ATELIER PROFESSIONNEL

MEDIATEKDOCUMENTS

COMPTE -RENDU

Réalisé par :

SOUMIA B.AZZA

BTS SIO-SLAM

INTRODUCTION

C O N T E X T E

MediaTek86 est un réseau qui gère les médiathèques de la Vienne, et qui a pour rôle de fédérer les prêts de livres, DVD et CD et de développer la médiathèque numérique pour l'ensemble des médiathèques du département. Afin de gérer le catalogue et les commandes de documents, MediaTek86 a confié un projet de développement d'une application de gestion à la société InfoTechServices 86.

MISSION

Le but de cet atelier est de faire évoluer une application de bureau (C#) exploitant une API REST (PHP) pour l'accès à une base de données relationnelle MySQL et qui permet de gérer les

documents et les commandes des médiathèques de la chaîne MediaTek86. Un premier développeur s'est occupé de la construction de la base de données et du développement de certaines fonctionnalités de l'application. . Il s'agit d'une application de bureau, prévue d'être installée sur plusieurs postes dans la médiathèque et accédant à la même base de données. Le but est d'y ajouter de nouvelles fonctionnalités pour réaliser la gestion du catalogue des médiathèques et des commandes de documents

LANGAGE ET OUTILS

- L. programmation: C# , PHP, SQL.
- Serveur: WampServer: APACHE, PHP, MYSQL.
- Tests: POSTMAN, SPECFLOW.
- Qualité logiciel: SONARQUBE, SONARLINT, JENKINS
- Versioning: GITHUB.
- IDE: NETBEANS, VISUAL STUDIO

SOMMAIRE

MISSION 1. GÉRER LES DOCUMENTS

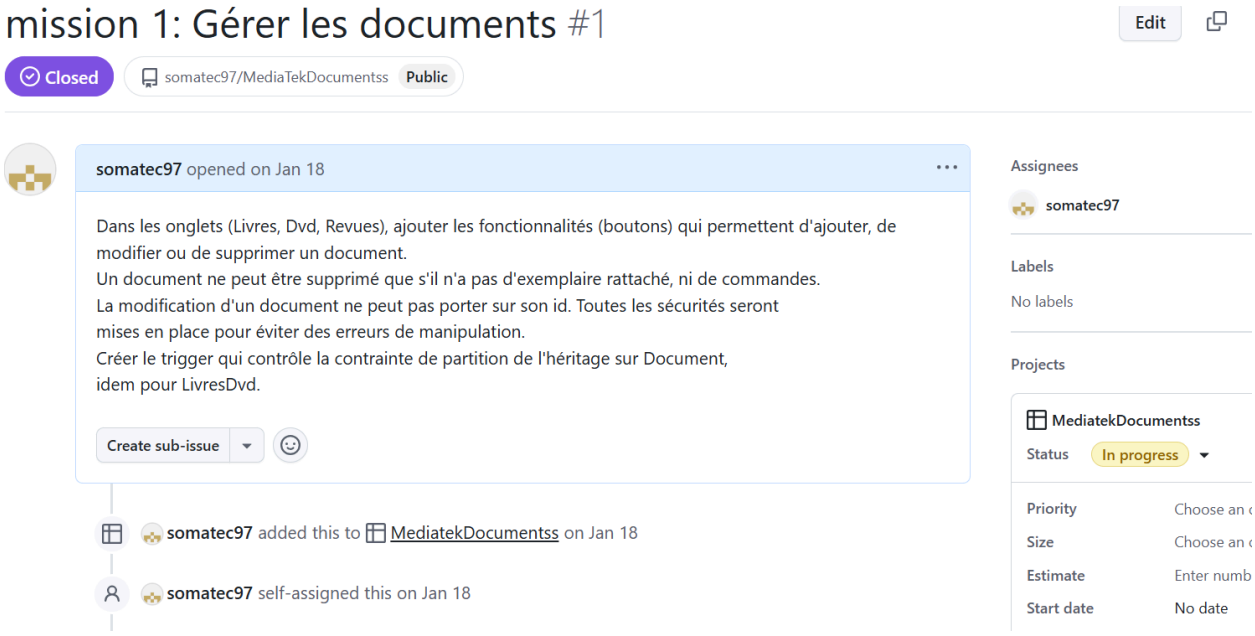
Dans les onglets actuels (Livres, Dvd, Revues), ajouter les fonctionnalités (boutons) qui permettent d'ajouter, de modifier ou de supprimer un document.

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 estimé 8h - Temps mis environ 15h.



AJOUTER LES FONCTIONNALITÉS "AJOUTER"

Lors de ce projet, une api_rest Php a été mise à disposition, afin d'exploiter la base de données dans l'application C#.

Grâce à cette API, les fonctions permettant d'ajouter, de modifier ou de supprimer des données dans la base, sont déjà créées dans le code de l'API. Il ne reste donc plus qu'à exploiter ces fonctions déjà construites.

FONCTIONNALITÉ "AJOUTER" - ONGLET LIVRES

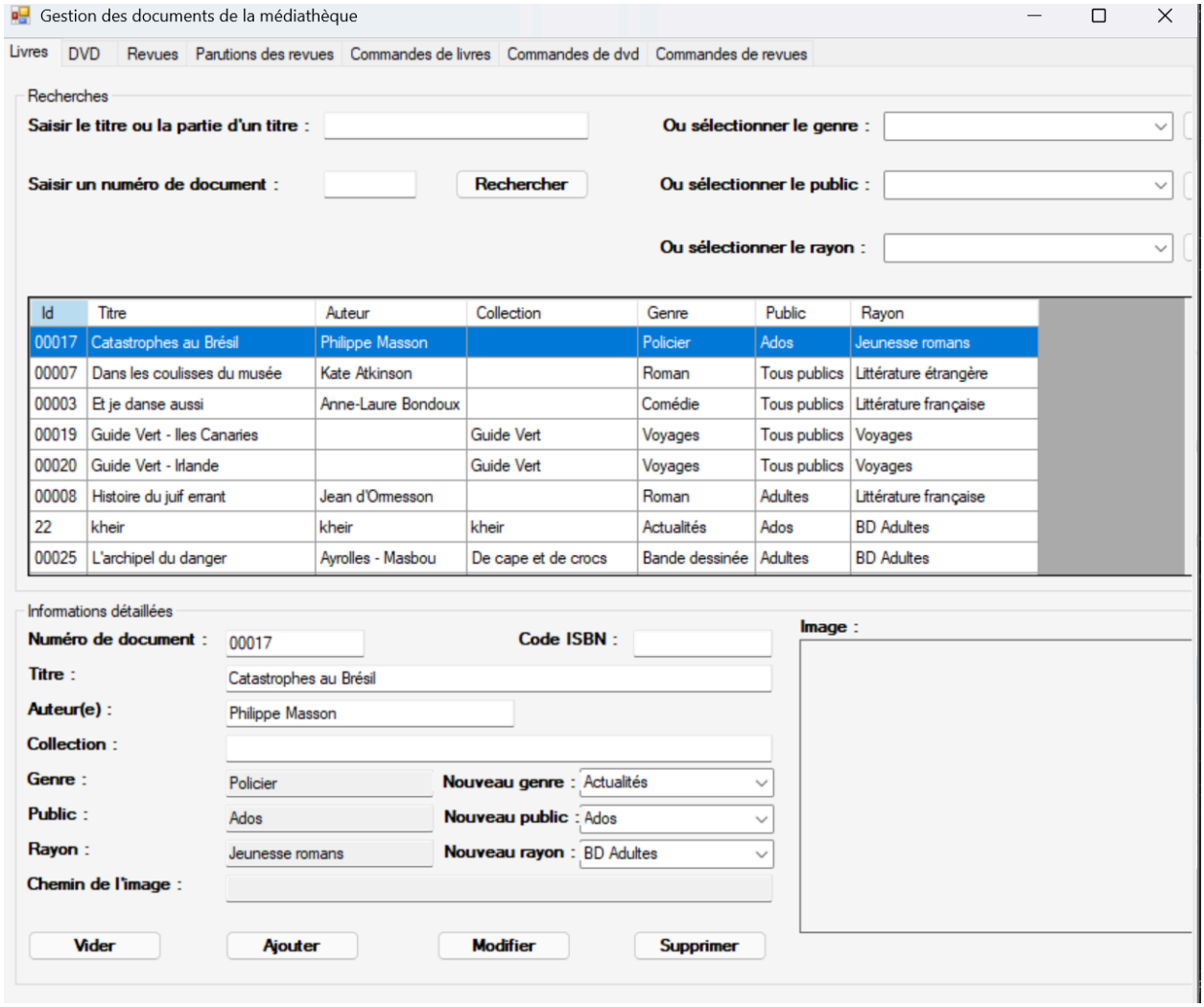
Je commence par créer les boutons "Ajouter", "Modifier" et "Supprimer", dans la vue, sur les onglets "livres", "dvd" et "revues".

