDvd" => $champs["IdLivreDvd"],
        "IdSuivi" => $champs["IdSuivi"]
    ];
    $resultCommandeDocument = $this->insertSimple("commandedocument", $champsCommandeDocument);

    return $resultCommande && $resultCommandeDocument;
}
```

Dans le cas d'un déclencheur sur le bouton d'annulation, une demande de confirmation est effectuée et les méthodes afin de rétablir l'onglet Commandes Livres à son état initial ainsi que l'appel de celui-ci sont codées :

```
private void BtnCmdLivreAnnuler_Click(object sender, EventArgs e)
{
    if (MessageBox.Show(this, "Voulez-vous annuler votre demande?", "Information",
        MessageBoxButtons.YesNo, MessageBoxIcon.Warning, MessageBoxDefaultButton.Button2) == DialogResult.Yes)
    {
        MessageBox.Show("Annulation réussie.", "Information");
        RestaureConfigCmdLivres();
        TabCommandesLivres_Enter(sender, e);
    }
}
```

Concernant la modification d'un livre et le déclencher sur ce bouton, une nouvelle méthode permet d'afficher les champs demandés et ici précisément de pouvoir agir sur le combobox Suivi :

```
// Active la modification statut commande + récupère item
cbxCmdLivreSuivi.Enabled = true;
recupSuivi = cbxCmdLivreSuivi.SelectedItem.ToString();
```

Si l'utilisateur, une fois que toutes les informations requises sont renseignées, désire enregistrer la modification de la commande ciblée, différents tests permettent de vérifier que le suivi ne soit pas incohérent avec le cahier des charges :

```
private void BtnCmdLivreModifierOk_Click(object sender, EventArgs e)
{
    if (!txtCmdLivreIdCmd.Equals(""))
    {
        try
        {
            string id = txtCmdLivreIdCmd.Text;
            DateTime dateCommande = dtpCmdLivre.Value;
            double montant = double.Parse(txtCmdLivreMontant.Text);
            int nbExemplaire = int.Parse(txtCmdLivreNbExemplaires.Text);
            string idSuivi = txtCmdLivreIdSuivi.Text;
            string libelleSuivi = cbxCmdLivreSuivi.SelectedItem.ToString();
            string idLivreDvd = txtCmdLivreIdLivreDvd.Text;
            CommandeDocument commandeDocument = new CommandeDocument(id, dateCommande, montant, nbExemplaire, idLivreDvd, idSuivi, libelleSuivi);
            if (MessageBox.Show(this, "Confirmez-vous la modification de cette commande ?", "INFORMATION", MessageBoxButtons.YesNo, MessageBoxIcon.Warning,
                MessageBoxDefaultButton.Button2) == DialogResult.Yes)
            {
                if (recupSuivi.Equals("réglée") || recupSuivi.Equals("livrée"))
                {
                    if (libelleSuivi.Equals("en cours") || libelleSuivi.Equals("relancée"))
                    {
                        MessageBox.Show("Impossible de rétrograder une commande déjà " + recupSuivi, "Erreur");
                    }
                    else
                    {
                        controller.ModifierCommandeDocument(commandeDocument);
                        MessageBox.Show("Modification de la commande effectuée", "Information");
                    }
                }
                else if (recupSuivi.Equals("en cours") || recupSuivi.Equals("relancée"))
                {
                    if (libelleSuivi.Equals("réglée"))
                    {
                        MessageBox.Show("Une commande ne peut être réglée avant sa livraison", "Erreur");
                    }
                    else
                    {
                        controller.ModifierCommandeDocument(commandeDocument);
                        MessageBox.Show("Modification de la commande effectuée", "Information");
                    }
                }
                else
                {
                    controller.ModifierCommandeDocument(commandeDocument);
                    MessageBox.Show("Modification de la commande effectuée", "Information");
                }
            }
        }
        catch
        {
            MessageBox.Show("Veuillez sélectionner une commande valide", "Erreur");
        }
    }
    else
    {
        MessageBox.Show("Merci de renseigner un numéro de commande valide", "Erreur");
    }
    RestaureConfigCmdLivres();
    TabCommandesLivres_Enter(sender, e);
}
}
```

L'appel au contrôleur pour la méthode *ModifierCommandeDocument* avec en paramètre l'objet dont on désire la modification, ici un objet de type *CommandeDocument*, qui fait appel à la méthode correspondante dans *Access.cs* :

```
/// <summary>
/// Modifier la commande d'un livre dans la BDD
/// </summary>
/// <param name="commandeDocument">l'objet commande document concerné</param>
/// <returns>true si la modification a pu se faire</returns>
4 références
public bool ModifierCommandeDocument(CommandeDocument commandeDocument) => access.ModifierCommandeDocument(commandeDocument);
```

Puis dans la classe *Access.cs*, on converti comme pour chaque requête l'objet de type *CommandeDocument* au format json afin de pouvoir l'envoyer sous forme de liste, précédé de l'id lié au document ciblé ainsi que du verbe HTTP PUT :

```
public bool ModifierCommandeDocument(CommandeDocument commandeDocument)
{
    String jsonCommandeDocument = JsonConvert.SerializeObject(commandeDocument);
    Console.WriteLine(jsonCommandeDocument);
    try
    {
        // récupération soit d'une liste vide (requête ok) soit de null (erreur)
        List<CommandeDocument> liste = TraitementRecup<CommandeDocument>(PUT, "commandedocument/" + commandeDocument.Id + "/" + jsonCommandeDocument);
        return (liste != null);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Log.Error("Access.ModifierCommandeDocument catch jsonCommandeDocument={0} error={1}", jsonCommandeDocument, ex);
    }
    return false;
}
```

Du côté de l'api, le verbe HTTP PUT correspond bien à une requête à trois paramètres :

```
} else if ($_SERVER['REQUEST_METHOD'] === 'PUT') {
    $controle->put($table, $id, $contenu);
}
```

Nous faisons donc appel à la fonction *put* présente dans contrôle dans *Controle.php*, qui formule une réponse valide ou non à cette demande en fonction de la requête écrite dans *Access.php* :

```
public function put($table, $id, $champs) {
    $result = $this->accessBDD->updateOne($table, $id, $champs);
    if ($result == null || $result == false) {
        $this->reponse(400, "requete invalide");
    } else {
        $this->reponse(200, "OK");
    }
}
```

On retrouve le nom de table "commandedocument" présent dans notre *Access.cs* :

```
public function updateOne($table, $id, $champs) {
    if ($this->conn != null && $champs != null) {
        switch ($table) {
            case "livre" { ...2 lines }
            case "dvd" { ...2 lines }
            case "revue" { ...2 lines }
            case "modifierexemplaire" { ...2 lines }
            case "commandedocument" :
                return $this->updateCommandeDocument($id, $champs);
        }
    }
}
```



Et l'écriture de la requête correspondante qui permet d'appeler la fonction *updateSimple* (traitant et reformulant les lignes de la requête) pour chaque table concernée, et les retournant via les deux variables initialisées (*\$resultCommandeDocument* et *\$resultCommande*) :

```
public function updateCommandeDocument($id, $champs) {
    // tableau associatif des données commande document
    $champsCommandeDocument = [
        "Id" => $champs["Id"],
        "NbExemplaire" => $champs["NbExemplaire"],
        "IdLivreDvd" => $champs["IdLivreDvd"],
        "IdSuivi" => $champs["IdSuivi"]
    ];
    $resultCommandeDocument = $this->updateSimple("commandedocument", $id, $champsCommandeDocument);

    // tableau associatif des données commande
    $champsCommande = [
        "Id" => $champs["Id"],
        "DateCommande" => $champs["DateCommande"],
        "Montant" => $champs["Montant"]
    ];
    $resultCommande = $this->updateSimple("commande", $id, $champsCommande);

    return $resultCommandeDocument && $resultCommande;
}
```

Dans le cas d'un delete, nous utilisons comme pour les fonction insert ou update un switch permettant de sélectionner la table correspondante et ce en fonction de la requête envoyée par la classe C# Access.cs :

```
public function deleteOne($table, $champs) {
    if ($this->conn != null && $champs != null) {
        switch ($table) {
            case "livre" { ...2 lines }
            case "dvd" { ...2 lines }
            case "revue" { ...2 lines }
            case "commandedocument" :
                return $this->deleteCommandeDocument($champs);
        }
    }
}
```

```
public function deleteCommandeDocument($champs) {
    // tableau associatif des données commande
    $champsCommande = [
        "Id" => $champs["Id"],
        "DateCommande" => $champs["DateCommande"],
        "Montant" => $champs["Montant"]
    ];
    $resultCommande = $this->deleteSimple("commande", $champsCommande);

    // tableau associatif des données commande document
    $champsCommandeDocument = [
        "Id" => $champs["Id"],
        "NbExemplaire" => $champs["NbExemplaire"],
        "IdLivreDvd" => $champs["IdLivreDvd"],
        "IdSuivi" => $champs["IdSuivi"]
    ];
    $resultCommandeDocument = $this->deleteSimple("commandedocument", $champsCommandeDocument);

    return $resultCommande && $resultCommandeDocument;
}
```



Trigger supprimant également une commande dans la classe fille :

- pour la table *abonnement*

```
1 DROP TRIGGER IF EXISTS delete_abonnement_to_commande_by_id;
2 DELIMITER //
3 CREATE TRIGGER delete_abonnement_to_commande_by_id AFTER DELETE ON abonnement
4 FOR EACH ROW BEGIN
5     DELETE FROM commande WHERE id = OLD.id;
6 END //
7 DELIMITER ;
```

- pour la table *commandedocument*

```
1 DROP TRIGGER IF EXISTS delete_commandedocument_to_commande_by_id;
2 DELIMITER //
3 CREATE TRIGGER delete_commandedocument_to_commande_by_id AFTER DELETE ON commandedocument
4 FOR EACH ROW BEGIN
5     DELETE FROM commande WHERE id = OLD.id;
6 END //
7 DELIMITER ;
```

Trigger qui contrôle la contrainte de partition de l'héritage sur commande :

- lors d'un insert de type *abonnement*

```
1 DROP TRIGGER IF EXISTS insert_abonnement_to_commande_by_id;
2 DELIMITER //
3 CREATE TRIGGER insert_abonnement_to_commande_by_id BEFORE INSERT ON abonnement
4 FOR EACH ROW BEGIN
5     IF NOT EXISTS (SELECT id FROM commande WHERE id = NEW.id) THEN
6         INSERT INTO commande (id) VALUES (NEW.id);
7     END IF;
8 END //
9 DELIMITER ;
```

- lors d'un insert de type *commandedocument*

```
1 DROP TRIGGER IF EXISTS insert_commandedocument_to_commande_by_id;
2 DELIMITER //
3 CREATE TRIGGER insert_commandedocument_to_commande_by_id BEFORE INSERT ON commandedocument
4 FOR EACH ROW BEGIN
5     IF NOT EXISTS (SELECT id FROM commande WHERE id = NEW.id) THEN
6         INSERT INTO commande (id) VALUES (NEW.id);
7     END IF;
8 END //
9 DELIMITER ;
```

Trigger créant le nombre d'exemplaires adéquats en fonction d'une commande passant à l'étape "livrée" :