Je me rends dans la vue "FrmMediatek.cs", je sélectionne "button" dans la boîte à outils de visual studio.

J'insère trois boutons "Ajouter" "Modifier" "Supprimer" , sous les informations détaillées du document de chacun des onglets.

J'en profite également pour modifier le "ReadOnly", en le passant à "false", sur les champs pour lesquels il faudra que l'utilisateur puisse insérer des valeurs.

J'ajoute 3 comboBox pour que l'utilisateur puisse sélectionner un nouveau genre, public et rayon, pour les affecter à un document qu'il souhaite ajouter ou modifier.



Afin de créer la fonctionnalité "ajouter" dans l'onglet "livres", je dois utiliser la fonction de l'api rest php, appelée "Insert One", cette fonction peut être exploitée dans l'application C#, en appelant la méthode "POST".

Cette méthode POST va permettre de récupérer les données saisies par l'utilisateur dans l'interface de l'application sur l'onglet "livres", puis de les insérer dans la base de données.

Pour cela, il faut que les données saisies soient envoyées à l'api au format JSON.

Je commence donc par réaliser la fonction d'ajout des informations d'un document dans la classe "Access.cs" de l'application C#.

J'ai nommé cette fonction "CreerDocument(string Id, string Titre, string Image, string IdRayon, string IdPublic, string IdGenre)", dans laquelle je lui affecte les paramètres correspondants aux champs de la table "document" de la base de données. J'effectue ensuite la sérialisation des données de l'api au format json, en les affectant dans une chaîne json "jsonCreerDocument".

Pour avoir une visibilité des données ajoutées, j'ajoute un "console.writeline" qui va afficher le résultat de la chaîne. Je récupère ensuite les données en appelant la méthode "POST" de l'api, qui va ajouter le nouveau document dans la table "document".

Je précise la route vers le endpoint "document/".

Méthode - CreerDocument

```
public bool CreerDocument(string Id, string Titre, string Image, string IdRayon, string IdPublic, string IdGenre)

{

String jsonCreerDocument = "{ \"id\" : \""+Id+ "\", \"titre\" : \""+Titre+ "\",

\"image\" : \""+Image+ "\", \"idRayon\" : \""+IdRayon+ "\",

\"idPublic\" : \""+IdPublic+ "\", \"idGenre\" : \""+IdGenre+ "\"}";
Console.WriteLine("jsonCreerDocument" + jsonCreerDocument); try { List liste = TraitementRecup(POST, "document/" + jsonCreerDocument);

return (liste != null);

}

catch (Exception ex) {

Console.WriteLine(ex.Message);
```

```
}
 return false;
}
```

Je dois ensuite appeler cette fonction dans le contrôleur, pour cela, je crée une méthode "CreerDocument(string Id, string Titre, string Image, string IdRayon, string IdPublic, string IdGenre)", dans la classe "FrmMediatekController.cs".

```
public bool CreerDocument(string Id, string Titre, string Image, string IdRayon, string IdPublic, string IdGenre) { return access.CreerDocument(Id, Titre, Image, IdRayon, IdPublic, IdGenre); }
```

La méthode "CreerDocument", me permet d'ajouter un document à la table "document", toutefois, pour que le bouton "Ajouter", ajoute également un nouveau "livre" à la table "livre", il faut que je récupère les valeurs du livre à ajouter, qui seront saisies par l'utilisateur dans les champs des informations détaillées. Je crée donc une nouvelle méthode "CreerLivre(string Id, string Isbn, string Auteur, string Collection)", dans la classe "Access.cs", qui va prendre en paramètres les différents champs de la table "livre". Ces données sont ensuite sérialisées dans une chaîne "json", appelée "jsonCreerLivre", puis envoyée avec la méthode "POST", sur le endpoint "livre/" .

Méthode - CreerLivre

```
public bool CreerLivre(string Id, string Isbn, string Auteur, string Collection)
{
 String jsonCreerLivre = "{ \"id\" : \"" + Id + "\", \"isbn\" : \"" + Isbn + "\", \"auteur\" : \"" + Auteur + "\", \"collection\" : \"" + Collection + "\"}";
Console.WriteLine("jsonCreerLivre" + jsonCreerLivre);
 try {
```

// récupération soit d'une liste vide (requête ok) soit de null (erreur) List liste = TraitementRecup(POST, "livre/" + jsonCreerLivre);

return (liste != null);

} catch (Exception ex) {

Console.WriteLine(ex.Message);

} return false;

Une fois réalisée, je dois appeler cette fonction dans le contrôleur, j'insère donc la méthode "CreerLivre(string Id, string Isbn, string Auteur, string Collection)", à la classe "FrmMediatekController.cs". public bool CreerLivre(string Id, string Isbn, string Auteur, string Collection) { return access.CreerLivre(Id, Isbn, Auteur, Collection); }

Je peux maintenant aller dans le code de la vue "FrmMediatek.cs", pour effectuer la procédure d'ajout d'un document et d'un livre, lors du clic sur le bouton "Ajouter".

Pour cela, je créé une procédure événementielle "btnReceptionLivreValider_Click", lorsque le numéro du document est inscrit dans le champ "txbLivresNumero", l'application va récupérer les données saisies par l'utilisateur dans chacun des champs des "informations détaillées", puis les affecter à un nouveau document, et à un nouveau livre, qui porteront tout les deux le même Id.

C'est grâce à l'appel des méthodes "CreerDocument" et "CreerLivre", que le nouveau document est ajouté à la base de données.

La fonction permet ensuite d'afficher la base de données mise à jour, avec les livres et les informations du document associé aux livres.

Pour sécuriser les saisies de l'utilisateur, j'ai choisi d'insérer des comboBox pour permettre d'ajouter le genre, le public et le rayon du document. Afin de remplir les comboBox avec les listes des catégories correspondantes, j'ai construis des méthodes permettant le remplissage de chacune d'elle "RemplirCbxAjoutModifGenreLivre()", et "RemplirCbxAjoutModifPublicLivre()", "RemplirCbxAjoutModifRayonLivre()".

Ces méthodes se présentent toutes les trois de cette manière, selon la méthode, l'appel des méthodes "GelAll" et les noms des cbx sont modifiées :

Méthode - RemplirCbxAjoutModifGenreLivre()

```
private string RemplirCbxAjoutModifGenreLivre()
{
List lesGenresLivres = controller.GetAllGenres();
foreach (Categorie genre in lesGenresLivres)
{
cbxAjoutModifGenreLivre.Items.Add(genre.Libelle);
}
if (cbxAjoutModifGenreLivre.Items.Count > 0) {
cbxAjoutModifGenreLivre.SelectedIndex = 0;
} return cbxAjoutModifGenreLivre.SelectedItem?.ToString();
}
```

Dans le but de récupérer l'id du genre, du public et du rayon sélectionnés, je crée 3 méthodes supplémentaires "GetIdGenreDocument(string genre)", "GetIdPublicDocument(string lePublic)" et "GetIdRayonDocument(string rayon)". Elles vont récupérer dans la liste des catégories, l'id de la catégorie correspondant au libelle sélectionné dans chaque comboBox, grâce aux appels des méthodes "GetAll" et à la condition "if" que j'ai ajouté. Elles se présentent de la même manière, il n'y a que les GetAll qui changent et le nom des variables :

Méthode - GetIdGenreDocument

```
private string GetIdGenreDocument(string genre)
```

{

List lesGenresDocument = controller.GetAllGenres(); foreach (Categorie c in lesGenresDocument) { if (c.Libelle == genre) { return c.Id; } }

return null;

}

J'ajoute également d'autres sécurités, dans la procédure événementielle du clic sur le bouton Ajouter" de l'onglet "livres". Les sécurités misent en place obligent l'utilisateur à saisir tous les champs obligatoires pour ajouter un document de type "livre", et l'utilisateur ne peut pas ajouter un nouveau livre si l'id est déjà présent dans la base de données grâce à la condition "if(idLivreNonExistant)".

C'est également dans cette procédure que j'affecte les méthodes "GetId" dans les propriétés "idGenre", "idPublic" et "idRayon", sinon la requête ne récupère pas l'id correspondant au libelle et par conséquent, ne fonctionne pas. Voici un extrait de la procédure événementielle "btnReceptionLivreAjouter_Click" :

Procédure évènementielle - btnReceptionLivreAjouter_Click

Document document = new Document(id, titre, image, idGenre, genre, idPublic, Public, idRayon, rayon);

Livre livre = new Livre(id, titre, image, isbn, auteur, collection, idGenre, genre, idPublic, Public, idRayon, rayon);

var idLivreExistant = controller.GetAllDocuments(id); var idLivreNonExistant = !idLivreExistant.Any();

if (idLivreNonExistant) {

```
 if (controller.CreerDocument(document.Id, document.Titre, document.Image,
document.IdRayon, document.IdPublic, document.IdGenre) && controller.CreerLivre(livre.Id,
livre.Isbn, livre.Auteur, livre.Collection)) {

 lesLivres = controller.GetAllLivres(); RemplirComboCategorie(controller.GetAllGenres(),
bdgGenres, cbxLivresGenres);

RemplirComboCategorie(controller.GetAllPublics(), bdgPublics, cbxLivresPublics);
RemplirComboCategorie(controller.GetAllRayons(), bdgRayons, cbxLivresRayons);
RemplirLivresListeComplete();

 MessageBox.Show("Le livre " + titre + " a correctement été ajouté.");

 }

 } else {

 MessageBox.Show("Le numéro de document existe déjà, veuillez saisir un autre numéro.",
"Erreur");

 }
```

Lors de la création de la fonctionnalité d'ajout, sur l'onglet "livres", je me suis rendue compte qu'il n'était pas possible d'ajouter un nouveau livre, si l'id du nouveau livre n'existait pas déjà dans la table "livres_dvd" et dans la table "document". Afin de pallier ce problème, j'ai dû réaliser un trigger, qui permet l'ajout de l'id du livre, dans la table "livres_dvd", lors de la création d'un nouveau livre.

Trigger - Ajout livre

```
DELIMITER $$

 CREATE TRIGGER `insert_livre_to_livres_dvd` BEFORE INSERT ON `livre`

 FOR EACH ROW
```

```
BEGIN

DECLARE cnt INT;

SELECT COUNT(*) INTO cnt FROM livres_dvd WHERE id = NEW.id;

IF cnt = 0 THEN INSERT INTO livres_dvd (id) VALUES (NEW.id);

END IF;

END

$$ DELIMITER ;
```

Sachant que l'id du nouveau livre, doit également exister dans la table "document" avant d'être ajouté, il a également fallu que je réalise un second trigger, permettant d'ajouter l'id du livre dans la table "document", avant de l'insérer dans la table "livres_dvd", les id de ces tables étant liés entre eux.

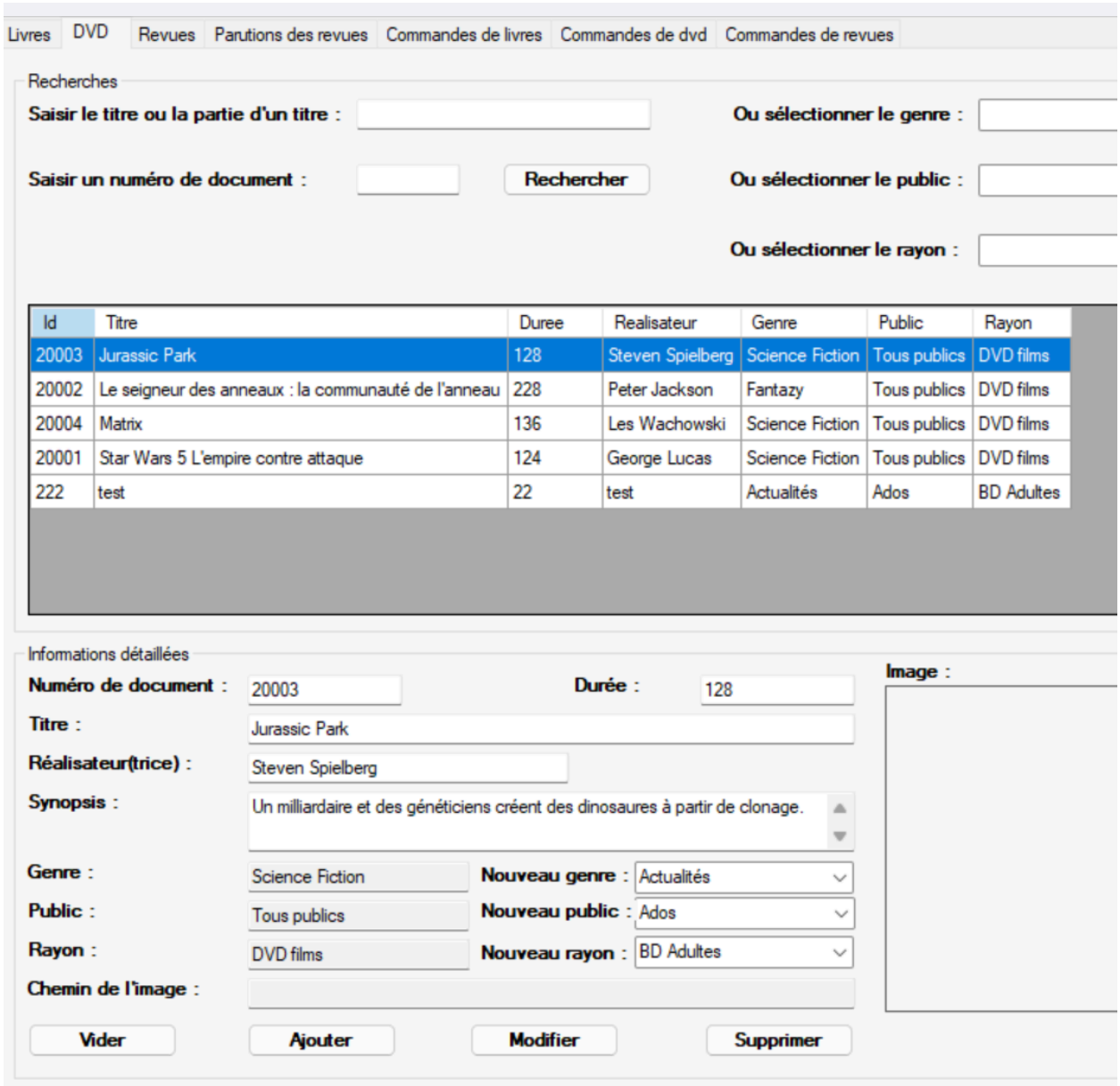
Trigger - Ajout livre_dvd

```
DELIMITER $$

CREATE TRIGGER `insert_livres_dvd_to_document` BEFORE INSERT ON `livres_dvd` FOR EACH ROW

BEGIN

DECLARE cnt INT;

SELECT COUNT(*) INTO cnt FROM document WHERE id = NEW.id;

IF cnt = 0 THEN INSERT INTO document (id) VALUES (NEW.id);

END IF;

END
```

$$ DELIMITER ;

Je teste ensuite le fonctionnement de mon bouton "ajouter", en insérant des valeurs dans les zones de saisie de l'onglet, puis je vérifie que la ligne a bien été ajoutée.

2 . FONCTIONNALITÉ "AJOUTER" - ONGLET DVD



La méthode "CreerDocument", réalisée précédemment dans la classe "Access", me permet d'ajouter un document à la table "document", toutefois, pour que le bouton "Ajouter", ajoute également un nouveau "dvd" à la table "dvd", il faut que je récupère les valeurs du dvd à ajouter, qui seront saisies par l'utilisateur dans les champs des informations détaillées.

Je crée donc une nouvelle méthode "CreerDvd(string Id, string Synopsis, string Realisateur, int Duree)", dans la classe "Access", qui va prendre en paramètres les différents champs de la table "dvd". Ces données sont ensuite sérialisées dans une chaîne "json", appelée "jsonCreerDvd", puis envoyée avec la méthode "POST", sur le endpoint "dvd/" .