```
1 DROP TRIGGER IF EXISTS update_exemplaires_on_commande;
2 DELIMITER //
3 CREATE TRIGGER update_exemplaires_on_commande AFTER UPDATE ON commandedocument
4 FOR EACH ROW BEGIN
5     DECLARE max_exemplaire INTEGER;
6     DECLARE num_exemplaire INTEGER;
7     DECLARE dateAchat DATE;
8     DECLARE cnt INTEGER DEFAULT 0;
9
10    SELECT dateCommande INTO dateAchat FROM commande WHERE id = NEW.id;
11    SELECT MAX(numero) INTO max_exemplaire FROM exemplaire WHERE id = NEW.idlivreDvd;
12
13    IF (NEW.idSuivi != OLD.idSuivi AND NEW.idSuivi = "00004") THEN
14        IF (max_exemplaire IS NOT NULL) THEN
15            SET num_exemplaire = max_exemplaire;
16        ELSE
17            SET num_exemplaire = 0;
18        END IF;
19
20        WHILE cnt < NEW.nbExemplaire DO
21            SET num_exemplaire = num_exemplaire + 1;
22            INSERT INTO exemplaire (id, numero, dateAchat, photo, idEtat)
23            VALUES (NEW.idLivreDvd, num_exemplaire, dateAchat, "", "00001");
24            SET cnt = cnt + 1;
25        END WHILE;
26    END IF;
27 END //
28 DELIMITER ;
```

Dans ce trigger, le processus est plus long et c'est pourquoi j'ai fait le choix de le détailler.

Nous devons récupérer dans un premier différentes valeurs présentes dans d'autres tables que commandedocument (*dateAchat* de commande et *max\_exemplaire* pour la table exemplaire), et créé deux variables qui nous permettront de stocker le nouveau numéro d'exemplaire (*num\_exemplaire*) et initialiser un compteur via la variable *cnt* initialisé à 0.

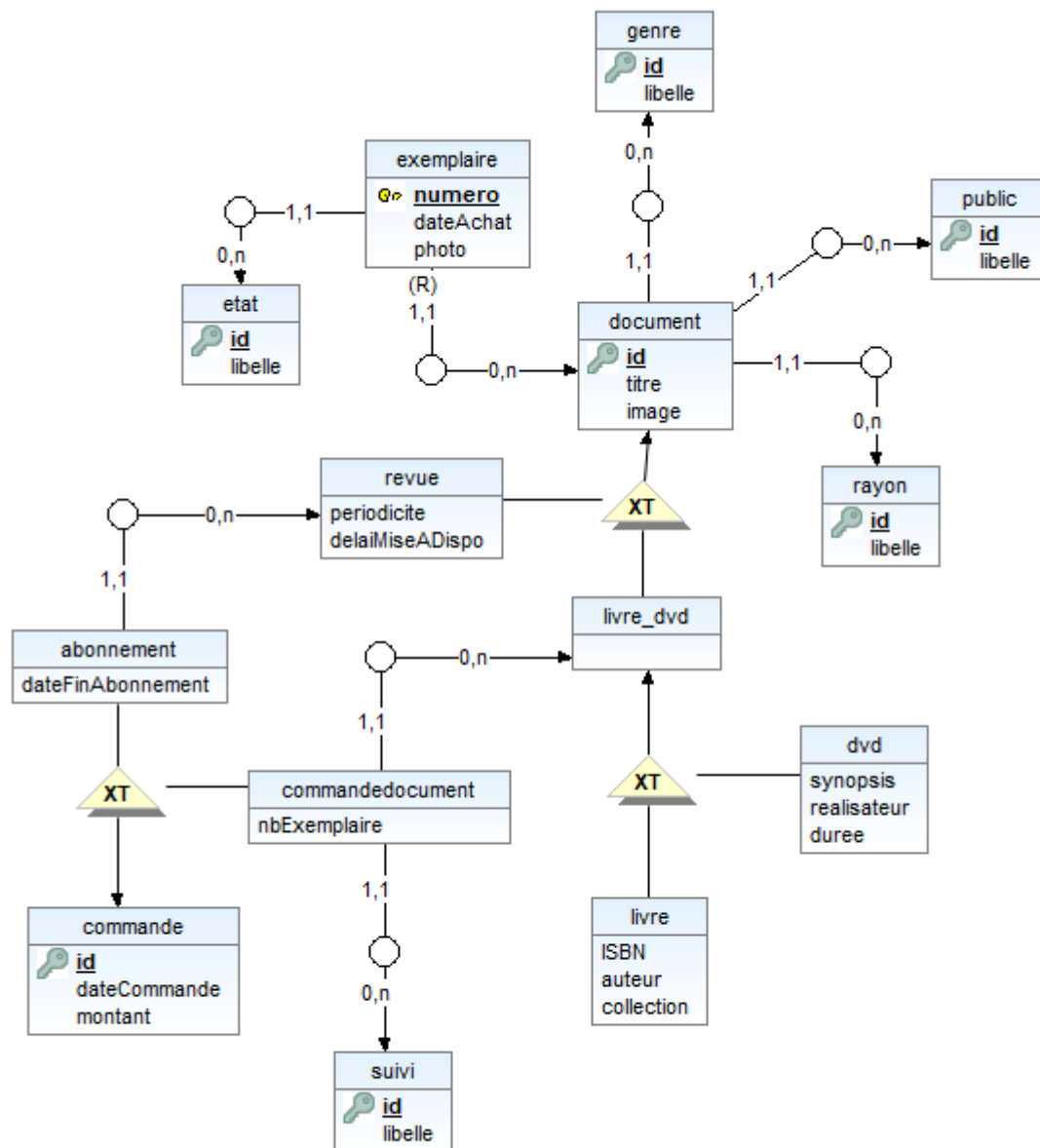
Le premier *if* vérifie si l'étape de livraison de l'id de la commande mise à jour est différente ou non de l'ancien ID et si ce nouvel ID a pour idSuivi '00004'.

Si tel est le cas, et si (nouveau *if*) le numéro maximum de l'exemplaire récupéré via le *select* correspondant n'est pas nul, alors le nouveau *num\_exemplaire* aura sa valeur. Par exemple *max\_exemplaire* a pour numéro 5, alors *num\_exemplaire* sera égal à 5.

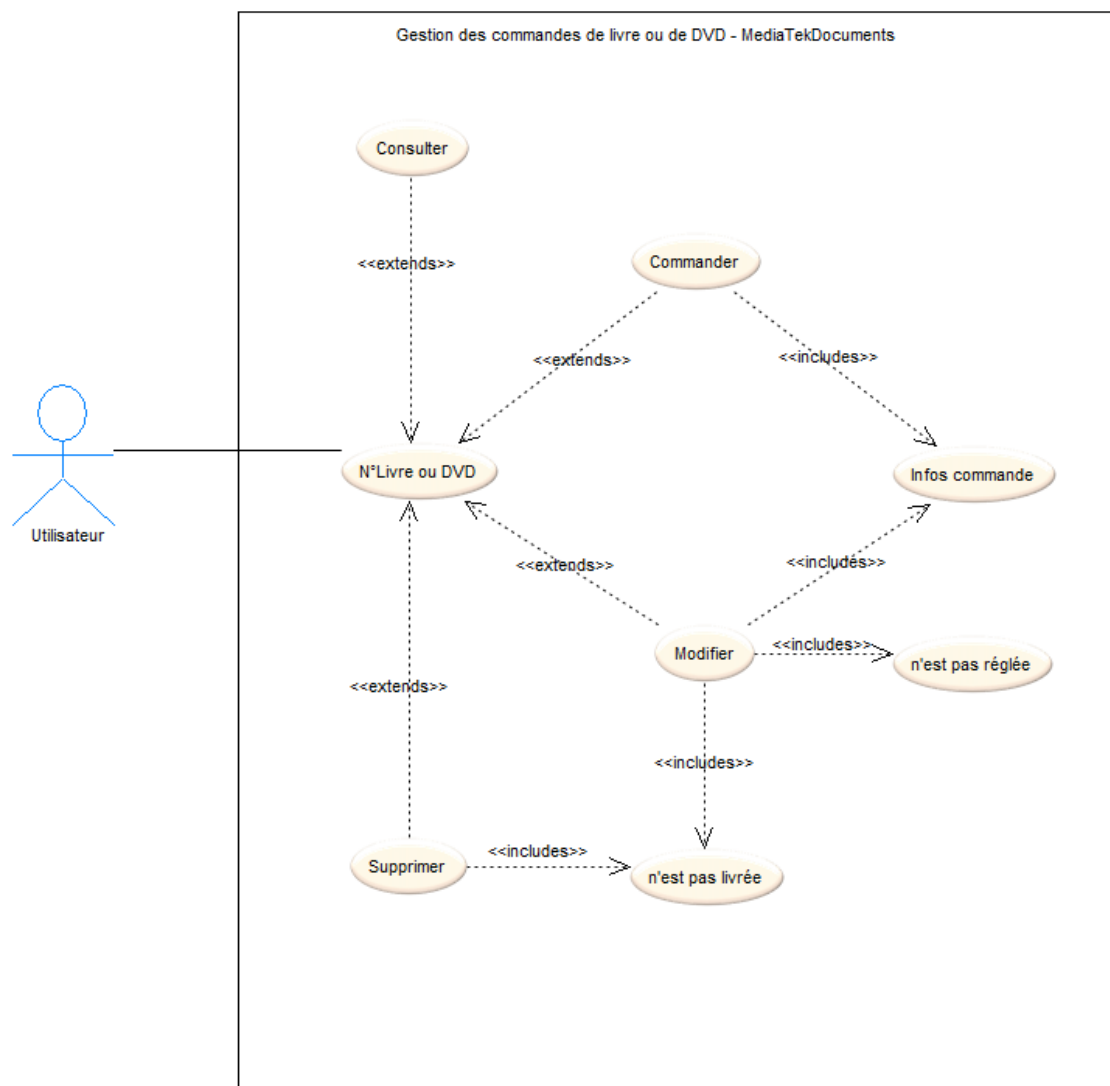
Sinon (*else*), aucun numéro maximum d'exemplaire est présent dans la table exemplaire correspondant au nouvel *idLivreDvd*. Le numéro de l'exemplaire sera ainsi de 0.

La boucle *while* continue jusqu'à ce que le compteur (*cnt*) initialisé à 0 reste inférieur au nouveau nombre d'exemplaires (*nbExemplaire*). A chaque tour de boucle, on initialise *num\_exemplaire* et on lui incrémente +1, ce qui permet de créer un nouveau numéro d'exemplaire pour chaque exemplaire ajouté. C'est également pour le cas du compteur.

On utilise enfin l'instruction *insert into* pour ajouter un nouvel exemplaire pour le livre ou le DVD concerné. Les valeurs pour chaque colonne de la table exemplaire sont spécifiées dans la clause *values* de cette instruction. Les valeurs de *id* et *numero* sont définies à partir des valeurs de *NEW.idLivreDvd* et *num\_exemplaire*, respectivement. La valeur de *dateAchat* est définie à partir de la variable *date\_commande*. La valeur de *photo* est définie à une chaîne vide (") car cela n'est pas l'objectif de ce travail. La valeur de *idEtat* est définie à '00001', ce qui représente l'état initial de l'exemplaire.



## Diagramme de cas d'utilisation



## Tâche 2

Une commande revue est différente des documents de type livre ou DVD, car celle-ci consiste à créer ou renouveler un nouvel abonnement.

On code les différentes méthodes dans *FrmMediatek.cs* en ayant à nouveau au préalable créé les nouveaux éléments requis :

Livres

DVD

Revue

Parutions des revues

Commandes livres

Commandes DVD

Commandes revues

Recherche d'une commande

Saisir un numéro de document :

Rechercher

Informations détaillées

Numéro de document :

Titre :

Périodicité :

Délai mise à dispo :

Genre :

Public :

Rayon :

Image :

Commandes

Numéro de commande:

Date de commande :

Date de fin d'abonnement :

Montant :

Gestion des abonnements revues

Supprimer

Modifier

Commander

Les méthodes permettant de gérer l'affichage nécessaires en fonction des déclencheurs actionnés par l'utilisateur sont identiques aux méthodes des boutons présentés précédemment. Afin de ne pas alourdir ce compte rendu, nous passerons les explications détaillées côté code pour ces nouveaux éléments et les méthodes liées au datagrid. Voici leur visuel lors d'un click utilisateur sur commander ou modifier une revue :

La méthode pour l'ajout d'une nouvelle commande (abonnement) de revue récupère les différents champs demandés par la classe métier *Abonnement.cs* :

```

/// <summary>
/// Ajout d'une commande (= abonnement) de revue en BDD
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void BtnCmdRevueCmdOk_Click(object sender, EventArgs e)
{
    if (!txbCmdRevueNumRecherche.Equals(""))
    {
        try
        {
            string id = txbCmdRevueIdCmd.Text;
            DateTime dateCommande = dtpCmdRevueDateCmd.Value;
            DateTime dateFinAbonnement = dtpCmdRevueFinAbo.Value;
            double montant = double.Parse(txbCmdRevueMontant.Text);
            string idRevue = txbCmdRevueIdRevue.Text;
            Abonnement abonnementRevue = new Abonnement(id, dateCommande, montant, dateFinAbonnement, idRevue);
            if (controller.CreerAbonnement(abonnementRevue))
            {
                MessageBox.Show("Nouvel abonnement de la revue " + idRevue + " effectué.");
            }
            else
            {
                MessageBox.Show("Cet abonnement " + id + " existe déjà", "ERREUR");
            }
        }
        catch
        {
            MessageBox.Show("Vérifiez que tous les champs soient correctement renseignés", "INFORMATION");
        }
    }
    else
    {
        MessageBox.Show("Aucune revue sélectionnée");
    }
    RestaureConfigCmdReves();
    TabCommandesReves_Enter(sender, e);
}

```

On vérifie si la validation de la requête adressée au controller, sous la forme de *CreerAbonnement* avec la liste *abonnementRevue* en objet, est effective. Voici la nouvelle méthode dans la classe *Access.cs* avec sérialisation au format json et création de la ligne de requête HTTP :

(visible page suivante)

```

/// <summary>
/// Ajout d'une commande document en BDD
/// </summary>
/// <param name="commandeDocument">objet commande document</param>
/// <returns>true si l'insertion a pu se faire (retour != null)</returns>
1 référence
public bool CreerCommandeDocument(CommandeDocument commandeDocument)
{
    String jsonCreerCommandeDocument = JsonConvert.SerializeObject(commandeDocument, new CustomDateTimeConverter());
    Console.WriteLine(jsonCreerCommandeDocument);
    try
    {
        // récupération soit d'une liste vide (requête ok) soit de null (erreur)
        List<CommandeDocument> liste = TraitementRecup<CommandeDocument>(POST, "commandedocument/" + jsonCreerCommandeDocument);
        return (liste != null);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return false;
}

```

Côté API REST dans la classe *AccessBDD.php* :

```

/**
 * Ajout d'une commande de type revue
 * @param type $champs nom et valeur de chaque champs
 */
public function insertAbonnementRevue($champs) {
    // tableau associatif des données commande
    $champsCommande = [
        "Id" => $champs["Id"],
        "DateCommande" => $champs["DateCommande"],
        "Montant" => $champs["Montant"]
    ];
    $resultCommande = $this->insertSimple("commande", $champsCommande);

    // tableau associatif des données abonnement
    $champsAbonnement = [
        "Id" => $champs["Id"],
        "DateFinAbonnement" => $champs["DateFinAbonnement"],
        "IdRevue" => $champs["IdRevue"]
    ];
    $resultAbonnement = $this->insertSimple("abonnement", $champsAbonnement);

    return $resultCommande && $resultAbonnement;
}

```

*Afin d'alléger la lecture de ce compte rendu, les méthodes ayant le même fonctionnement que celles expliquées précédemment seront présentées sans détail complémentaire. Veuillez vous référer à la Mission 1 et Mission 2 Tâche 1 pour plus de détails.*



La modification d'une revue côté C# dans *FrmMediatek.cs* :

```
/// <summary>
/// Modification d'une commande de type revue
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void BtnCmdRevueModifierOk_Click(object sender, EventArgs e)
{
    if (!txtCmdRevueIdCmd.Equals(""))
    {
        try
        {
            string id = txtCmdRevueIdCmd.Text;
            DateTime dateCommande = dtpCmdRevueDateCmd.Value;
            double montant = double.Parse(txtCmdRevueMontant.Text);
            DateTime dateFinAbonnement = dtpCmdRevueFinAbo.Value;
            string idRevue = txtCmdRevueIdRevue.Text;
            Abonnement abonnementRevue = new Abonnement(id, dateCommande, montant, dateFinAbonnement, idRevue);
            if (MessageBox.Show(this, "Confirmez-vous la modification de cette commande ?", "INFORMATION",
                MessageBoxButtons.YesNo, MessageBoxIcon.Warning, MessageBoxDefaultButton.Button2) == DialogResult.Yes)
            {
                controller.ModifierCommandeRevue(abonnementRevue);
                MessageBox.Show("Modification de la commande effectuée");
            }
            else
            {
                MessageBox.Show("Erreur lors de la modification de cette commande");
            }
        }
    }
}
```

Et Access.cs :

```
/// <summary>
/// Modification d'une commande revue en BDD
/// </summary>
/// <param name="abonnementRevue">la revue à modifier</param>
/// <returns>true si la modification a pu se faire (retour != null)</returns>
1 référence
public bool ModifierCommandeRevue(Abonnement abonnementRevue)
{
    String jsonCommandeRevue = JsonConvert.SerializeObject(abonnementRevue);
    Console.WriteLine(jsonCommandeRevue);
    try
    {
        // récupération soit d'une liste vide (requête ok) soit de null (erreur)
        List<Abonnement> liste = TraitementRecup<Abonnement>(PUT, "commanderevue/" + abonnementRevue.Id + "/" + jsonCommandeRevue);
        return (liste != null);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Log.Error("Access.ModifierCommandeRevue catch jsonCommandeRevue={0} error={1}", jsonCommandeRevue, ex);
    }
    return false;
}
```

(voir page suivante)

Côté API REST :

```
public function updateCommandeRevue($id, $champs) {
    // tableau associatif des données abonnement
    $champsAbonnement = [
        "Id" => $champs["Id"],
        "DateFinAbonnement" => $champs["DateFinAbonnement"],
        "IdRevue" => $champs["IdRevue"]
    ];
    $resultAbonnement = $this->updateSimple("abonnement", $id, $champsAbonnement);