Méthode - CreerDvd

```
public bool CreerDvd(string Id, string Synopsis, string Realisateur, int Duree) {

String jsonCreerDvd = "{ \"id\" : \"" + Id + "\", \"synopsis\" : \"" + Synopsis + "\", \"realisateur\" : \"" + Realisateur + "\", \"duree\" : \"" + Duree + "\"}";

Console.WriteLine("jsonCreerDvd" + jsonCreerDvd);

try {

List liste = TraitementRecup(POST, "dvd/" + jsonCreerDvd); return (liste != null);

}

 catch (Exception ex) { Console.WriteLine(ex.Message); }

return false;

 }
```

Une fois réalisée, je dois appeler cette fonction dans le contrôleur, j'insère donc la méthode "CreerDvd, à la classe "FrmMediatekController.cs".

```
public bool CreerDvd(string Id, string Synopsis, string Realisateur, int Duree) { return access.CreerDvd(Id, Synopsis, Realisateur, Duree); }
```

Je peux maintenant aller dans le code de la vue "FrmMediatek.cs", pour effectuer la procédure d'ajout d'un document et d'un dvd, lors du clic sur le bouton "Ajouter".

Pour cela, je créé une procédure évènementielle "btnReceptionDvdValider_Click", lorsque le numéro du document est inscrit dans le champ "txbDvdNumero", l'application va récupérer les données saisies par l'utilisateur dans chacun des champs, puis les insérer dans un nouveau document, et dans un nouveau dvd, qui porteront tout les deux le même Id.

C'est grâce à l'appel des méthodes "CreerDocument" et "CreerDvd", que le nouveau document est ajouté à la base de données.

La fonction permet ensuite d'afficher la base de données mise à jour, avec les dvd et les informations du document associé aux dvd. Pour que l'ajout d'un dvd fonctionne, je dois aller dans l'api rest_mediatekdocuments, et ajouter à la fonction "SelectOne" : case "dvd" : return $this->selectAllDvd($id);

Afin de remplir les comboBox de l'onglet dvd, avec les listes des catégories correspondantes, j'ai construis des méthodes permettant le remplissage de chacune d'elle "RemplirCbxAjoutModifGenreDvd()","RemplirCbxAjoutModifPublicDvd()", et "RemplirCbxAjoutModifRayonDvd()".

Ces méthodes se présentent toutes les trois de cette manière, selon la méthode, l'appel des méthodes "GelAll" et les noms des cbx sont modifiées :

Méthode - RemplirCbxAjoutModifGenreDvd()

```
 private string RemplirCbxAjoutModifGenreDvd() {

List lesGenresDvd = controller.GetAllGenres();

foreach (Categorie genre in lesGenresDvd) {

 cbxAjoutModifGenreDvd.Items.Add(genre.Libelle);

} if (cbxAjoutModifGenreDvd.Items.Count > 0) {
```

```
 cbxAjoutModifGenreDvd.SelectedIndex = 0; }

 return cbxAjoutModifGenreDvd.SelectedItem?.ToString();

}
```

J'ajoute également d'autres sécurités, dans la procédure évènementielle du clic sur le bouton "Ajouter" de l'onglet "dvd".

Les sécurités misent en place obligent l'utilisateur à saisir tous les champs obligatoires pour ajouter un document de type "dvd", et l'utilisateur ne peut pas ajouter un nouveau dvd si l'id est déjà présent dans la base de données grâce à la condition "if(idDvdNonExistant)".

C'est également dans cette procédure que j'affecte les méthodes "GetId" dans les propriétés "idGenre", "idPublic" et "idRayon", sinon la requête ne récupère pas l'id correspondant au libelle et par conséquent, ne fonctionne pas.

Voici un extrait de la procédure évènementielle "btnReceptionDvdValider_Click" :

Procédure évènementielle - btnReceptionDvdValider_Click

```
 string idGenre = GetIdGenreDocument(cbxAjoutModifGenreDvd.Text);

 string genre = txbDvdGenre.Text; string idPublic =
GetIdPublicDocument(cbxAjoutModifPublicDvd.Text);

 string lePublic = txbDvdPublic.Text; string idRayon =
GetIdRayonDocument(cbxAjoutModifRayonDvd.Text);

 string rayon = txbDvdRayon.Text;

Document document = new Document(id, titre, image, idGenre, genre, idPublic,

lePublic, idRayon, rayon); Dvd dvd = new Dvd(id, titre, image, duree, realisateur,

 synopsis, idGenre, genre, idPublic, lePublic, idRayon, rayon); var idDvdExistant =
controller.GetAllDocuments(id);
```

```
var idDvdNonExistant = !idDvdExistant.Any();

if (idDvdNonExistant) {

if (controller.CreerDocument(document.Id, document.Titre, document.Image,
document.IdRayon, document.IdPublic, document.IdGenre) && controller.CreerDvd(dvd.Id,
dvd.Synopsis, dvd.Realisateur, dvd.Duree)) {

lesDvd = controller.GetAllDvd();

RemplirComboCategorie(controller.GetAllGenres(), bdgGenres, cbxDvdGenres);
RemplirComboCategorie(controller.GetAllPublics(), bdgPublics, cbxDvdPublics);
RemplirComboCategorie(controller.GetAllRayons(), bdgRayons, cbxDvdRayons);
RemplirDvdListeComplete();

 MessageBox.Show("Le dvd " + titre + " a correctement été ajouté."); }

} else { MessageBox.Show("Le numéro de document existe déjà.", "Erreur");

} }
```

Lors de la création de la fonctionnalité d'ajout, sur l'onglet "dvd", je me suis rendue compte qu'il n'était pas possible d'ajouter un nouveau dvd, si l'id du nouveau dvd n'existait pas déjà dans la table "livres_dvd". Afin de pallier à ce problème, j'ai dû réaliser un trigger, qui permet l'ajout de l'id du dvd, dans la table "livres_dvd", lors de la création d'un nouveau dvd.

Trigger - Ajout dvd

```
DELIMITER $$

CREATE TRIGGER `insert_dvd_to_livres_dvd` BEFORE INSERT ON `dvd`

FOR EACH ROW

BEGIN
```

IF NOT EXISTS (SELECT id FROM livres_dvd WHERE id = NEW.id) THEN INSERT INTO livres_dvd (id) VALUES (NEW.id);

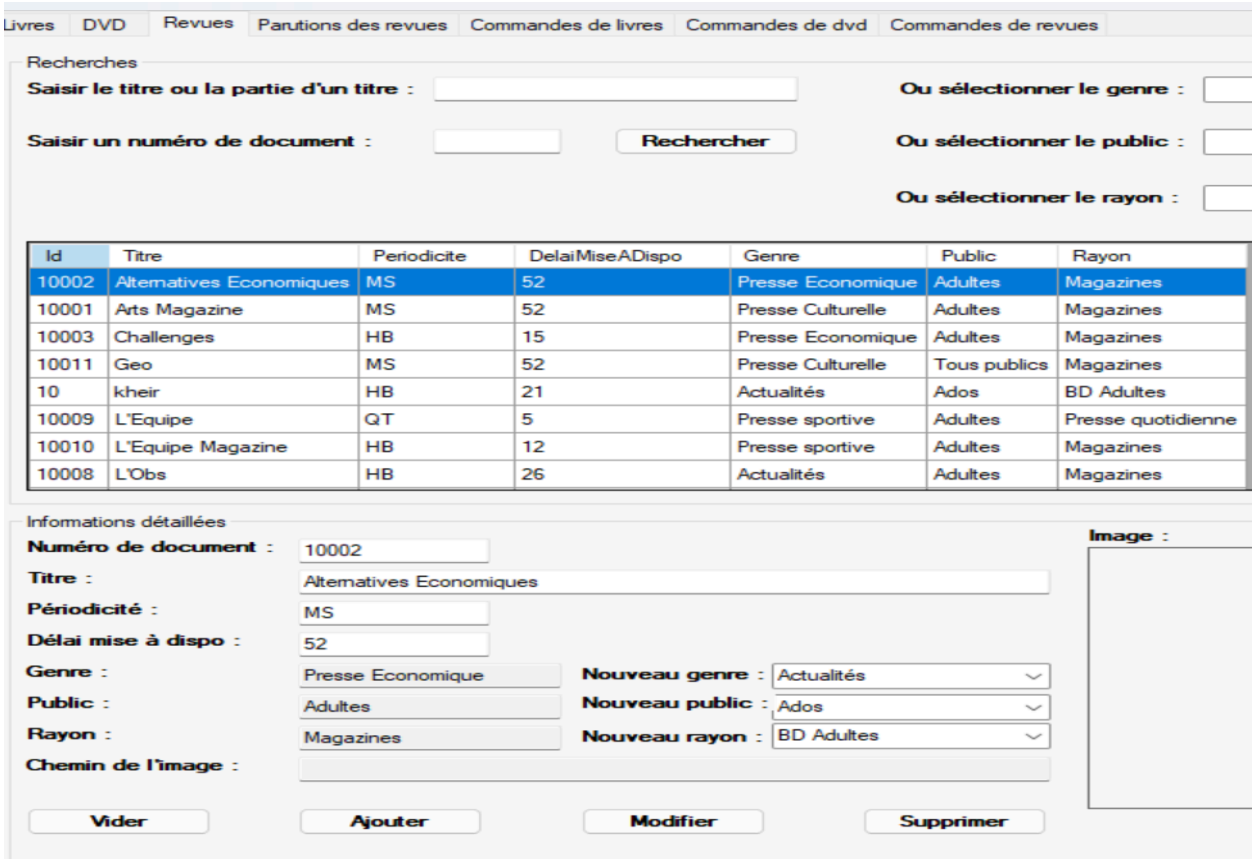
END IF;

END

$$ DELIMITER ;

Je teste ensuite la fonctionnalité "ajouter un dvd" dans l'onglet "dvd", en saisissant des informations sur un nouveau dvd à ajouter, puis je clique sur le bouton "Ajouter".

3 . FONCTIONNALITÉ "AJOUTER" - ONGLET REVUES



Pour que le bouton "Ajouter", ajoute également une nouvelle "revue" à la table "revue", il faut que je récupère les valeurs de la revue à ajouter, qui seront saisies par l'utilisateur dans les champs des informations détaillées.

Je crée donc une nouvelle fonction "CreerRevue(string Id, string Periodicite, int DelaiMiseADispo)", dans la classe "Access.cs", qui va prendre en paramètres les différents champs de la table "revue".

Ces données sont ensuite sérialisées dans une chaîne "json", appelée "jsonCreerRevue", puis envoyée avec la méthode "POST", sur le endpoint "revue/".

Méthode - CreerRevue

```
public bool CreerRevue(string Id, string Periodicite, int DelaiMiseADispo) {

 String jsonCreerRevue = "{ \"id\" : \"" + Id + "\", \"periodicite\" : \"" + Periodicite + "\", \"delaiMiseADispo\" : \"" + DelaiMiseADispo + "\"}";

 Console.WriteLine("jsonCreerRevue" + jsonCreerRevue);

 try {

List liste = TraitementRecup(POST, "revue/" + jsonCreerRevue); return (liste != null);

 } catch (Exception ex) {

 Console.WriteLine(ex.Message);

} return false;

 }
```

Une fois réalisée, je dois appeler cette fonction dans le contrôleur, j'insère donc la méthode "CreerRevue, à la classe "FrmMediatekController.cs". public bool CreerRevue(string Id, string Periodicite, int DelaiMiseADispo) { return access.CreerRevue(Id, Periodicite, DelaiMiseADispo); }

Je peux maintenant aller dans le code de la vue "FrmMediatek.cs", pour effectuer la procédure d'ajout d'un document et d'une revue lors du clic sur le bouton "Ajouter".

Pour cela, je créé une procédure évènementielle "btnReceptionRevueValider_Click", lorsque le numéro du document est inscrit dans le champ "txbRevueNumero", l'application va récupérer les données saisies par l'utilisateur dans chacun des champs, puis les affecter à un nouveau document, et à une nouvelle revue, qui porteront tout les deux le même Id.

C'est grâce à l'appel des méthodes "CreerDocument" et "CreerRevue", que le nouveau document est ajouté à la base de données. La fonction permet ensuite d'afficher la base de données mise à jour, avec les revues et les informations du document associé aux revues.

Pour que l'ajout d'une revue fonctionne, je dois aller dans l'api rest Php, et ajouter à la fonction "SelectOne" : case "revue" : return $this->selectAllRevues($id); précédents, Je remplis les comboBox du genre, du public et du rayon, de la même manière que dans les onglets avec les méthodes "RemplirCbxAjoutModifGenreRevue()" "RemplirCbxAjoutModifPublicRevue()", et "RemplirCbxAjoutModifRayonRevue()",

Je crée aussi les mêmes sécurités. Voici un extrait de la procédure événementielle "btnReceptionRevueValider_Click" :

Procédure évènementielle - btnReceptionRevueValider_Click

```
string idRayon = GetIdRayonDocument(cbxAjoutModifRayonRevue.Text); string rayon =

txbRevuesRayon.Text;

 string periodicite = txbRevuesPeriodicite.Text;

 int delaiMiseADispo = int.Parse(txbRevuesDateMiseADispo.Text);

Document document = new Document(id, titre, image, idGenre, genre, idPublic,

 lePublic, idRayon, rayon);
```

```
Revue revue = new Revue(id, titre, image, idGenre, genre, idPublic, lePublic, idRayon, rayon, periodicite, delaiMiseADispo);

var idRevueExistante = controller.GetAllDocuments(id);

var idRevueNonExistante = !idRevueExistante.Any();
```

Lors de la création de la fonctionnalité d'ajout, sur l'onglet "revue", je me suis rendue compte qu'il n'était pas possible d'ajouter une nouvelle revue, si l'id de la nouvelle revue n'existait pas déjà dans la table "document".

Afin de pallier à ce problème, j'ai dû réaliser un trigger, qui permet l'ajout de l'id de la revue, dans la table "document", lors de la création d'une nouvelle revue.

Trigger - Ajout revue

```
DELIMITER $$

CREATE TRIGGER `insert_revue_to_document` BEFORE INSERT ON `revue`

FOR EACH ROW

BEGIN

DECLARE cnt INT; SELECT COUNT(*) INTO cnt FROM document WHERE id = NEW.id; IF cnt = 0 THEN INSERT INTO document (id) VALUES (NEW.id);

END IF;

END

$$ DELIMITER ;
```

Je teste ensuite la fonctionnalité "Ajouter une revue", dans l'onglet "revue", en saisissant les informations d'une nouvelle revue à ajouter, puis en cliquant sur le bouton "Ajouter".

AJOUTER LES FONCTIONNALITÉS "MODIFIER"

FONCTIONNALITÉ "MODIFIER" - ONGLET LIVRES

Afin de créer le bouton "modifier" dans l'onglet "livres", je dois utiliser la fonction de l'api rest mediatekdocuments, appelée "update One", cette fonction peut être exploitée dans l'application C#, en appelant la méthode "PUT".

Cette méthode PUT va permettre de récupérer les données saisies par l'utilisateur dans l'interface de l'application sur les différents onglets, puis d'enregistrer les modifications de la ligne sélectionnée, dans la base de données.

Pour cela, il faut que les données saisies soient envoyées à l'api au format JSON.

Je commence donc par réaliser la méthode de modification des informations d'un document dans la classe "Access.cs" de l'application C#.

J'ai nommé cette méthode "ModifierDocument(string Id, string Titre, string Image, string IdGenre, string IdPublic, string IdRayon)", dans laquelle j'ai affecté les paramètres correspondants aux champs de la table "document" de la base de données.

J'effectue ensuite la sérialisation des données de l'api au format json, en les plaçant dans une chaîne json "jsonModifierDocument".

Pour avoir une visibilité des données modifiées, j'ajoute un "console.writeline" qui va afficher le résultat de la chaîne. Je récupère ensuite les données en appelant la méthode "PUT" de l'api, qui va enregistrer les modifications du document dans la table "document".

Pour le endpoint, je précise la table "document/" et l'id du document concerné.

Méthode - ModifierDocument

```
public bool ModifierDocument(string Id, string Titre, string Image, string IdRayon, string IdPublic, string IdGenre) {

String jsonModifierDocument = "{ \"id\" : \""+Id+ "\", \"titre\" : \""+Titre+ "\", \"image\" : \""+Image+ "\", \"idRayon\" : \""+IdRayon+ "\", \"idPublic\" : \""+IdPublic+ "\", \"idGenre\" : \""+IdGenre+ "\"}";
```

```
Console.WriteLine("jsonModifierDocument" + jsonModifierDocument);

 try {

 List liste = TraitementRecup(PUT, "document/" + Id + "/" + jsonModifierDocument);

return (liste != null);

 } catch (Exception ex) {

Console.WriteLine(ex.Message);

 }

 return false;

 }
```

Je dois ensuite appeler cette méthode dans le contrôleur, pour cela, je crée une méthode "ModifierDocument", dans la classe "FrmMediatekController.cs"

```
. public bool ModifierDocument(string Id, string Titre, string Image, string IdRayon, string IdPublic, string IdGenre) { return access.ModifierDocument(Id, Titre, Image, IdRayon, IdPublic, IdGenre); }
```

Pour que le bouton "Modifier", modifie également les informations d'un livre, dans la table "livre", il faut que je récupère les valeurs du livre à modifier, qui seront saisies par l'utilisateur dans les champs des informations détaillées.