    // tableau associatif des données commande
    $champsCommande = [
        "Id" => $champs["Id"],
        "DateCommande" => $champs["DateCommande"],
        "Montant" => $champs["Montant"]
    ];
    $resultCommande = $this->updateSimple("commande", $id, $champsCommande);

    return $resultAbonnement && $resultCommande;
}
```

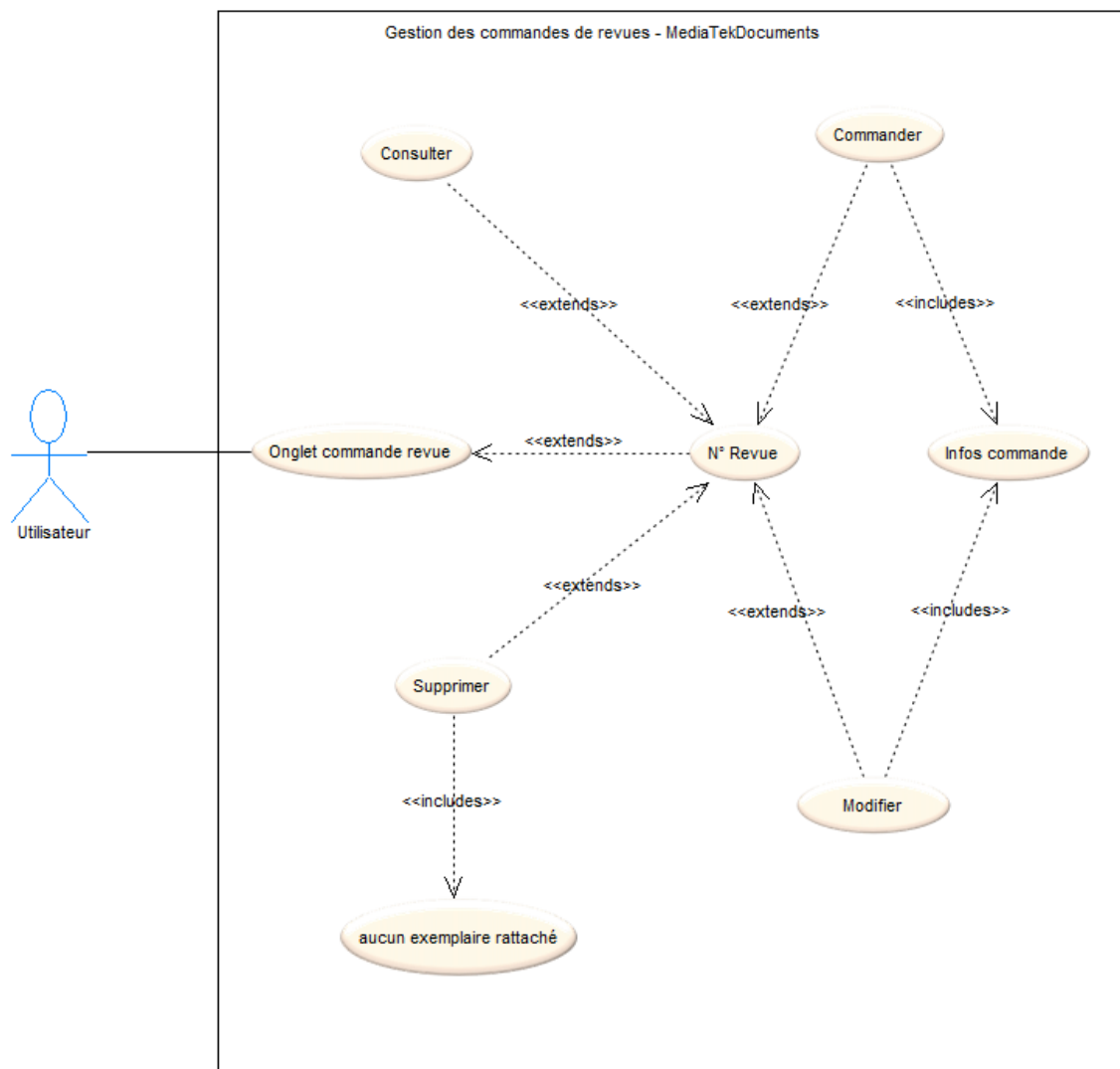
Une revue ne pouvant être supprimée uniquement si aucun exemplaire y est rattaché, donc en vérifiant la date de l'exemplaire comprise entre la date de la commande et la date de fin d'abonnement, nous avons codé deux méthodes dans *FrmMediatek.cs* permettant cette vérification. Dans un premier temps *ParutionAbonnement* effectue une comparaison entre les différentes dates que l'on reçoit (et recevra) en paramètre et retourne le résultat :

```
// <summary>
/// Vérifie si la date de parution est comprise entre date commande et date fin abonnement
/// <param name="dateCommande">date de prise de commande</param>
/// <param name="dateFinAbonnement">date de fin d'abonnement</param>
/// <param name="dateParution">date de comparaison entre les deux précédentes</param>
/// <returns>true si la date est comprise entre ces deux dates</returns>
1 référence
public bool ParutionAbonnement(DateTime dateCommande, DateTime dateFinAbonnement, DateTime dateParution)
{
    return (DateTime.Compare(dateCommande, dateParution) < 0 && DateTime.Compare(dateParution, dateFinAbonnement) < 0);
}
```

Ce résultat est alors exploité dans *VerifierLienAbonnementExemplaire* afin de vérifier si l'exemplaire cible, que l'on récupère via l'api grâce à la requête adressée par le controller *GetExemplairesDocument* avec l'id de la revue cible. La méthode *ParutionAbonnement* est utilisée afin de parcourir le tableau *Exemplaire* et ce afin de vérifier qu'aucun exemplaire ne soit donc rattaché à cette revue :

```
// <summary>
/// Vérifie qu'aucun exemplaire ne soit rattaché à un abonnement
/// </summary>
/// <param name="abonnement">l'abonnement cible</param>
/// <returns>return true si aucun exemplaire rattaché</returns>
1 référence
public bool VerifierLienAbonnementExemplaire(Abonnement abonnement)
{
    List<Exemplaire> lesExemplairesLienAbo = controller.GetExemplairesDocument(abonnement.IdRevue);
    bool supprimer = false;
    foreach (Exemplaire exemplaire in lesExemplairesLienAbo.Where(exemplaires => ParutionAbonnement(
        abonnement.DateCommande, abonnement.DateFinAbonnement, exemplaires.DateAchat)))
    {
        supprimer = true;
    }
    return !supprimer;
}
```

## Diagramme de cas d'utilisation



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

- Ajouter dans les onglets Livres et DVD partie basse la liste des exemplaires du livre sélectionné.
- Sur la sélection d'un exemplaire, il doit être possible de changer son état.
- Dans l'onglet Parutions des revues, permettre aussi le changement d'état.
- Permettre de supprimer un exemplaire.

Temps de travail estimé  
réel  
5 heures

Temps de travail  
  
12 heures

#### Kanban de la tâche actuelle "In Progress"



## Processus

Actualisation des fenêtres *Livres*, *DVD* et *Parutions des revues*. Ajout en bas de page d'un groupe contenant un nouveau datagrid ainsi qu'un groupe de gestion des exemplaires permettant de modifier l'état ou de supprimer l'exemplaire sélectionné, et ce en fonction du livre recherché :

Gestion des documents de la médiathèque

Livres

DVD

Revues

Parutions des revues

Commandes livres

Commandes DVD

Commandes revues

Recherches

Saisir le titre ou la partie d'un titre :

Ou sélectionner le genre :

Saisir un numéro de document :

Rechercher

Ou sélectionner le public :

Ou sélectionner le rayon :

Id	Titre	Auteur	Collection	Genre	Public	Rayon
00001	Quand sort la recluse	Fred Vargas	Commissaire Adamsb...	Policier	Adultes	Policiers français étra...

Informations détaillées

Número de document :

Code ISBN :

Titre :

Auteur(e) :

Collection :

Genre :

Public :

Rayon :

Chemin de l'image :

Image :

Gestion livres

Supprimer

Modifier

Ajouter

Exemplaires du livre sélectionné

	Date d'achat	État	Numéro
►	28/03/2023	neuf	4
	08/04/2023	neuf	9
	08/04/2023	neuf	7
	08/04/2023	détérioré	5
	08/04/2023	neuf	10
	08/04/2022	neuf	8

Gestion exemplaires

Etat actuel :

Nouvel etat :

Modifier

Supprimer

Ecriture de la méthode afin de remplir le datagrid listant les exemplaires par document :

```
/// <summary>
/// Remplir le datagrid avec la liste reçue en paramètre
/// </summary>
/// <param name="lesDetailsExemplaires">liste des exemplaires revue</param>
2 références
private void RemplirExemplairesLivresListe(List<ExemplaireDetail> lesDetailsExemplaires)
{
    if (lesDetailsExemplaires != null)
    {
        bdgExemplairesLivresListe.DataSource = lesDetailsExemplaires;
        dgvExemplairesLivresListe.DataSource = bdgExemplairesLivresListe;
        dgvExemplairesLivresListe.Columns["Photo"].Visible = false;
        dgvExemplairesLivresListe.Columns["Id"].Visible = false;
        dgvExemplairesLivresListe.Columns["IdEtat"].Visible = false;
        dgvExemplairesLivresListe.Columns["LibelleEtat"].HeaderText = "État";
        dgvExemplairesLivresListe.Columns["DateAchat"].HeaderText = "Date d'achat";
        dgvExemplairesLivresListe.Columns["Numero"].HeaderText = "Numéro";
        dgvExemplairesLivresListe.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
        dgvExemplairesLivresListe.Columns["DateAchat"].DisplayIndex = 0;
        dgvExemplairesLivresListe.Columns["LibelleEtat"].DisplayIndex = 1;
    }
    else
    {
        dgvExemplairesLivresListe.DataSource = null;
    }
}
```

Ainsi que pour remplir le combo "état" :

```
/// <summary>
/// Remplir le combo état
/// </summary>
/// <param name="lesEtats">liste des objets de type État</param>
/// <param name="bdg">bindingsource contient les informations</param>
/// <param name="cbx">combobox à remplir</param>
4 références
public void RemplirComboEtat(List<Etat> lesEtats, BindingSource bdg, ComboBox cbx)
{
    bdg.DataSource = lesEtats;
    cbx.DataSource = bdg;
    if (cbx.Items.Count > 0)
    {
        cbx.SelectedIndex = -1;
    }
}
```

Et l'initialiser à l'ouverture d'un onglet livre ou DVD :

```

/// <summary>
/// Ouverture de l'onglet Livres
/// Appel des méthodes pour remplir le datagrid des livres et des combos (genre, rayon, public)
/// Appel des méthodes pour remplir le datagrid des exemplaires et du combo état
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
8 références
private void TabLivres_Enter(object sender, EventArgs e)
{
    if (DefaultLivres)
    {
        lesLivres = controller.GetAllLivres();
        RemplirComboCategorie(controller.GetAllGenres(), bdgGenres, cbxLivresGenres);
        RemplirComboCategorie(controller.GetAllPublics(), bdgPublics, cbxLivresPublics);
        RemplirComboCategorie(controller.GetAllRayons(), bdgRayons, cbxLivresRayons);
        RemplirComboEtat(controller.GetAllEtats(), bdgEtats, cbxExemplairesLivreEtat);
    }
}

```

La méthode pour récupérer l'id correspondant à l'item sélectionné dans le combobox et le stocker dans un textbox invisible pour l'utilisateur de l'application :

```

/// <summary>
/// Récupère l'idEtat en fonction de l'exemplaire select en cbx
/// Ainsi qu'un check dev pour l'id
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void CbxExemplairesLivreEtat_SelectedIndexChanged(object sender, EventArgs e)
{
    if (cbxExemplairesLivreEtat.SelectedIndex >= 0)
    {
        Etat etat = (Etat)cbxExemplairesLivreEtat.SelectedItem;
        cbxExemplairesLivreEtat.Text = etat.Id;
        txbCheckIdEtatExLivre.Text = cbxExemplairesLivreEtat.Text;
    }
}

```

L'événement déclencheur lors du click sur le bouton de modification de l'état d'un exemplaire, afin d'initialiser les éléments visuels adéquats :

```

/// <summary>
/// Active les champs nécessaires à la modification de l'état d'un exemplaire livre
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void BtnModifierExemplairesLivre_Click(object sender, EventArgs e)
{
    // Liste des contrôles à afficher
    Control[] show = { btnModifierExemplairesLivreOk, btnAnnulerExemplairesLivre };
    foreach (Control control in show)
    {
        control.Show();
    }
    // Liste des contrôles à masquer
    Control[] hide = { btnSupprimerExemplairesLivre, btnModifierExemplairesLivre };
    foreach (Control control in hide)
    {
        control.Hide();
    }
    cbxExemplairesLivreEtat.Enabled = true;
}

```

Ainsi que la méthode liée à la validation de la modification de l'état d'un exemplaire :

(voir page suivante)

```
/// <summary>
/// Valide ou non la modification de l'état d'un exemplaire livre dans la BDD
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void BtnModifierExemplairesLivreOk_Click(object sender, EventArgs e)
{
    if (!txbLivresNumero.Text.Equals("") && cbxExemplairesLivreEtat.SelectedIndex != -1)
    {
        try
        {
            ExemplaireDetail exemplairesDet = (ExemplaireDetail)bdgExemplairesLivreListe.List[bdgExemplairesLivreListe.Position];
            int numero = exemplairesDet.Numero;
            DateTime dateAchat = exemplairesDet.DateAchat;
            string photo = exemplairesDet.Photo;
            string idEtat = txbCheckIdEtatExLivre.Text;
            string id = txbLivresNumero.Text;
            Exemplaire exemplaire = new Exemplaire(numero, dateAchat, photo, idEtat, id);
            if (MessageBox.Show(this, "Confirmez-vous la modification de l'état de l'exemplaire " + exemplaire.Numero + " ?", "Information",
                MessageBoxButtons.YesNo, MessageBoxIcon.Warning, MessageBoxDefaultButton.Button2) == DialogResult.Yes)
            {
                controller.ModifierExemplaire(exemplaire);
                MessageBox.Show("Modification de l'état de l'exemplaire n°" + exemplaire.Numero + " réussie", "Information");
            }
            else
            {
                MessageBox.Show("Modification de l'état de l'exemplaire n°" + exemplaire.Numero + " annulée", "Information");
            }
        }
    }
}
```

L'appel du controller côté *Access.cs* via la méthode *ModifierExemplaire* avec pour paramètre l'exemplaire à modifier :

```
/// <summary>
/// Modification d'un exemplaire en BDD
/// </summary>
/// <param name="exemplaire">exemplaire à modifier</param>
/// <returns>true si la modification a pu se faire (retour != null)</returns>
1 référence
public bool ModifierExemplaire(Exemplaire exemplaire)
{
    String jsonModifierExemplaire = JsonConvert.SerializeObject(exemplaire);
    Console.WriteLine(jsonModifierExemplaire);
    try
    {
        // récupération soit d'une liste vide (requête ok) soit de null (erreur)
        List<Exemplaire> liste = TraitementRecup<Exemplaire>(PUT, "modifierexemplaire/" + exemplaire.Numero + "/" + jsonModifierExemplaire);
        return (liste != null);
    }
    catch (Exception ex)
    {
    }
}
```

Côté API REST, écriture de la requête update correspondante :