Je crée donc une nouvelle fonction "ModifierLivre(string Id, string Isbn, string Auteur, string Collection)", dans la classe "Access.cs", qui va prendre en paramètres les différents champs de la table "livre".

Ces données sont ensuite sérialisées dans une chaîne "json", appelée "jsonModifierLivre", puis envoyée avec la méthode "PUT", sur le endpoint livre/id.

Méthode - ModifierLivre

```
 public bool ModifierLivre(string Id, string Isbn, string Auteur, string Collection) {
String jsonModifierLivre = "{ \"id\" : \"" + Id + "\", \"isbn\" : \"" + Isbn + "\", \"auteur\" :
 \"" + Auteur + "\", \"collection\" : \"" + Collection + "\"}";
 Console.WriteLine("jsonModifierLivre" + jsonModifierLivre);
 try {
 // récupération soit d'une liste vide (requête ok) soit de null (erreur) List liste =
TraitementRecup(POST, "livre/" + Id + "/" + jsonModifierLivre); return (liste != null);
 }
catch (Exception ex) {
 Console.WriteLine(ex.Message);
}
 return false;
```

Une fois réalisée, je dois appeler cette méthode dans le contrôleur, j'insère donc la méthode "ModifierLivre", à la classe "FrmMediatekController.cs".

```
public bool ModifierLivre(string Id, string Isbn, string Auteur, string Collection)
 {
 return access.ModifierLivre(Id, Isbn, Auteur, Collection);
}
```

Je peux maintenant aller dans le code de la vue "FrmMediatek.cs", pour effectuer la procédure de modification d'un document et d'un livre, lors du clic sur le bouton "Modifier".

Pour cela, je créé une procédure évènementielle "btnReceptionLivreModifier_Click", lorsqu'une ligne est sélectionnée dans la datagrid "dgvLivresListe.SelectedRows.Count > 0", l'application va récupérer les données saisies par l'utilisateur dans chacun des champs, puis les affecter au document sélectionné, et au livre associé à ce document, qui portent tout les deux le même Id.

C'est grâce à l'appel des méthodes "ModifierDocument" et "ModifierLivre", que le document est modifié dans la base de données.

La fonction permet ensuite d'afficher la base de données mise à jour, avec les livres et les informations du document associé.

Voici un extrait de la procédure évènementielle btnReceptionLivreModifier_Click :

Procédure évènementielle - btnReceptionLivreModifier_Click

```
if (controller.ModifierLivre(id, isbn, auteur, collection)
&& controller.ModifierDocument(id, titre, image, idGenre, idPublic, idRayon)) {
lesLivres = controller.GetAllLivres();
RemplirComboCategorie(controller.GetAllGenres(), bdgGenres, cbxLivresGenres);
RemplirComboCategorie(controller.GetAllPublics(), bdgPublics, cbxLivresPublics);
RemplirComboCategorie(controller.GetAllRayons(), bdgRayons, cbxLivresRayons);
RemplirLivresListeComplete(); MessageBox.Show("Le livre " + titre + " a correctement été modifié.");
}
else
{
```

```
 MessageBox.Show("Erreur lors de la modification du livre", "Erreur");

 }
```

FONCTIONNALITÉ "MODIFIER" - ONGLET DVD

Pour que le bouton "Modifier", modifie également un "dvd" sélectionné dans la table "dvd", il faut que je récupère les valeurs du dvd à modifier, qui seront saisies par l'utilisateur dans les champs des informations détaillées.

Je crée donc une nouvelle méthode "ModifierDvd(string Id, string Synopsis, string Realisateur, int Duree)", dans la classe "Access.cs", qui va prendre en paramètres les différents champs de la table "dvd".

Ces données sont ensuite sérialisées dans une chaîne "json", appelée "jsonModifierDvd", puis envoyées avec la méthode "PUT", sur le endpoint dvd/id.

Méthode - ModifierDvd

```
public bool ModifierDvd(string Id, string Synopsis, string Realisateur, int Duree) {

 String jsonModifierDvd = "{ \"id\" : \"" + Id + "\", \"synopsis\" : \"" + Synopsis + "\",

 \"realisateur\" : \"" + Realisateur + "\", \"duree\" : \"" + Duree + "\"}";

Console.WriteLine("jsonModifierDvd" + jsonModifierDvd);

try {

 List liste = TraitementRecup(PUT, "dvd/" + Id + "/" + jsonModifierDvd);

return (liste != null);

} catch (Exception ex) {

Console.WriteLine(ex.Message);

 } return false; }
```

Une fois réalisée, je dois appeler cette fonction dans le contrôleur, j'insère donc la méthode "ModifierDvd, à la classe "FrmMediatekController.cs".

```
public bool ModifierDvd(string Id, string Synopsis, string Realisateur, int Duree) {
return access.ModifierDvd(Id, Synopsis, Realisateur, Duree);
}
```

Je peux maintenant aller dans le code de la vue "FrmMediatek.cs", pour effectuer la procédure de modification d'un document et d'un dvd, lors du clic sur le bouton "Modifier".

Pour cela, je créé une procédure évènementielle "btnReceptionDvdModifier_Click", lorsqu'une ligne est sélectionnée dans la datagrid "dgvDvdListe.SelectedRows.Count > 0", l'application va récupérer les données saisies par l'utilisateur dans chacun des champs, puis les affecter au document sélectionné, et au dvd associé à ce document, qui portent tout les deux le même Id.

C'est grâce à l'appel des méthodes "ModifierDocument" et "ModifierDvd", que le document est modifié dans la base de données.

La fonction permet ensuite d'afficher la base de données mise à jour, avec les dvd et les informations du document associé. Voici un extrait de la procédure événementielle "btnReceptionDvdModifier_Click" :

Procédure évènementielle - btnReceptionDvdModifier_Click

```
if (controller.ModifierDvd(id, synopsis, realisateur, duree)
&& controller.ModifierDocument(id, titre, image, idGenre, idPublic, idRayon)) {
lesDvd = controller.GetAllDvd();
RemplirComboCategorie(controller.GetAllGenres(), bdgGenres, cbxDvdGenres);
RemplirComboCategorie(controller.GetAllPublics(), bdgPublics, cbxDvdPublics);
RemplirComboCategorie(controller.GetAllRayons(), bdgRayons, cbxDvdRayons);
```

```
RemplirDvdListeComplete();

 MessageBox.Show("Le dvd " + titre + " a correctement été modifié.");

 }
```

FONCTIONNALITÉ "MODIFIER" - ONGLET REVUES

Je crée une nouvelle méthode "ModifierRevue", dans la classe "Access.cs", qui va prendre en paramètres les différents champs de la table "revue".

Ces données sont ensuite sérialisées dans une chaîne "json", appelée "jsonModifierRevue", puis envoyées avec la méthode "PUT", sur le endpoint revue/id.

Méthode - ModifierRevue

```
public bool ModifierRevue(string Id, string Periodicite, int DelaiMiseADispo) {

String jsonModifierRevue = "{ \"id\" : \"" + Id + "\", \"periodicite\" : \"" + Periodicite + "\",

\"delaiMiseADispo\" : \"" + DelaiMiseADispo + "\"}";

Console.WriteLine("jsonModifierRevue" + jsonModifierRevue);

try { List liste = TraitementRecup(PUT, "revue/" + Id + "/" + jsonModifierRevue);

 return (liste != null);

} catch (Exception ex) {

Console.WriteLine(ex.Message);

 } return false; }
```

Une fois réalisée, je dois appeler cette fonction dans le contrôleur, j'insère donc la méthode "ModifierRevue, à la classe "FrmMediatekController.cs".

```
 public bool ModifierRevue(string Id, string Periodicite, int DelaiMiseADispo) {
```

return access.ModifierRevue(Id, Periodicite, DelaiMiseADispo);

}

Je peux maintenant aller dans le code de la vue "FrmMediatek.cs", pour effectuer la procédure de modification d'un document et d'une revue, lors du clic sur le bouton "Modifier".

Pour cela, je créé une procédure évènementielle "btnReceptionRevueModifier_Click", lorsqu'une ligne est sélectionnée dans la datagrid "dgvRevuesListe.SelectedRows.Count > 0", l'application va récupérer les données saisies par l'utilisateur dans chacun des champs, puis les affecter au document sélectionné, et à la revue associée à ce document, qui portent tout les deux le même Id.