```

/**
 * Modification d'un exemplaire
 * @param type $id
 * @param type $champs
 * @return type
 */
public function updateExemplaire($id, $champs) {
    $champsExemplaire = [
        'id' => $champs['Id'],
        'numero' => $champs['Numero'],
        'dateAchat' => $champs['DateAchat'],
        'photo' => $champs['Photo'],
        'idEtat' => $champs['IdEtat']
    ];
    $requete = "UPDATE exemplaire SET ";
    foreach ($champsExemplaire as $key => $value) {
        $requete .= "$key=$value,";
    }
    $requete = substr($requete, 0, strlen($requete) - 1);
    $requete .= " WHERE id=:id AND numero=:numero;";
    $champsExemplaire['numero'] = $id;
    $updateExemplaire = $this->conn->execute($requete, $champsExemplaire);
    return $updateExemplaire;
}

```

Dans l'onglet *"Parutions des revues"* nous modifions l'entrée *"photo"* par *"état"* et un bouton pour supprimer un exemplaire également :

Gestion des documents de la médiathèque

Livres
DVD
Revue
Parutions des revues
Commandes livres
Commandes DVD
Commandes revues

Recherche revue

Numéro revue :
Rechercher

Titre :

Périodicité :

Délai mise à dispo :

Genre :

Public :

Rayon :

Chemin de l'image :

Parutions :

Image revue :

Image exemplaire :

Gestion parutions

Etat actuel :

Nouvel état :

Modifier

Supprimer

Nouvelle parution réceptionnée pour cette revue

Numéro réceptionné :

Date de parution :
13/04/2023

Chemin de l'image :
Rechercher

Valider la réception

Image exemplaire :

Supprimer un exemplaire dans *FrmMediatek.cs* :

```

/// <summary>
/// Supprimer un exemplaire revue dans la BDD
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void BtnSupprimerExemplairesRevue_Click(object sender, EventArgs e)
{
    ExemplaireDetail exemplairesDet = (ExemplaireDetail)bdgExemplairesRevueListe.List[bdgExemplairesRevueListe.Position];
    int numero = exemplairesDet.Numero;
    DateTime dateAchat = exemplairesDet.DateAchat;
    string photo = exemplairesDet.Photo;
    string idEtat = txtCheckIdEtatExRevue.Text;
    string id = txtRevueNumero.Text;
    Exemplaire exemplaire = new Exemplaire(numero, dateAchat, photo, idEtat, id);
    if (exemplaire != null)
    {
        try
        {
            if (MessageBox.Show(this, "Confirmez-vous la suppression de l'exemplaire n°" + exemplaire.Numero + " ?", "Information",
                MessageBoxButtons.YesNo, MessageBoxIcon.Warning, MessageBoxDefaultButton.Button2) == DialogResult.Yes)
            {
                controller.SupprimerExemplaire(exemplaire);
                MessageBox.Show("Suppression de l'exemplaire n°" + exemplaire.Numero + " effectuée");
            }
            else
            {
                MessageBox.Show("Suppression de l'exemplaire n°" + exemplaire.Numero + " annulée");
            }
        }
        catch
    }
}

```

L'écriture de la méthode *SupprimerExemplaire* dans *Access.cs* et qui sera appelée par le *Controller* :

```

/// <summary>
/// Suppression d'un exemplaire en BDD
/// </summary>
/// <param name="exemplaire">l'exemplaire à supprimer</param>
/// <returns>true si la suppression a pu se faire (retour != null)</returns>
1 référence
public bool SupprimerExemplaire(Exemplaire exemplaire)
{
    String jsonExemplaire = "{\"id\":\"" + exemplaire.Id + "\", \"numero\":\"" + exemplaire.Numero + "\"}";
    try
    {
        // récupération soit d'une liste vide (requête ok) soit de null (erreur)
        List<Exemplaire> liste = TraitementRecup<Exemplaire>(DELETE, "exemplaire/" + jsonExemplaire);
        return (liste != null);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Log.Error("Access.SupprimerExemplaire catch jsonExemplaire={0} error={1}", jsonExemplaire, ex);
    }
    return false;
}

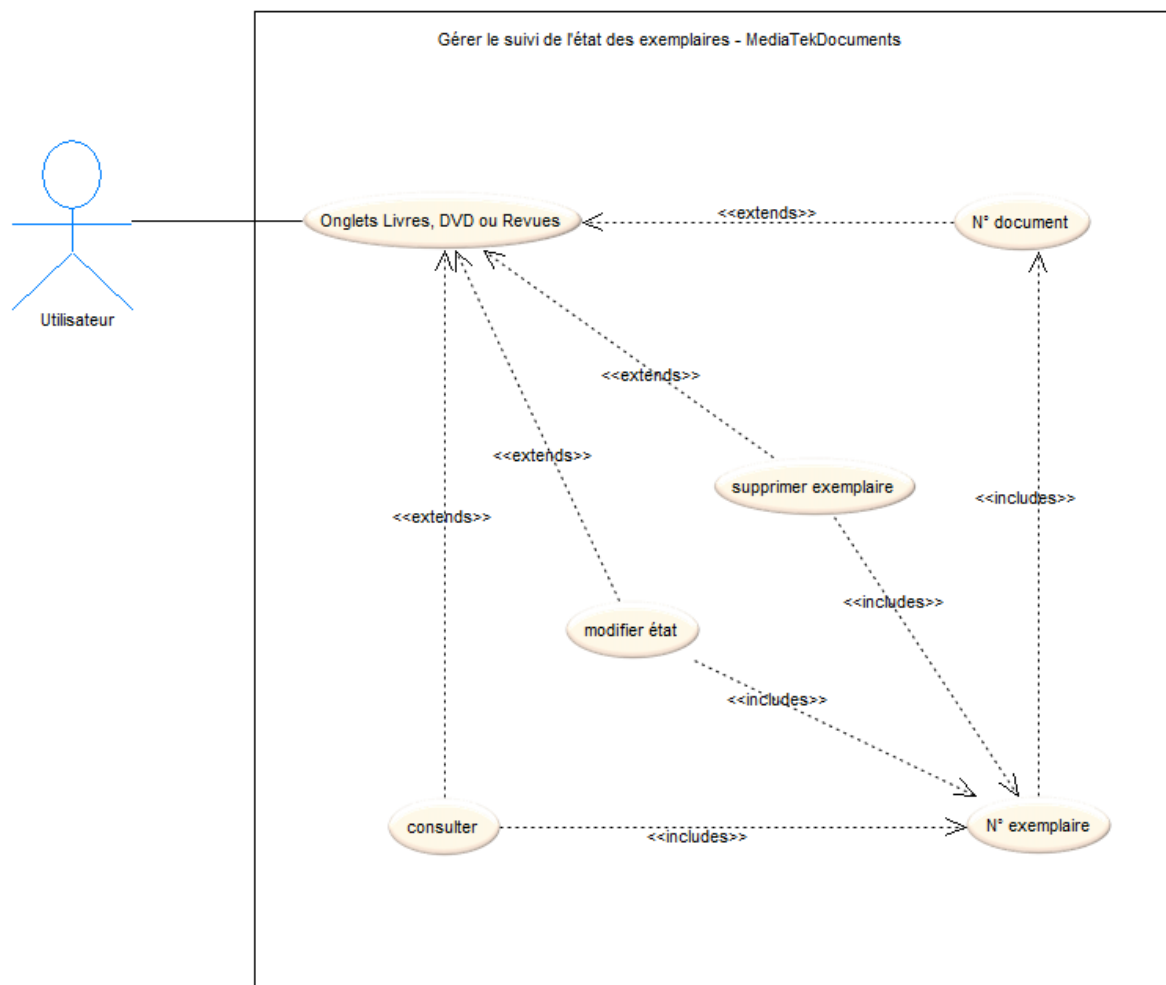
```

Dans l'API REST :

Ici c'est un delete simple car la table exemplaire (PAS SÛR A VOIR)

```
/**
 * Suppresion d'une ou plusieurs lignes dans une table
 * @param string $table nom de la table
 * @param array $champs nom et valeur de chaque champs
 * @return true si la suppression a fonctionné
 */
public function deleteOne($table, $champs) {
    if ($this->conn != null && $champs != null) {
        switch ($table) {
            case "livre" { ...2 lines }
            case "dvd" { ...2 lines }
            case "revue" { ...2 lines }
            case "commandedocument" { ...2 lines }
            case "abonnement" { ...2 lines }
            default:
                // cas d'un delete portant sur une table simple
                return $this->deleteSimple($table, $champs);
        }
    }
}
```

Diagramme de cas d'utilisation



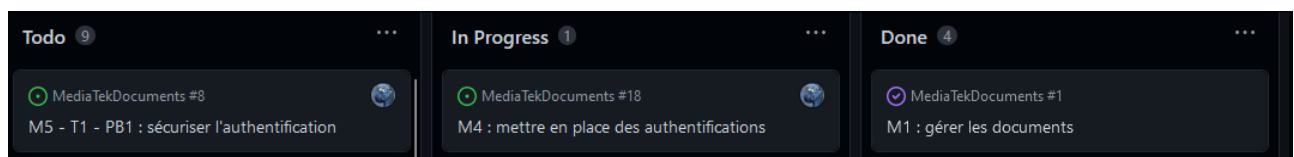
## Mission 4 : mettre en place les authentifications

- Ajouter une table Utilisateur et Service dans la base de données.
- Ajouter une fenêtre d'authentification, faire en sorte que l'application démarre sur cette fenêtre.
- Gérer les conditions d'accès à l'application en fonction du service appartenu. Le service culture n'a pas accès à l'application et l'alerte de fin d'abonnement est uniquement visible par les personnes concernées (qui gèrent les commandes).

Temps de travail estimé  
réel  
4 heures

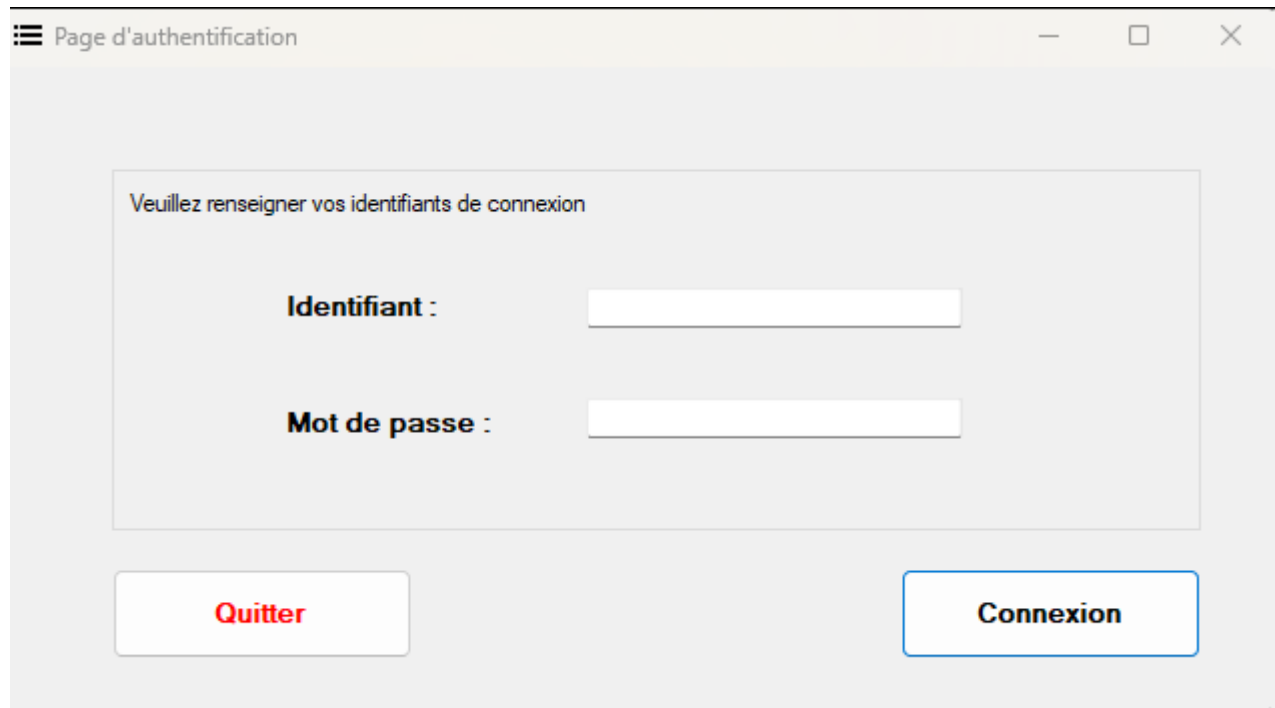
Temps de travail  
  
10 heures

### Kanban de la tâche actuelle "In Progress"



## Processus

Création d'un nouveau frame *FrmAuthentification.cs* ainsi que d'un controller *FrmAuthentificationController.cs* :



Modification du *Main* dans *Program.cs* afin que l'application s'exécute sur ce frame à son lancement :

```
namespace MediaTekDocuments
{
    0 références
    static class Program
    {
        /// <summary>
        /// Point d'entrée principal de l'application.
        /// </summary>
        [STAThread]
        0 références
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FrmAuthentification());
        }
    }
}
```

Le nouveau controller dédié qui récupère également l'instant d'Access afin de pouvoir interagir avec celle-ci :

```
namespace MediaLekDocuments.controller
{
    /// <summary>
    /// Contrôleur lié à FrmAuthentification
    /// </summary>
    3 références
    class FrmAuthentificationController
    {
        /// <summary>
        /// Objet d'accès aux données
        /// </summary>
        private readonly Access access;

        /// <summary>
        /// Récupération de l'instance unique d'accès aux données
        /// </summary>
        1 référence
        public FrmAuthentificationController() => access = Access.GetInstance();

        1 référence
        public String ControleAuthentification(Utilisateur utilisateur) => access.ControleAuthentification(utilisateur);
    }
}
```

Dans *Access.php* écriture de la méthode qui permet de hacher le mot de passe en sha256 :

```
/// <summary>
/// Hashe le mot de passe
/// </summary>
/// <param name="randomString">la valeur à hasher</param>
/// <returns></returns>
1 référence
static string Sha256(string randomString)
{
    var crypt = new System.Security.Cryptography.SHA256Managed();
    var hash = new System.Text.StringBuilder();
    byte[] crypto = crypt.ComputeHash(System.Text.Encoding.UTF8.GetBytes(randomString));
    foreach (byte theByte in crypto)
    {
        hash.Append(theByte.ToString("x2"));
    }
    return hash.ToString();
}
```

Puis de la méthode lue lors du déclenchement du click pour se connecter. On vérifie si l'utilisateur est autorisé ou non à se rendre sur l'application en fonction du service auquel il est rattaché. Le service "50003" n'est ainsi pas autorisé à se connecter :

*(voir page suivante)*