C'est grâce à l'appel des méthodes "ModifierDocument" et "ModifierRevue", que le document est modifié dans la base de données.

La procédure permet ensuite d'afficher la base de données mise à jour, avec les revues et les informations du document associé.

Voici un extrait de la procédure évènementielle "btnReceptionRevueModifier_Click" :

Procédure évènementielle - btnReceptionRevueModifier_Click

```
if (controller.ModifierRevue(id, periodicite, delaiMiseADispo)
&& controller.ModifierDocument(id, titre, image, idGenre, idPublic, idRayon)) {
 lesRevues = controller.GetAllRevues();
 RemplirComboCategorie(controller.GetAllGenres(), bdgGenres, cbxLivresGenres);
RemplirComboCategorie(controller.GetAllPublics(), bdgPublics, cbxLivresPublics);
RemplirComboCategorie(controller.GetAllRayons(), bdgRayons, cbxLivresRayons);
RemplirRevuesListeComplete();
```

```
MessageBox.Show("La revue " + titre + " a correctement été modifiée.");
}
```

AJOUTER LES FONCTIONNALITÉS "SUPPRIMER"

FONCTIONNALITÉ "SUPPRIMER" - ONGLET LIVRES

Afin de créer la fonctionnalité "supprimer" dans l'onglet "livres", je dois utiliser la fonction de l'api rest_mediatekdocuments, appelée "delete", cette fonction peut être exploitée dans l'application C#, en appelant la méthode "DELETE".

Cette méthode DELETE va permettre de récupérer l'id de la ligne sélectionnée par l'utilisateur dans l'interface de l'application sur les différents onglets, puis de supprimer le contenu des champs de la ligne, dans les tables de la base de données.

Pour cela, il faut que les données saisies soient envoyées à l'api au format JSON.

Je commence donc par réaliser la méthode de suppression d'un livre dans la classe "Access.cs" de l'application C#.

J'ai nommé cette méthode "SupprimerLivre(string Id)", dans laquelle je lui affecte l'id du livre à supprimer.

J'effectue ensuite la sérialisation de l'id de l'api au format json, en l'affectant dans une chaîne json "jsonIdLivre".

Pour avoir une visibilité de la ligne à supprimer, j'ajoute un "console.writeline" qui va afficher l'id récupéré dans la chaîne.

Je supprime ensuite les données en appelant la méthode "DELETE" de l'api, qui va enregistrer la suppression du livre sur le endpoint livre/id .

Méthode - SupprimerLivre

```
 public bool SupprimerLivre(string Id) {

String jsonIdLivre = "{ \"id\" : \"" + Id + "\"}"; Log.Error("jsonIdLivre" + jsonIdLivre);

 try {

// récupération soit d'une liste vide (requête ok) soit de null (erreur)

 List liste = TraitementRecup(DELETE, "livre/" + jsonIdLivre);

return (liste != null);

} catch (Exception ex) {

 Console.WriteLine(ex.Message);

 Log.Error("Access.SupprimerLivre catch jsonIdLivre={0} erreur={1} ", jsonIdLivre, ex.Message); }
return false;

 }
```

Je dois ensuite appeler cette fonction dans le contrôleur, pour cela, je crée une méthode "SupprimerLivre", dans la classe "FrmMediatekController.cs".

```
public bool SupprimerLivre(string Id, { return access.SupprimerLivre(Id); }
```

Je peux maintenant aller dans le code de la vue "FrmMediatek.cs", pour effectuer la procédure de suppression d'un livre, lors du clic sur le bouton "Supprimer".

Pour cela, je créé une procédure évènementielle "btnSupprimerLivre_Click", lorsqu'une ligne est sélectionnée dans la datagrid "bdgLivresListe.Current", l'application va récupérer l'id du livre sélectionné par l'utilisateur, puis supprimer le contenu de la ligne sélectionnée, ainsi que l'id.

C'est grâce à l'appel de la méthode "SupprimerLivre", que le livre est supprimé dans la base de données. La procédure permet ensuite d'afficher la base de données mise à jour, avec les livres restants dans la base de données.

Il est précisé qu'un document ne peut pas être supprimé s' il possède un ou plusieurs exemplaires, ou une ou plusieurs commandes.

Je n'ai pu réaliser cette fonctionnalité qu'après avoir créé les classes métiers nécessaires, je donnerais les explications sur la constitution de ces classes lors de la mission de gestion des commandes.

Afin de réaliser cette condition de suppression, j'ai ajouté des variables qui vérifient si l'id du livre existe dans la liste des exemplaires, et dans la liste des commandes de documents. J'ai pour cela utiliser les getters "GetExemplairesDocument(livre.Id)" et "GetCommandesDocument(livre.Id)", si le livre ne comporte aucun exemplaire, alors la suppression se réalise.

Sinon des messages d'erreurs s'affichent pour expliquer à l'utilisateur qu'il ne peut pas supprimer le livre.

Procédure évènementielle - btnSupprimerLivre_Click

```
private void btnSupprimerLivre_Click(object sender, EventArgs e) {

Livre livre = (Livre)bdgLivresListe.Current;

if (MessageBox.Show("Souhaitez-vous confirmer la suppression?", "Confirmation de suppression", MessageBoxButtons.OKCancel) == DialogResult.OK) {

var exemplairesLivre = controller.GetExemplairesDocument(livre.Id);

var aucunExemplaire = !exemplairesLivre.Any();

var commandesLivre = controller.GetCommandesDocument(livre.Id);

var aucuneCommande = !commandesLivre.Any();

if (aucunExemplaire && aucuneCommande) {

if (controller.SupprimerLivre(livre.Id)) {
```

```
lesLivres = controller.GetAllLivres();

RemplirComboCategorie(controller.GetAllGenres(), bdgGenres, cbxLivresGenres);

RemplirComboCategorie(controller.GetAllPublics(), bdgPublics, cbxLivresPublics);

RemplirComboCategorie(controller.GetAllRayons(), bdgRayons, cbxLivresRayons);

RemplirLivresListeComplete();

 MessageBox.Show($"Le livre {livre.Titre} a correctement été supprimé.");

 } else {

 MessageBox.Show("Une erreur s'est produite.", "Erreur");

 }

 } else {

 MessageBox.Show($"Impossible de supprimer le livre car il possède {(aucunExemplaire ? "une ou plusieurs commande(s)" : "un ou plusieurs exemplaire(s)")}.", "Erreur");

 }}}
```

Il est précisé qu'un document ne peut pas être supprimé s' il possède un ou plusieurs exemplaires, ou une ou plusieurs commandes.

Je n'ai pu réaliser cette fonctionnalité qu'après avoir créé les classes métiers nécessaires, je donnerais les explications sur la constitution de ces classes lors de la mission de gestion des commandes.

Afin de réaliser cette condition de suppression, j'ai ajouté des variables qui vérifient si l'id du livre existe dans la liste des exemplaires, et dans la liste des commandes de documents.

J'ai pour cela utiliser les getters "GetExemplairesDocument(livre.Id)" et "GetCommandesDocument(livre.Id)", si le livre ne comporte aucun exemplaire, alors la suppression se réalise.

Sinon des messages d'erreurs s'affichent pour expliquer à l'utilisateur qu'il ne peut pas supprimer le livre.

Lors de la création de la fonctionnalité de suppression, sur l'onglet "livres", je me suis rendue compte qu'il n'était pas possible de supprimer un livre, si l'id du livre existe dans la table "livres_dvd" et dans la table "document", car les id sont liés entre ces tables.

Afin de pallier à cette contrainte, j'ai dû réaliser des triggers, qui permettent de supprimer l'id d'un livre supprimé, dans la table "document", et dans la table "livres_dvd".

J'en ai profité pour créer les triggers des autres tables.