```

/// <summary>
/// Authentification valide ou non en fonction des champs renseignés
/// Hashe du mot de passe via la fonction appelée
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void BtnConnexion_Click(object sender, EventArgs e)
{
    string login = txbAuthentificationLogin.Text.ToString();
    string pwdNoH = txbAuthentificationPwd.Text.ToString();
    string idService = "";
    try
    {
        if (string.IsNullOrEmpty(login) || string.IsNullOrEmpty(pwdNoH))
        {
            MessageBox.Show("Tous les champs doivent être remplis", "Information");
            ViderChampsAuth();
        }
        else
        {
            string pwdWH = Sha256(pwdNoH);
            Utilisateur utilisateur = new Utilisateur(login, pwdWH, idService);
            idService = controller.ControleAuthentification(utilisateur);
            if (idService != null)
            {
                if (idService == "50003")
                {
                    MessageBox.Show("Vos droits ne sont pas suffisants pour accéder à cette application.", "Information");
                    ViderChampsAuth();
                }
                else
                {
                    FrmMediatek frmMediatek = new FrmMediatek(idService);
                    frmMediatek.ShowDialog();
                }
            }
            else
            {
                MessageBox.Show("Mot de passe et/ou login incorrects", "Erreur");
                ViderChampsAuth();
            }
        }
    }
}

```

Côté Access.cs suite à la demande du nouveau controller *FrmAuthentificationController.cs* :

```

/// <summary>
/// Récupère l'utilisateur cible en BDD
/// </summary>
/// <param name="utilisateur">l'utilisateur cible</param>
/// <returns>true si l'utilisateur a été trouvé (retour != null)</returns>
1 référence
public string ControleAuthentification(Utilisateur utilisateur)
{
    String jsonRecupererUtilisateur = JsonConvert.SerializeObject(utilisateur);
    try
    {
        List<Utilisateur> utilisateurs = TraitementRecup<Utilisateur>(GET, "utilisateur/" + jsonRecupererUtilisateur);
        if (utilisateurs != null && utilisateurs.Count > 0)
        {
            // récupération de l'utilisateur authentifié
            Utilisateur utilisateurCible = utilisateurs[0];
            // mise à jour l'objet utilisateur avec l'idService
            utilisateur.IdService = utilisateurCible.IdService;
            return utilisateur.IdService;
        }
        else
        {
            return utilisateur.IdService = null;
        }
    }
    catch (Exception ex)
    {
    }
}

```

Côté API REST, nous ajoutons une ligne dans le traitement du verbe HTTP GET afin de récupérer le contenu concernant l'utilisateur cible. En effet, cette requête est différente des autres car nous recherchons ici un contenu :

```
// traitement suivant le verbe HTTP utilisé
if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    if ($contenu != "") {
        $controle->getUtilisateur($contenu);
    } else {
        $controle->get($table, $id);
    }
}
```

Afin d'adapter cette nouvelle condition GET, nous écrivons la fonction *getUtilisateur* dans *Controle.php* :

```
/**
 * Requête arrivée en GET (select)
 * @param type $contenu de la table utilisateur
 */
public function getUtilisateur($contenu) {
    $result = $this->accessBDD->selectUtilisateur($contenu);
    if($result == null || $result == false) {
        $this->reponse(400, "requete invalide");
    } else {
        $this->reponse(200, "OK", $result);
    }
}
```

Ce qui nous permet d'initialiser une nouvelle fonction *selectUtilisateur* dans *Access.php* qui prend en paramètre un contenu, le login et le mot de passe de l'utilisateur à récupérer en base de données :

```
/**
 * Récupère les informations d'un utilisateur cible
 * @param type $contenu contenu d'identification de l'utilisateur ciblé
 * @return lignes de la requête
 */
public function selectUtilisateur($contenu) {
    $param = [
        "login" => $contenu["Login"],
        "pwd" => $contenu["Pwd"]
    ];
    $req = "Select u.idService ";
    $req .= "from utilisateur u ";
    $req .= "where login= :login and pwd= :pwd";
    return $this->conn->query($req, $param);
}
```

De ce fait, il est nécessaire de créer deux nouvelles tables dans la base de données afin de correspondre aux requêtes api concernant les services et utilisateurs, avec une clef étrangère dans la table utilisateur :

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/> 1	id	int			Non	Aucun(e)		AUTO_INCREMENT	Modifier  Supprimer  Plus
<input type="checkbox"/> 2	login	varchar(30)	utf8mb4_0900_ai_ci		Oui	NULL			Modifier  Supprimer  Plus
<input type="checkbox"/> 3	pwd	varchar(100)	utf8mb4_0900_ai_ci		Oui	NULL			Modifier  Supprimer  Plus
<input type="checkbox"/> 4	idService	varchar(10)	utf8mb4_0900_ai_ci		Oui	NULL			Modifier  Supprimer  Plus

Afin que chaque utilisateur soit lié à un service :

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/> 1	id	varchar(30)	utf8mb4_0900_ai_ci		Non	Aucun(e)			Modifier  Supprimer  Plus
<input type="checkbox"/> 2	libelle	varchar(60)	utf8mb4_0900_ai_ci		Oui	NULL			Modifier  Supprimer  Plus

Certains éléments graphiques ne doivent pas être visibles pour le service "50002". Pour ce faire, nous envoyons en paramètre idService au format string dans le constructeur à *FrmMediatek.cs*. Soit le frame des échéances d'abonnement sont affichées, soit dans le cas du service "50002" l'application principale s'exécute :

```

/// <summary>
/// Constructeur : création du contrôleur lié à ce formulaire
/// </summary>
1 référence
public FrmMediatek(string idService)
{
    InitializeComponent();
    this.controller = new FrmMediatekController();
    if (idService != null)
    {
        if (idService == "50001" || idService == "50000")
        {
            FrmEcheancesAbos frmEcheancesAbos = new FrmEcheancesAbos();
            frmEcheancesAbos.ShowDialog();
        }
        else if (idService == "50002")
        {
            AfficherInfosServicePrets();
        }
    }
}

```

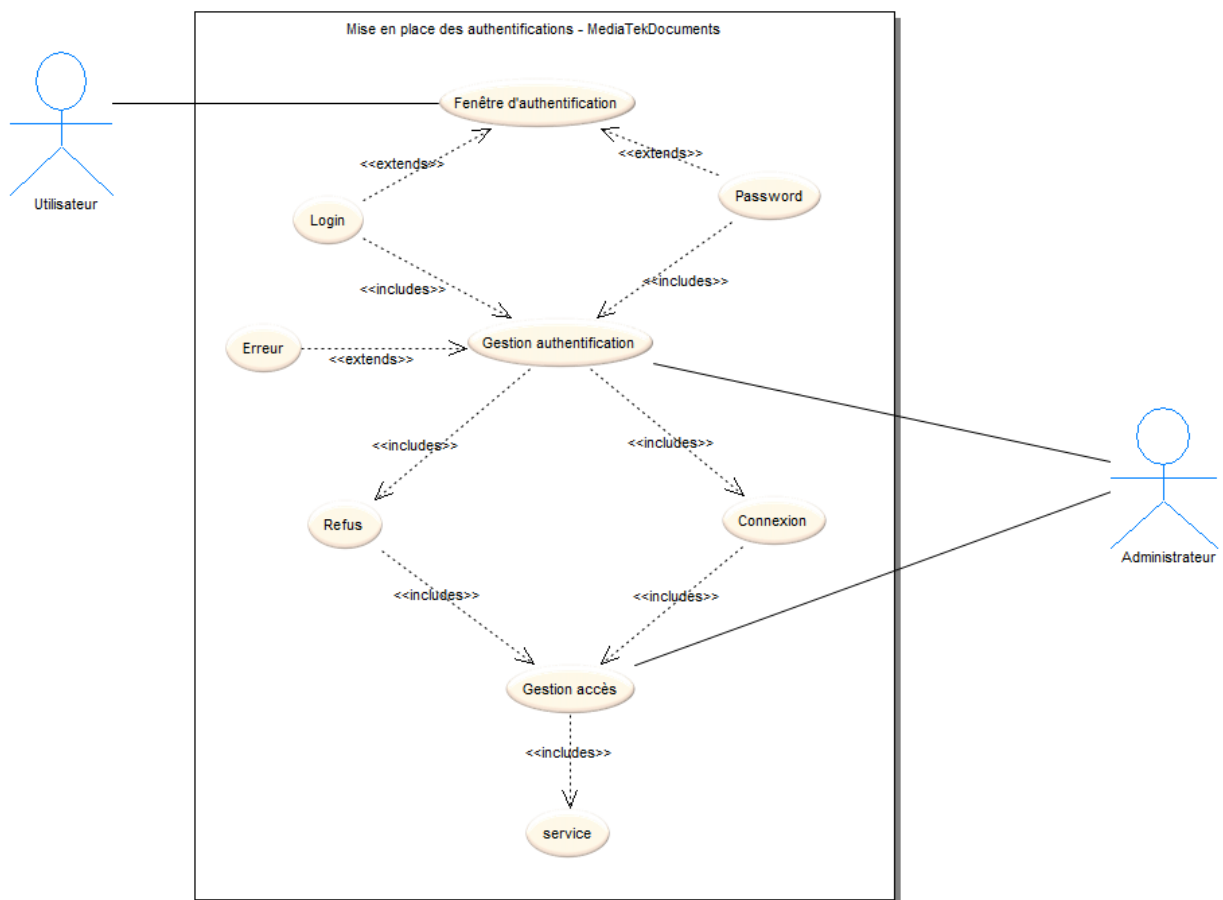
A noter la méthode *AfficherInfosServicePrets* qui permet de n'afficher que les pages de gestion des livres, dvd et revues :

```

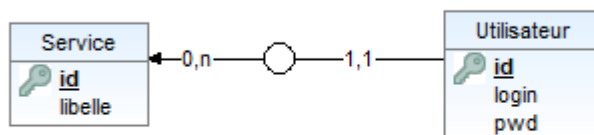
/// <summary>
/// Afficher uniquement les pages livres, DVD et revues
/// Si service prêts
/// </summary>
1 référence
private void AfficherInfosServicePrets()
{
    tabOngletsApplication.TabPages.Remove(tabReceptionRevue);
    tabOngletsApplication.TabPages.Remove(tabCmdLivres);
    tabOngletsApplication.TabPages.Remove(tabCmdDvd);
    tabOngletsApplication.TabPages.Remove(tabCmdRevues);
}

```

## Diagramme de cas d'utilisation



## MCD des nouvelles tables *service* et *utilisateur*



## Mission 5 : assurer la sécurité, la qualité et intégrer des logs

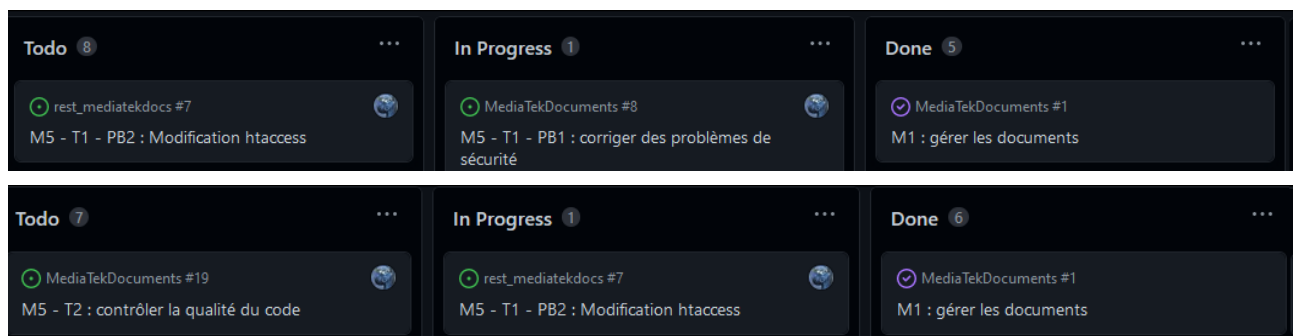
### Tâche 1 : corriger des problèmes de sécurité

- Sécuriser l'accès à l'API.
- Bloquer l'accès HTTP en racine de l'API.

Temps de travail estimé  
réel  
3 heures

Temps de travail  
  
2 heures

### Kanban de la tâche actuelle "In Progress"



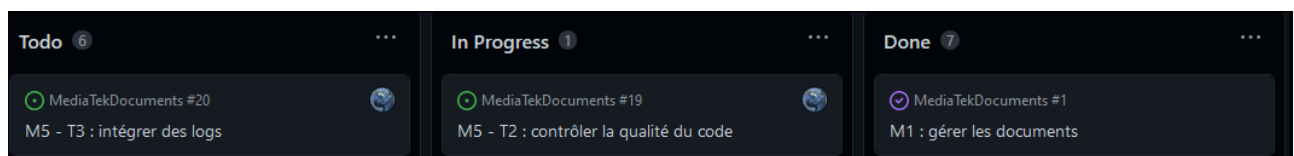
### Tâche 2 : contrôler la qualité

- Mettre en place, en local, un serveur SonarQube et le lier avec l'application C#.
- Corriger les problèmes relevés par SonarLint.
- Contrôler les messages dans SonarQube.

Temps de travail estimé  
réel  
3 heures

Temps de travail  
  
4 heures

### Kanban de la tâche actuelle "In Progress"



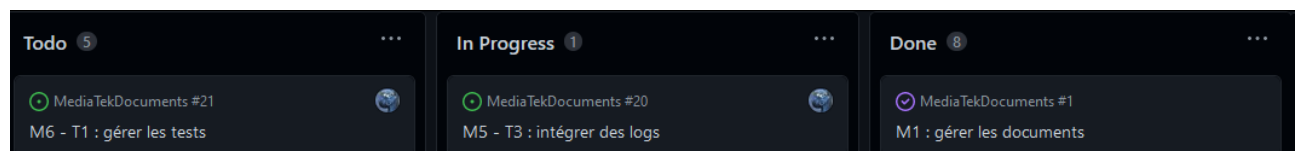
### Tâche 3 : intégrer des logs

- Ajouter le code de configuration des logs.
- Ajouter les logs au niveau de chaque affichage console, à enregistrer dans un fichier de logs.

Temps de travail estimé  
réel  
2 heures

Temps de travail  
  
3 heures

### Kanban de la tâche actuelle "In Progress"



## Tâche 1

Concernant le problème n°1, c'est à dire sécuriser l'authentification qui s'effectuait jusqu'alors en dur dans le code de l'application de la classe *Access.cs*), nous modifions *App.config* afin d'enlever les informations de connexion :

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
  <add name="MediaTekDocuments.Properties.Settings.mediatek86ConnectionString"
    connectionString="admin:adminpwd" providerName="MySql.Data.MySqlClient" />
  </connectionStrings>
</configuration>
```

Actualisation de la déclaration de *connectionName* afin de correspondre à son nouvel intitulé, dans *Access.cs*, et une redirection vers l'api via *uriApi* :

```
/// <summary>
/// Chaîne de connexion référence externe
/// </summary>
private static readonly string connectionName = "MediaTekDocuments.Properties.Settings.mediatek86ConnectionString";

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

Afin de récupérer la chaîne de connexion, nous écrivons la méthode *GetConnectionStringByName* qui attend en paramètre une chaîne de connexion pour lire les données à partir de la BDD :

```
static string GetConnectionStringByName(string name)
{
    string returnValue = null;
    ConnectionStringSettings settings = ConfigurationManager.ConnectionStrings[name];
    if (settings != null)
    {
        returnValue = settings.ConnectionString;
    }
    return returnValue;
}
```

Nous pouvons maintenant l'utiliser dans notre nouvelle méthode afin d'initialiser la demande de connexion à l'API REST :

```
/// <summary>
/// Méthode privée pour créer un singleton
/// Initialise l'accès à l'API
/// </summary>
1 référence
private Access()
{
    string connectionString;
    try
    {
        connectionString = GetConnectionStringByName(connectionName);
        api = ApiRest.GetInstance(uriApi, connectionString);
    }
    catch (Exception e)
```

Concernant le problème n°2, et afin de bloquer la lisibilité en racine de l'API, nous utilisons le flag [L] afin de créer un break dans la requête, donc une requête invalide :

```
RewriteEngine on  
RewriteRule ^$ mediatekdocuments.php [L]
```

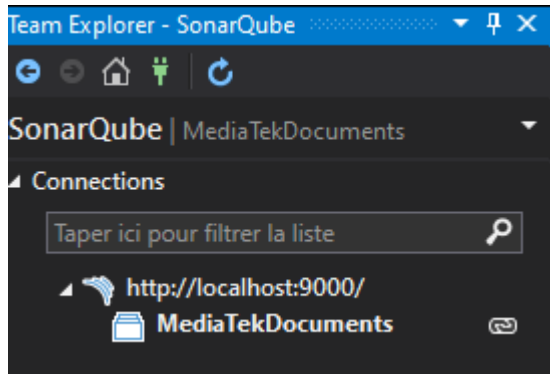
Résultat:

JSON	Données brutes	En-têtes
<div>Enregistrer Copier Tout réduire Tout développer Filtre le JSON</div> <div>code: 401 message: "authentification incorrecte" result: ""</div>		

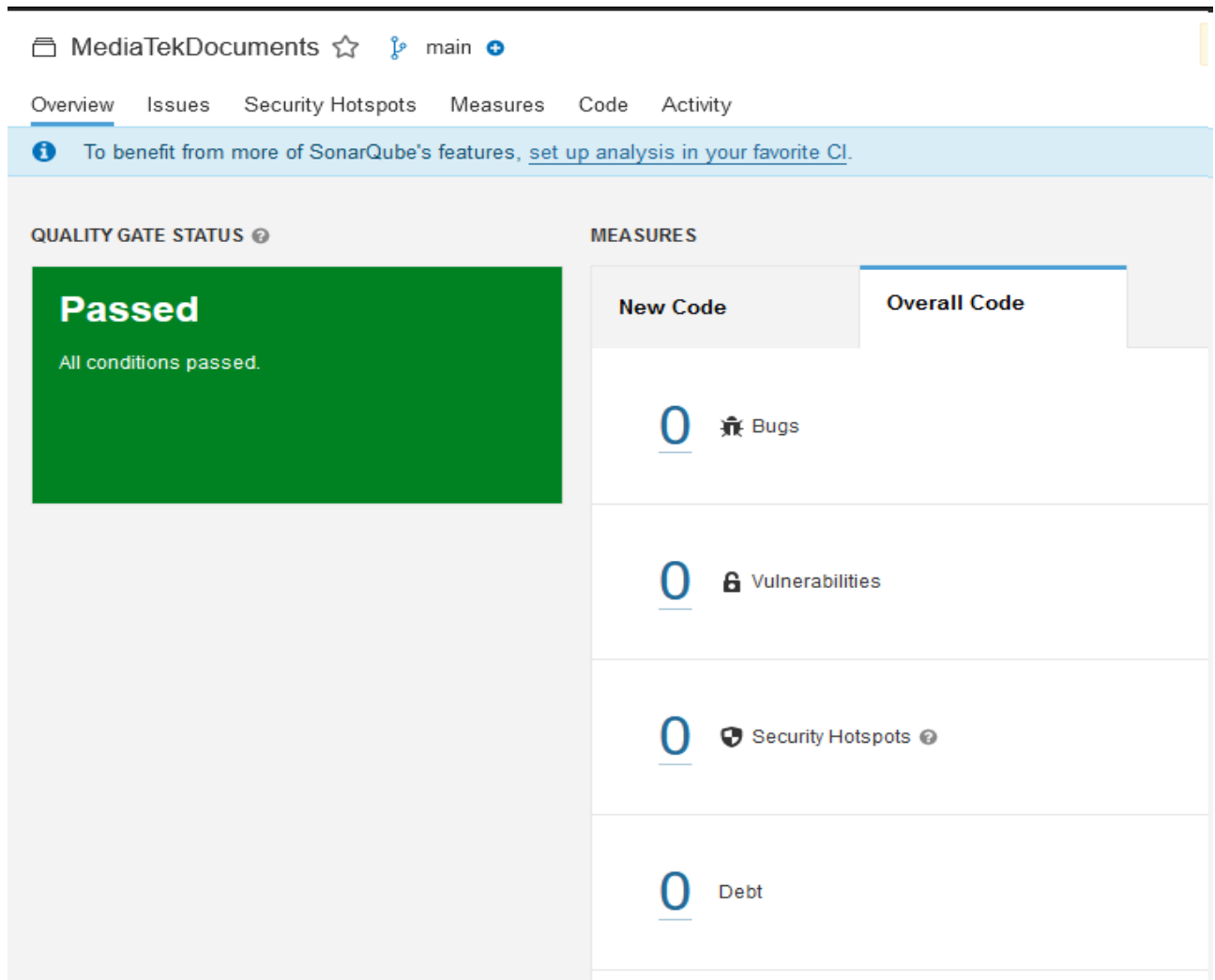


## Tâche 2

Nous mettons en place en local un serveur SonarQube qui est lié sous Visual Studio Code 2019 à notre projet :



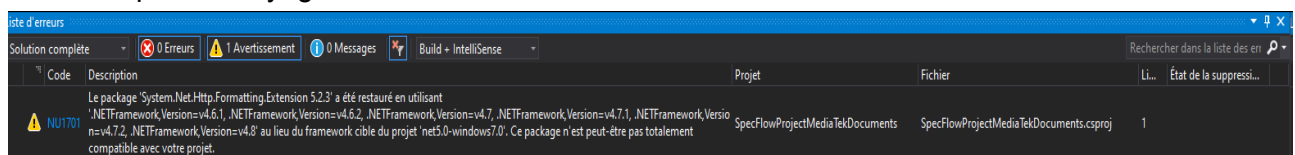
Le statut du projet sous SonarQube :



The screenshot shows the SonarQube web interface for the project 'MediaTekDocuments'. The 'Overview' tab is selected, displaying a green 'Passed' status for the quality gate. The 'MEASURES' section shows zero issues across four categories: Bugs, Vulnerabilities, Security Hotspots, and Debt.

Category	Count
Bugs	0
Vulnerabilities	0
Security Hotspots	0
Debt	0

Résultat après nettoyage dans le code suite aux avertissements SonarLint :



## Tâche 3




Configuration d'un logger dans `Access.cs`. Le niveau d'erreur n'a pas été configuré, sa valeur par défaut correspondant à nos attentes :

```
/// <summary>
/// Méthode privée pour créer un singleton
/// Initialise l'accès à l'API
/// </summary>
1 référence
private Access()
{
    String connectionString;
    try
    {
        connectionString = GetConnectionStringByName(connectionName);
        //connectionString = "admin:adminpwd";
        Log.Logger = new LoggerConfiguration()
            .WriteTo.Console()
            .WriteTo.File(new JsonFormatter(), "logs/log.txt",
                rollingInterval: RollingInterval.Day)
            .CreateLogger();
        api = ApiRest.GetInstance(uriApi, connectionString);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        Log.Fatal("Access.Access catch connectionString={0} error={1}", connectionName, e);
        Environment.Exit(0);
    }
}
```

Une sortie de log lors d'une erreur dans l'application :

```
16:43:50 ERR] Access.TraitementRecup catch liste=[] error=System.Net.Http.UnsupportedMediaTypeException:
```

Ainsi que leur emplacement dans le folder `bin\Debug\` :

Atelier 3 > MediaTekDocuments > MediaTekDocuments > bin > Debug > logs					Rechercher dans
Nom	Statut	Modifié le	Type	Taille	
 log20230414.txt	 	14/04/2023 16:43	Document texte	18 Ko	

## Mission 6 : tester et documenter

### Tâche 1 : gérer les tests

- Écrire les tests unitaires sur les classes du package Model.
- Écrire les tests fonctionnels sur les recherches dans l'onglet des livres.
- Construire une collection de tests dans Postman pour contrôler les fonctionnalités de l'API d'accès à la BDD.

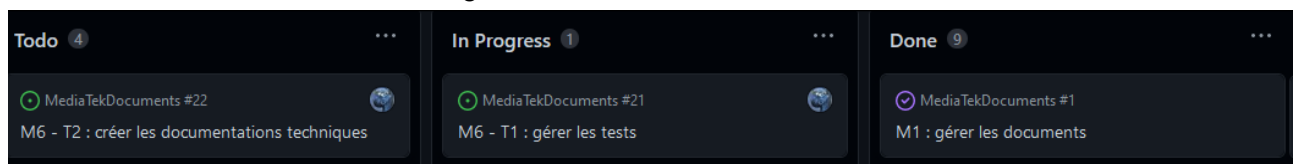
Temps de travail estimé  
réel

5 heures

Temps de travail

20 heures

### Kanban de la tâche actuelle "In Progress"



### Tâche 2 : créer les documentations techniques

- Contrôler, dans chaque application, que les commentaires normalisés sont présents et corrects.
- Générer les documentations techniques des deux applications : sous format HTML pour l'API et sous format chm pour l'application C#.

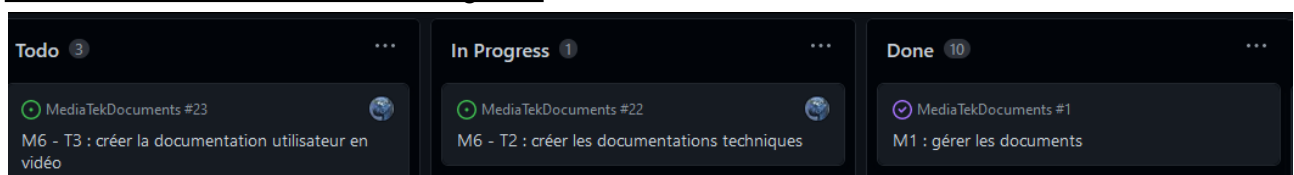
Temps de travail estimé  
réel

1 heures

Temps de travail

2 heures

### Kanban de la tâche actuelle "In Progress"



### Tâche 3 : créer la documentation utilisateur en vidéo

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

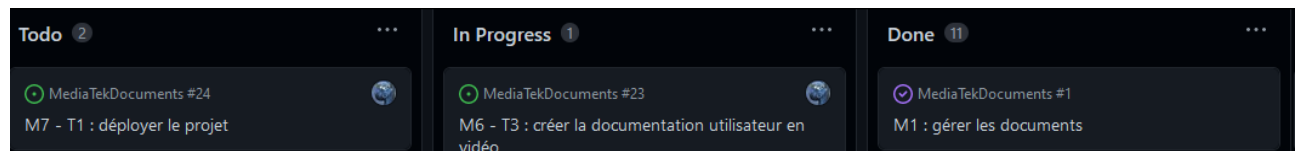
Temps de travail estimé  
réel

2 heures

Temps de travail

6 heures

### Kanban de la tâche actuelle "In Progress"

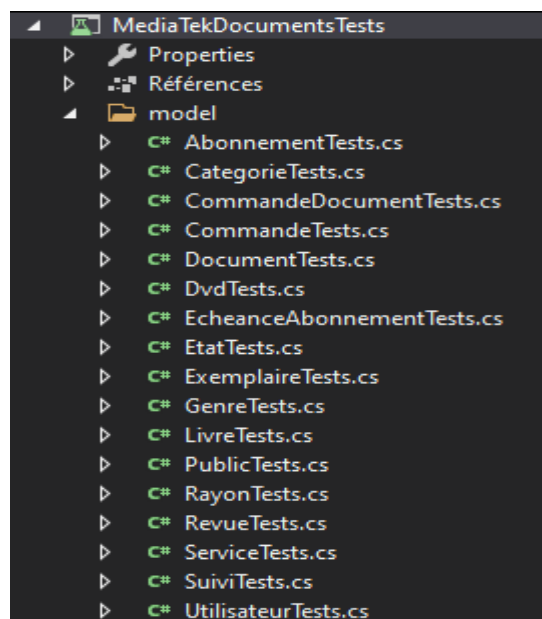


## Tâche 1

### Les tests unitaires

Création d'un nouveau projet de test UNIT dans la même solution que l'on nomme *MediaTekDocumentsTests* ainsi que deux sous-dossiers en racine de ce nouveau projet, *model* et *view*.

On lie ensuite ce projet à *MediaTekDocuments* en y ajoutant une référence de projet et ce dans la fenêtre du gestionnaire de référence.



Toutes les classes partie modèle sont créées afin de tester leurs constructeurs un à un en respectant la règle des 3 A. Par exemple, le constructeur de la classe **AbonnementTests.cs** est testé. Toutes les autres classes reprennent la même logique :

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;

namespace MediaTekDocuments.model.Tests
{
    [TestClass()]
    0 références
    public class AbonnementTests
    {
        private const string id = "12315";
        private static readonly DateTime dateCommande = new DateTime(2023, 03, 30);
        private const double montant = 1230;
        private static readonly DateTime dateFinAbonnement = new DateTime(2023, 05, 02);
        private const string idRevue = "10001";
        private static readonly Abonnement abonnement = new Abonnement(id, dateCommande, montant, dateFinAbonnement, idRevue);

        [TestMethod()]
        0 références
        public void AbonnementTest()
        {
            Assert.AreEqual(id, abonnement.Id, "devrait réussir : id valorisé");
            Assert.AreEqual(dateCommande, abonnement.DateCommande, "devrait réussir : date de commande initialisée");
            Assert.AreEqual(montant, abonnement.Montant, "devrait réussir : montant valorisé");
            Assert.AreEqual(dateFinAbonnement, abonnement.DateFinAbonnement, "devrait réussir : date de fin d'abonnement initialisée");
            Assert.AreEqual(idRevue, abonnement.IdRevue, "devrait réussir : id revue valorisé");
        }
    }
}

```

Nous régénérons la solution et observons les résultats dans l'explorateur de tests :

Test	Durée
MediaTekDocumentsTests (20)	226 ms
MediaTekDocuments.model.Tests (...)	226 ms
AbonnementTests (1)	225 ms
CategorieTests (2)	< 1 ms
CommandeDocumentTests (1)	1 ms
CommandeTests (1)	< 1 ms
DocumentTests (1)	< 1 ms
DvdTests (1)	< 1 ms
EcheanceAbonnementTests (1)	< 1 ms
EtatTests (1)	< 1 ms
ExemplaireTests (1)	< 1 ms
GenreTests (1)	< 1 ms
LivreTests (1)	< 1 ms
PublicTests (1)	< 1 ms
RayonTests (1)	< 1 ms
RevueTests (1)	< 1 ms
ServiceTests (1)	< 1 ms
SuiviTests (2)	< 1 ms
UtilisateurTests (1)	< 1 ms

**Récapitulatif des détails du test**

✓ AbonnementTest

Source: [AbonnementTests.cs](#) ligne 18

⌚ Durée: < 1 ms

Nous effectuons également un test de la méthode *ToString* présente dans la classe métier *Categorie.cs* afin de vérifier que les deux valeurs passées en paramètre correspondent et soient de bien des chaînes de caractères :

```

[TestClass()]
0 références
public class CategorieTests
{
    private const string id = "00001";
    private const string libelle = "Testons";
    private static readonly Categorie categorie = new Categorie(id, libelle);

    [TestMethod()]
    0 références
    public void CategorieTest()...

    [TestMethod()]
    0 références
    public void ToStringTest()
    {
        Assert.AreEqual(libelle.ToString(), categorie.Libelle.ToString(), "devrait réussir : libelle en string");
    }
}

```

Résultat du test valide pour la méthode de test unitaire *ToStringTest()* :

Test	Durée	Caractéris...	Message d'erreur
MediaTekDocumentsTests (21)	8 ms		
MediaTekDocuments.model.Tests (...)	4 ms		
AbonnementEcheanceTests (1)	< 1 ms		
AbonnementTests (1)	< 1 ms		
CategorieTests (2)	1 ms		
CategorieTest	< 1 ms		
ToStringTest	1 ms		

Récapitulatif des détails du test

✓ ToStringTest

Source: [CategorieTests.cs](#) ligne 26

Durée: 1 ms

### Les tests fonctionnels :

Configuration en amont notre IDE avec l'installation de l'extension SpecFlow qui va nous permettre de gérer les tests fonctionnels.

On peut alors ajouter un nouveau projet SpecFlow dans la même solution et faisons référence à notre projet *MediaTekDocuments* via les dépendances.

Une fois le projet SpecFlow créé , nous installons également les packages NuGet :

- Newtonsoft.Json
- System.Net.Http
- System.Net.Http.Formatting.Extension

Nous effectuons un premier test avec le code auto généré afin d'obtenir le lien externe vers SpecFlow qui nous permettra de s'identifier ou de créer un compte le cas échéant.

On écrit dans un premier temps les scénarios, dans le sous-dossier *Features*. Les différents scénarios concernent les fonctionnalités de recherche présentes dans l'onglet Livres de l'application :

```

1 Feature: RechercherOngletLivres
2 Recherche d'un ou plusieurs livres avec Les différents outils de recherche
3 Via L'onglet Livres
4
5 Scenario: Rechercher un livre par titre
6     Given Je sélectionne l'onglet Livres
7     When Je saisi une partie de titre "Le"
8     Then Le nombre de livre(s) obtenu est de 11
9
10 Scenario: Afficher les livres par genre
11     Given Je sélectionne l'onglet Livres
12     When Je sélectionne le genre 2
13     Then Le nombre de livre(s) obtenu est de 5
14
15 Scenario: Afficher les livres par public
16     Given Je sélectionne l'onglet Livres
17     When Je sélectionne le public 3
18     Then Le nombre de livre(s) obtenu est de 10
19
20 Scenario: Afficher les livres par rayon
21     Given Je sélectionne l'onglet Livres
22     When Je sélectionne le rayon 7
23     Then Le nombre de livre(s) obtenu est de 7
24

```

Nous générons les étapes automatiquement dans le sous-dossier *StepDefinitions*. Celles-ci sont vides et nécessitent différents lignes de code, toujours dans l'optique du respect du standard des 3A (Arrange-Act-Assert). En amont, nous importons *FrmMediatek* et l'initialisons afin de pouvoir réaliser nos tests dans ce frame. Ce frame recevant *idService* en paramètre, nous créons également un nouveau constructeur sans paramètres afin de pouvoir travailler sans celui-ci :

```

[Binding]
0 références
public class RechercherOngletLivresSteps
{
    // Initialisation du FrmMediatek, cible des tests fonctionnels
    public readonly FrmMediatek frmMediatek = new();
}

```

Ci dessous, on prend pour exemple le test fonctionnel du premier scénario "rechercher un livre par titre" :

Given "je sélectionne l'onglet Livres" :

- on retrouve le chemin graphique du choix d'un onglet présent dans le groupe *tabOngletsApplications*.
- on active le frame *FrmMediatek.cs* en le passant à true.
- l'onglet sélectionné est comme requis *tabLivres*.

```

/// <summary>
/// Sélectionne l'onglet Livres afin de correspondre avec les requêtes ciblées
/// </summary>
[Given(@"Je sélectionne l'onglet Livres")]
0 références
public void GivenJeSelectionneLOngletLivres()
{
    TabControl tabonglet = (TabControl)frmMediatek.Controls["tabOngletsApplication"];
    frmMediatek.Visible = true;
    tabonglet.SelectedTab = (TabPage)frmMediatek.Controls["tabLivres"];
}

```

When "je saisi une partie de titre "Le"":

- on retrouve le chemin graphique du choix du textbox permettant de renseigner une partie du titre du livre recherché.
- on lui assigne la valeur recherchée et indiquée dans le scénario grâce à la valorisation du paramètre *valeur*.

```

/// <summary>
/// Récupère une partie du titre cible
/// </summary>
/// <param name="valeur">le titre du livre recherché</param>
[When(@"Je saisi une partie de titre "(.*)")]]
0 références
public void WhenJeSaisiUnePartieDeTitre(string valeur)
{
    TextBox txbLivresTitreRecherche = (TextBox)frmMediatek.Controls["tabOngletsApplication"].Controls["tabLivres"].
        Controls["grpLivresRecherche"].Controls["txbLivresTitreRecherche"];
    txbLivresTitreRecherche.Text = valeur;
}

```

Then "le nombre de livre(s) obtenu est de 11" :

- on récupère le chemin graphique du datagrid correspondant, *dgvLivresListe*.
- on initialise le résultat du nombre de lignes trouvées dans *resultat*.
- et on effectue un Assert afin de comparer le résultat présent dans l'écriture du scénario et celui récupéré en base de données.

```

/// <summary>
/// Récupère le résultat disponible dans la datagrid Livres
/// </summary>
/// <param name="valeur">valeur cible</param>
[Then(@"Le nombre de livre(s) obtenu est de (.*)")]
0 références
public void ThenLeNombreDeLivreSObtenuEstDe(int valeur)
{
    DataGridView dgvLivresListe = (DataGridView)frmMediatek.Controls["tabOngletsApplication"].Controls["tabLivres"].
        Controls["grpLivresRecherche"].Controls["dgvLivresListe"];
    int resultat = dgvLivresListe.Rows.Count;
    Assert.AreEqual(valeur, resultat);
}

```

Le résultat du test après génération du projet :

**Récapitulatif des détails du test**

RechercherUnLivreParTitre

Source: [RechercherOngletLivres.feature](#) ligne 5

Durée: 324 ms

Sortie standard:

```

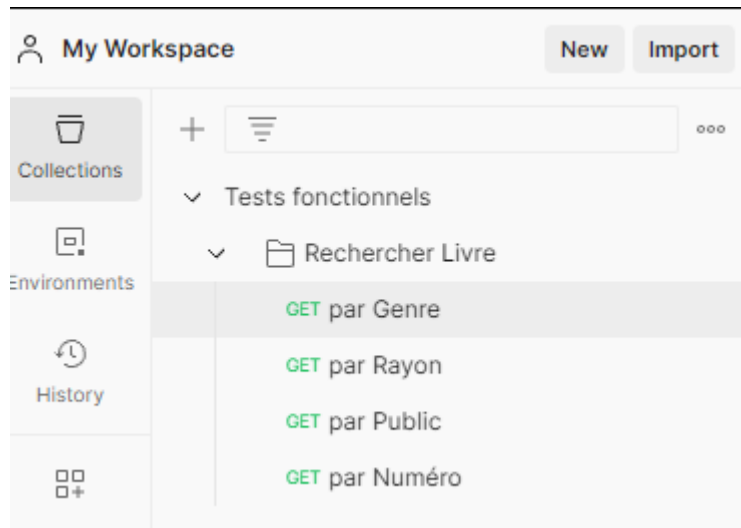
Given Je sélectionne l'onglet Livres
-> done: RechercherOngletLivresSteps.GivenJeSelectionneLOngletLivres() (0,3s)
When Je saisi une partie de titre "Le"
-> done: RechercherOngletLivresSteps.WhenJeSaisiUnePartieDeTitre("Le") (0,0s)
Then Le nombre de livre(s) obtenu est de 11
-> done: RechercherOngletLivresSteps.ThenLeNombreDeLivreSObtenuEstDe(11) (0,0s)

```

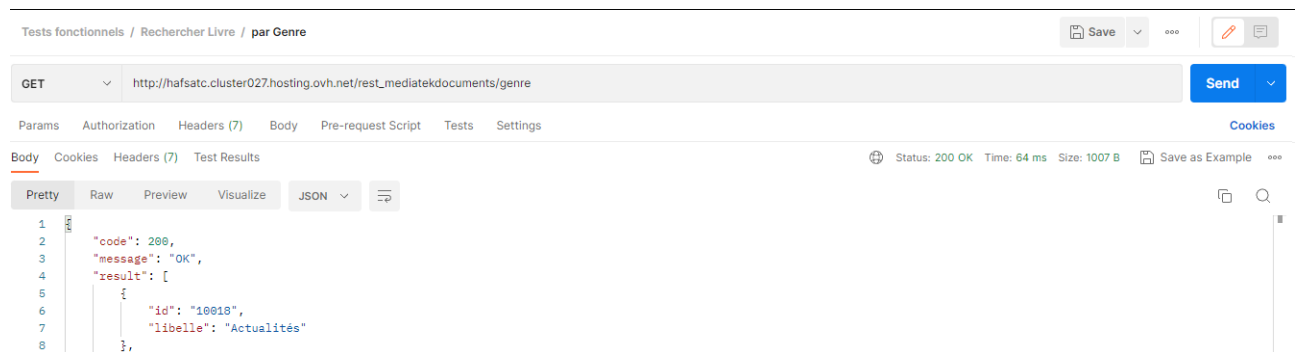
Postman



Sous Postman, création d'un dossier *Tests fonctionnels* (et sous-dossier *Rechercher Livre*) afin d'organiser la collection de tests :

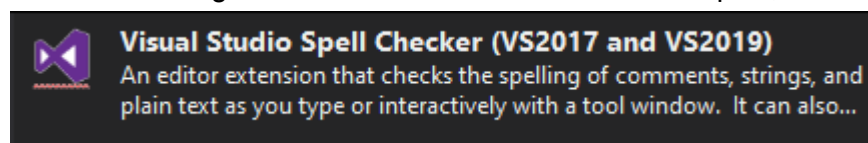


Ainsi qu'un exemple de la requête *GET* sur Genre :



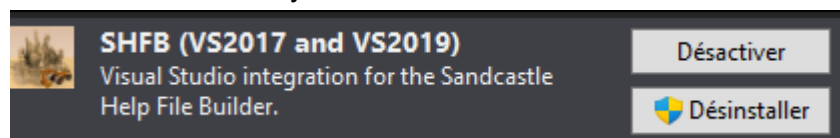
## Tâche 2

En amont de la génération des documentations techniques, nous installons :



Dans le but d'effectuer une double vérification de nos commentaires normalisés.

Une fois le code nettoyé, nous installons l'extension SHFB :



Et configurons dans les propriétés de notre projet la sortie de la documentation technique en XML :

Sortie

Chemin de sortie :

☒ Fichier de documentation XML :

Et construisons à l'aide de l'outil SHFB notre documentation technique. Voici le rendu pour la classe *FrmMediatek.cs* par exemple :

## FrmMediatekController Class

Contrôleur lié à FrmMediatek

### ▼ Definition

**Namespace:** MediaTekDocuments.controller

**Assembly:** MediaTekDocuments (in MediaTekDocuments.exe) Version: 1.0.0.0 (1.0.0.0)

C#

Copy

```
internal class FrmMediatekController
```

Inheritance [Object](#) → [FrmMediatekController](#)

### ▼ Constructors

[FrmMediatekController](#)

Récupération de l'instance unique d'accès aux données

### ▼ Methods

<a href="#">CreerAbonnement</a>	Créer une commande dans la BDD
<a href="#">CreerAbonnementRevue</a>	Créer un abonnement revue en BDD
<a href="#">CreerCommandeDocument</a>	Créer une commande document dans la BDD
<a href="#">CreerDocument</a>	Crée un document dans la BDD
<a href="#">CreerDvd</a>	Crée un DVD dans la BDD
<a href="#">CreerExemplaire</a>	Créer un exemplaire d'une revue dans la BDD

Concernant la documentation de l'API REST, nous utilisons l'outil intégré à Netbeans afin de générer la documentation technique. Une erreur console est contournée en reprenant le message et en retirant l'expression “*--progressbar*” et en exécutant cette nouvelle ligne de commande via le système d'exploitation.

Rendu pour la classe *AccessBDD.php* :

## Packages

Application

## Reports

Deprecated

Errors

Markers

## Indices

Files

Application

## AccessBDD

[AccessBDD.php](#) : 8

in package Application

*Classe de construction des requêtes SQL à envoyer à la BDD*

### Table of Contents

**P** [\\$bdd](#) : mixed**P** [\\$conn](#) : mixed**P** [\\$login](#) : mixed**P** [\\$mdp](#) : mixed**P** [\\$port](#) : mixed**P** [\\$serveur](#) : mixed**M** [\\_\\_construct\(\)](#) : mixed*Constructeur : demande de connexion à la BDD***M** [deleteAbonnement\(\)](#) : mixed*Suppression d'un abonnement***M** [deleteCommandeDocument\(\)](#) : mixed*Suppression d'une commande document*

## Tâche 3

[Lien](#) vers la documentation utilisateur. Mes excuses pour la qualité.

## **Mission 7 : déployer et gérer les sauvegardes de données**

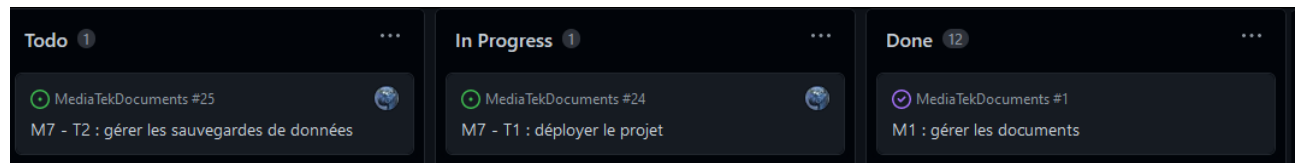
### Tâche 1 : déployer le projet

- Déployer l'API et la BDD.
- Tester l'API avec Postman.
- Créer et tester un installateur pour l'application C#.

Temps de travail estimé  
réel  
3 heures

Temps de travail  
  
20 heures

#### Kanban de la tâche actuelle "In Progress"



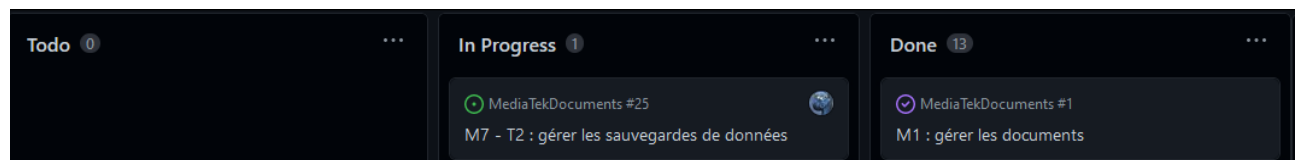
#### Tâche 2 : gérer les sauvegardes des données

- Une sauvegarde journalière automatisée doit être programmée pour la BDD.
- La restauration pourra se faire manuellement, en exécutant le script de sauvegarde.

Temps de travail estimé  
réel  
1 heure

Temps de travail  
  
2 heures

#### Kanban de la tâche actuelle "In Progress"



#### Tâche 1

#### Processus

Choix de OVH Cloud pour l'hébergement suite à un essai infructueux du côté de PlanetHoster pour la redirection HTTP au niveau des verbes PUT et DELETE.

Dans un premier temps nous créons une base de données chez le fournisseur et exportons la BDD locale sur laquelle nous travaillons depuis le début du projet (via wamp) vers le phpMyAdmin distant. La version locale n'étant pas compatible avec la version OVHCloud (problème de collations), il faut également configurer l'encodage de celle-ci.

Un compte FTP est également créé et utilisé afin de téléverser par l'intermédiaire de FileZilla l'API chez l'hébergeur.

Nous recopions ensuite les identifiants de connexion nouvellement définis (non disponibles ici) dans la classe *AccessBDD.php* de l'API REST :

### Visualiser le fichier AccessBDD.php



```
1. <?php
2.
3. include_once("ConnexionPDO.php");
4.
5. /**
6.  * Classe de construction des requêtes SQL à envoyer à La BDD
7.  */
8. class AccessBDD {
9.
10.     public $login = " ";
11.     public $mdp = " ";
12.     public $bd = " ";
13.     public $serveur = "hafsatcadmin.mysql.db";
14.     public $port = "3306";
15.     public $conn = null;
```

Et l'adresse du serveur dans la classe **Access.cs** de l'application bureau C#, l'adresse locale étant gardée en commentaire en cas de nécessité :

```
/// <summary>
/// Adresse de l'API
/// </summary>
private static readonly string uriApi = "http://hafsatc.cluster027.hosting.ovh.net/rest_mediatekdocuments/";

/* private static readonly string uriApi = "http://localhost/rest_mediatekdocuments/";*/
```

Il aura également fallu un certain nombre d'heures afin d'arriver à ce que notre **.htaccess** téléversé permette l'authentification en distant lorsque nous exécutons l'application C# en subissant successivement des erreurs 500 puis 401.

Finalement un seul flag est à insérer dans ce document :

```
RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]
```

## Tâche 2

Création d'un document texte afin de rédiger un script de sauvegarde automatisé. Il sera sauvegardé en .php, et basé sur ce modèle :

## OVH - Base de données - Sauvegarde type GZIP - Toutes les tables

```
<?php
error_reporting(E_ALL); // Activer le rapport d'erreurs PHP

$db_charset = "latin1"; /* mettre utf8 ou latin1 */

$db_server      = "xxxxxx"; // Nom du serveur MySQL. ex. mysql5-26.perso
$db_name        = "xxxxxx"; // Nom de la base de données. ex. mabase
$db_username    = "xxxxxx"; // Nom de la base de données. ex. mabase
$db_password    = "xxxxxx"; // Mot de passe de la base de données.

$cmd_mysql = "mysqldump";

$archive_GZIP = "sauve_base_format_gzip.gz";

echo " Sauvegarde de la base <font color=red><b>$db_name</b></font> par
<b>mysqldump</b> dans le fichier <b>$archive_GZIP</b> <br> \n";
$commande = $cmd_mysql." --host=$db_server --user=$db_username
--password=$db_password -C -Q -e --default-character-set=$db_charset $db_name |
gzip -c > $archive_GZIP ";
$SCR_exec = system($commande);

if (file_exists($archive_GZIP))
{
    $Taille_Sauve = filesize($archive_GZIP);
    echo " Sauvegarde effectuée dans &nbsp; &nbsp; <b>$archive_GZIP</b> &nbsp; &nbsp;
&nbsp; de taille &nbsp; &nbsp; <b>".$Taille_Sauve."</b> Ko</font> <br> \n";
}

echo " Fin de la Sauvegarde <b>GZIP</b> de la <u>totalité de la base</u> <font
color=red><b>".$db_name."</b></font> <i>(depuis le serveur SQL <font color=red>
<b>".$db_server."</b></font></i> <br> \n";

?>
```

Nous créons un sous-dossier savebdd dans le FTP de notre hébergeur et téléversons le document backup.php, en modifiant les droits d'accès au fichier à la valeur numérique 777 (tous droits pour tout le monde).

Nous pouvons maintenant rédiger une tâche automatisée (CRON) avec le chemin correspondant au dossier de téléversement du script, la version PHP et la fréquence demandée :

### Tâches planifiées - Cron

 Besoin d'aide pour configurer vos tâches planifiées ? Consultez nos guides en ligne. [🔗](#)

Commande	Description	Fréquence	Langage	État	E-mail	
www/savebdd/backup.php	Sauvegarde journalière de la bdd	54 ****	PHP 7.4	Activé	Contact administrateur	...

Mode opératoire de restauration de la base de données :

Rendez-vous dans la page de gestion de votre hébergement puis sur FTP-SSH, et cliquez à l'endroit indiqué :

Renouvellement automatique prévu en **avr. 2024**[Informations générales](#)[Multisite](#)[Modules en 1 clic](#)[Statistiques et logs](#)[FTP - SSH](#)[Bases de données](#)[Tâches en cours](#)[Plus ▾](#)

Ces paramètres vous permettront de mettre votre site en ligne

✚ Besoin d'aide pour accéder à l'espace de stockage de votre hébergement web ? [Consultez nos guides en ligne.](#)

Serveur FTP et SFTP :

Port FTP : 21  
Port SFTP : 22

Lien FTP vers le cluster :

Chemin du répertoire home :

[Restaurer une sauvegarde](#)[FTP Explorer](#)

Cliquer sur FTP Explorer

Identifiez vous avec votre nom d'utilisateur et mot de passe FTP :

Serveur FTP **ftp.cluster027.hosting.ovh.net**

Nom d'utilisateur

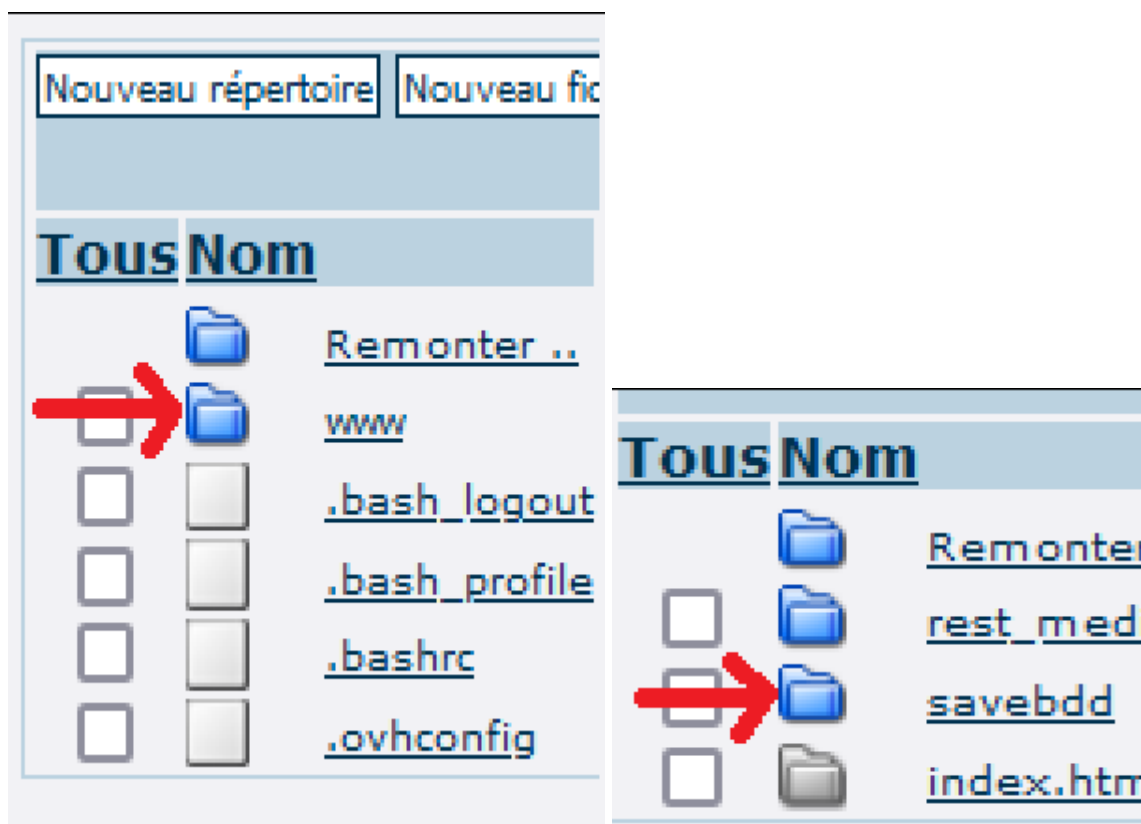
Mot de passe

Langue French ▾

Mode FTP ☐ Binary ☒ Automatique

[Effacer les cookies](#)

Cliquez aux emplacements marqués d'une flèche rouge (suite du processus page suivante) :

Puis télécharger le dossier **save\_base\_format\_gzip.gz** :



Nouveau répertoire	Nouveau fichier	Upload	Upload Java	Flash Upload	Install	Avancé	Transformer les entrées sélectionnées: Copier Déplacer Effacer Renommer Chmod				
									Recharger	Unzip	Taille
											Rechercher
Tous	Nom	Type	Taille	Propriétaire	Groupe	Permissions	Modifié le	Actions			
	Remonter ..										
	backup.php	Script PHP	1403	148834	users	rwcrwcrwoc	Apr 19 23:31	Voir	Éditer	Ouvrir	
	saue_base_format_gzip.gz	Archive GZ	5527	148834	users	rwcrwcrwoc	Apr 20 16:35	Voir	Éditer	Ouvrir	

Se rendre à nouveau dans votre gestion d'hébergement, cette fois-ci onglet **Base de données**, cliquer sur **Accéder à phpMyAdmin** sur la ligne correspondant à la base de donnée nécessitant une restauration :

Nom d'utilisateur	Nom de la base	Adresse du serveur	Taille	Type	Etat	Sauvegardes
			375 ko / 200 Mo	MYSQL v.5.7	OK	4

1

- Changer le mot de passe
- Créer une sauvegarde
- Restaurer une sauvegarde
- Importer un fichier
- Recalculer le quota
- Supprimer la base de données
- Accéder à phpMyAdmin

Renseigner les identifiants liés à la base de donnée :



## Bienvenue dans phpMyAdmin

Français - French

Sélectionner la base de données devant être restaurée et cliquer sur **importer** :

PHPMYADMIN		Serveur : hafsatcadmin.mysql.db	
<input type="button" value="Accueil"/> <input type="button" value="Ajouter"/> <input type="button" value="Ajouter"/> <input type="button" value="Ajouter"/> <input type="button" value="Ajouter"/> <input type="button" value="Ajouter"/>	<input type="button" value="Bases de données"/> <input type="button" value="SQL"/> <input type="button" value="État"/> <input type="button" value="Exporter"/> <input type="button" value="Importer"/>		
Récentes	Préférées		

(suite de la procédure page suivante)

Cliquer sur *Parcourir* et sélectionnez l'archive *sauve\_base\_format\_gzip.gz* récupérée chez l'hébergeur. Cliquer sur *Exécuter* en bas de page. Veiller également à laisser toutes les options cochées et représentées sur cette illustration :

## Importation dans le serveur courant

### FICHIER À IMPORTER :

Le fichier peut être compressé (gzip, bzip2, zip) ou non.  
Le nom du fichier compressé doit se terminer par .(format).(compression). Exemple : .sql.zip

Parcourir les fichiers :  Aucun fichier sélectionné. (Taille maximale : 128Mio)

Il est également possible de glisser-déposer un fichier sur n'importe quelle page.

Jeu de caractères du fichier :

### IMPORTATION PARTIELLE :

☒ Permettre l'interruption de l'importation si la limite de temps configurée dans PHP est sur le point d'être atteinte. (Ceci pourrait aider à importer des fichiers volumineux, au détriment du respect des transactions.)

Ignorer ce nombre de requêtes (pour SQL), à partir du début :

### AUTRES OPTIONS :

☒ Activer la vérification des clés étrangères

### FORMAT :

### OPTIONS SPÉCIFIQUES AU FORMAT :

Mode de compatibilité SQL :

☒ Ne pas utiliser AUTO\_INCREMENT pour la valeur zéro

La base de données vient normalement d'être restaurée.

Une autre solution consiste à se rendre directement dans le gestionnaire de base de données de l'hébergeur choisi pour cet atelier avec une offre d'hébergement, et de télécharger une sauvegarde automatique effectuée par l'hébergeur lui-même. Cela peut-être judicieux comme double facteur de sécurisation des données et non pas comme solution unique.

## Bilan

La plupart des objectifs ont été atteints. Comme pour le bilan de l'atelier de professionnalisation n°1, la différence entre le temps estimé pour chaque tâches et missions, et le temps réel, est conséquente. Il m'a fallu chercher de nombreuses informations afin de pallier mes lacunes. Mais c'est surtout sans l'appui et le soutien de ma tutrice Madame Martins Da Silva cela aurait été tout bonnement impossible de terminer ce devoir dans les temps. Son aide m'a été plus que précieuse, ainsi que sa disponibilité et bienveillance.

Concernant le codage du côté de l'application C#, cela n'a pas été forcément très problématique car cela se rapproche de Java que j'ai également eu le temps d'expérimenter en amont et nous sommes dans la même logique de développement orienté objet. Les différents outils utilisés ont été vus dans différents cours et travaux pratiques mais le fait de les utiliser dans cet atelier m'a permis de mieux les intégrer et prendre en main. Du côté de l'API REST, il m'aura fallu de nombreuses heures de travail afin de commencer à comprendre cette logique, voire débloquer ma logique. La partie tests a également été compliquée à aborder et reste confronté à un échec de test fonctionnel.

Expliquer les différentes étapes de chaque mission est également un exercice complexe, mais c'est aussi ce qui m'a permis de mieux comprendre certaines parties de code fonctionnelles et/ou de les réécrire. Évidemment, de nombreux points sont à revoir et je prévois de retourner sur ce projet dans quelques temps afin de l'optimiser et de mieux me rendre compte du travail effectué.

Je vais également profiter d'un peu de temps libre après mes examens pour pratiquer Git correctement, car je ne maîtrise pas assez cet outil indispensable.

A noter que ce travail devait être à l'origine exécuté en collaboration avec un de mes collègues de formation. Malheureusement je n'ai pu compter sur lui et n'ai pas trouvé d'autre alternative que de réaliser ce projet tout seul.