Trigger - Suppression livre

```
DELIMITER $$
CREATE TRIGGER `delete_livre_to_document` AFTER DELETE ON `livre`
FOR EACH ROW
BEGIN
DELETE FROM livres_dvd WHERE id=OLD.id ;
DELETE FROM document WHERE id=OLD.id ;
END
$$ DELIMITER ;
```

Trigger - Suppression document

```
DELIMITER $$
```

```
CREATE TRIGGER `delete_document_to_livres_dvd` BEFORE DELETE ON `document`
FOR EACH ROW
BEGIN
DELETE FROM livres_dvd WHERE id = OLD.id;
END
$$ DELIMITER ;
```

Trigger - Suppression dvd

```
DELIMITER $$
CREATE TRIGGER `delete_dvd_to_document` AFTER DELETE ON `dvd`
FOR EACH ROW
BEGIN
DELETE FROM livres_dvd WHERE id=OLD.id ;
DELETE FROM document WHERE id=OLD.id ;
END
$$ DELIMITER ;
```

Trigger - Suppression revue

```
DELIMITER $$
CREATE TRIGGER `delete_revue_to_document` AFTER DELETE ON `revue`
FOR EACH ROW
BEGIN DELETE FROM document WHERE id=OLD.id ;
```

```
END$$
 DELIMITER ;
```

Trigger - Suppression abonnement

```
DELIMITER $$
CREATE TRIGGER `delete_abonnement_to_commande` AFTER DELETE ON `abonnement`
 FOR EACH ROW
 BEGIN
DELETE FROM commande WHERE id = OLD.id;
END
$$ DELIMITER ;
```

Trigger - Suppression commandedocument

```
DELIMITER $$
CREATE TRIGGER `delete_commandedocument_to_commande` AFTER DELETE ON
`commandedocument`
 FOR EACH ROW
BEGIN
DELETE FROM commande WHERE id=OLD.id ;
END$$
DELIMITER ;
```

FONCTIONNALITÉ "SUPPRIMER" - ONGLET DVD

Je crée une nouvelle fonction "SupprimerDvd", dans la classe "Access.cs", qui va prendre en paramètre l'id du dvd.

L'id sera ensuite envoyé dans une chaîne "json", appelée "jsonIdDvd", puis supprimé avec la méthode "DELETE", sur le endpoint dvd/id. Fonction permettant la suppression d'un dvd

public bool SupprimerDvd(string Id) { String jsonIdDvd = "{ \"id\" : \"" + Id + "\"}"; Console.WriteLine("jsonIdDvd" + jsonIdDvd);

try {

// récupération soit d'une liste vide (requête ok) soit de null (erreur)

List liste = TraitementRecup(DELETE, "dvd/" + jsonIdDvd); return (liste != null);

} catch (Exception ex) {

Console.WriteLine(ex.Message);

} return false;

Une fois réalisée, je dois appeler cette fonction dans le contrôleur, j'insère donc la méthode "SupprimerDvd, au controller "FrmMediatekController.cs".

Je peux maintenant aller dans le code de la vue "FrmMediatek.cs", pour effectuer la procédure de suppression d'un dvd, lors du clic sur le bouton "Supprimer".

Pour cela, je créé la procédure évènementielle "btnSupprimerDvd_Click", lorsqu'une ligne est sélectionnée dans la datagrid "bdgDvdListe.Current", l'application va récupérer l'id du dvd sélectionné par l'utilisateur, puis supprimer le contenu de la ligne sélectionnée, ainsi que l'id.

C'est grâce à l'appel de la méthode "SupprimerDvd", que le dvd est supprimé dans la base de données.

La procédure permet ensuite d'afficher la base de données mise à jour, avec les dvd et les informations des documents restants dans la base de données.

Là encore j'utilise la même technique que pour la suppression d'un livre, en ajoutant la condition qui va empêcher la suppression d'un dvd s' il possède un ou plusieurs exemplaires ou une ou plusieurs commandes.

Voici un extrait de la procédure évènementielle "btnSupprimerDvd_Click" :

Procédure évènementielle - btnSupprimerDvd_Click

```
if (aucunExemplaireDvd && aucuneCommandeDvd) {
 if (controller.SupprimerDvd(dvd.Id)) {
 lesDvd = controller.GetAllDvd();
RemplirComboCategorie(controller.GetAllGenres(), bdgGenres, cbxDvdGenres);
RemplirComboCategorie(controller.GetAllPublics(), bdgPublics, cbxDvdPublics);
RemplirComboCategorie(controller.GetAllRayons(), bdgRayons, cbxDvdRayons);
RemplirDvdListeComplete();
 MessageBox.Show("Le dvd " + dvd.Titre + " a correctement été supprimé.");
 } else {
 MessageBox.Show("Une erreur s'est produite.", "Erreur");
}
```

FONCTIONNALITÉ "SUPPRIMER" - ONGLET REVUES

Je crée une nouvelle méthode "SupprimerRevue(string Id)", dans la classe "Access.cs", qui va prendre en paramètres l'id de la revue.

L'id sera ensuite envoyé dans une chaîne "json", appelée "jsonIdRevue", puis supprimé avec la méthode "DELETE", sur le endpoint revue/id. Méthode - SupprimerRevue

```
public bool SupprimerRevue(string Id) {
```

```
String jsonIdRevue = "{ \"id\" : \"" + Id + "\"}";

 Console.WriteLine("jsonIdRevue" + jsonIdRevue);

 try {

 // récupération soit d'une liste vide (requête ok) soit de null (erreur)

 List liste = TraitementRecup(DELETE, "revue/" + jsonIdRevue); return (liste != null);

} catch (Exception ex) {

 Console.WriteLine(ex.Message);

 } return false;
```

Une fois réalisée, je dois appeler cette fonction dans le contrôleur, j'insère donc la méthode "SupprimerRevue à la classe "FrmMediatekController.cs".

```
 public bool SupprimerRevue(string Id) {

return access.SupprimerRevue(Id); }
```

Je peux maintenant aller dans le code de la vue "FrmMediatek.cs", pour effectuer la procédure de suppression d'une revue, lors du clic sur le bouton "Supprimer".

Pour cela, je créé la procédure évènementielle "btnSupprimerRevue_Click", lorsqu'une ligne est sélectionnée dans la datagrid "bdgRevuesListe.Current", l'application va récupérer l'id de la revue sélectionnée par l'utilisateur, puis supprimer le contenu de la ligne sélectionnée, ainsi que l'id.

C'est grâce à l'appel de la méthode "SupprimerRevue", que la revue est supprimée dans la base de données.

Pour les revues, il faut ajouter une condition qui permette de ne pas supprimer les revues si elles ont un ou plusieurs exemplaires associés.

Par conséquent, j'ai récupéré les exemplaires existants dans la fonction, puis j'ai effectué ma condition permettant de ne pas faire la suppression si l'id d'un exemplaire est égal à celui d'une revue, car ces deux listes possèdent le même id.

La fonction permet ensuite d'afficher la base de données mise à jour, avec les revues restantes dans la base de données.

Voici un extrait de la procédure évènementielle "btnSupprimerRevue_Click" :

Procédure évènementielle - btnSupprimerRevue_Click

```
List lesExemplairesRevue = controller.GetExemplairesRevue(revue.Id);
bool aucunExemplaire = true;
foreach (Exemplaire exemplaire in lesExemplairesRevue) {
if (exemplaire.Id == revue.Id) { aucunExemplaire = false; break;
} }
```

MISSION 2. GÉRER LES COMMANDES

TÂCHE 1 : GÉRER LES COMMANDES DE LIVRES

Dans la base de données, créer la table 'Suivi' qui contient les différentes étapes de suivi d'une commande de document de type livre ou dvd.

Relier cette table à CommandeDocument.

Créer un onglet (ou une nouvelle fenêtre) pour gérer les commandes de livres.

La charte graphique doit correspondre à l'existant.

Dans toutes les listes, permettre le tri sur les colonnes.

Dans l'onglet (ou la fenêtre), permettre la sélection d'un livre par son numéro, afficher les informations du livre ainsi que la liste des commandes, triée par date (ordre inverse de la chronologie).

La liste doit comporter les informations suivantes : date de la commande, montant, nombre d'exemplaires commandés et l'étape de suivi de la commande.

Créer un groupbox qui permet de saisir les informations d'une nouvelle commande et de l'enregistrer.

Lors de l'enregistrement de la commande, l'étape de suivi doit être mise à "en cours".

Permettre de modifier l'étape de suivi d'une commande en respectant certaines règles (une commande livrée ou réglée ne peut pas revenir à une étape précédente (en cours ou relancée), une commande ne peut pas être réglée si elle n'est pas livrée).

Permettre de supprimer une commande uniquement si elle n'est pas encore livrée.

Faire un trigger qui réalise aussi la suppression dans la classe fille.

Créer le trigger qui se déclenche si une commande passe à l'étape "livrée" et qui crée autant de tuples dans la table "Exemplaire" que nécessaires, en valorisant la date d'achat avec la date de commande et en mettant l'état de l'exemplaire à "neuf".

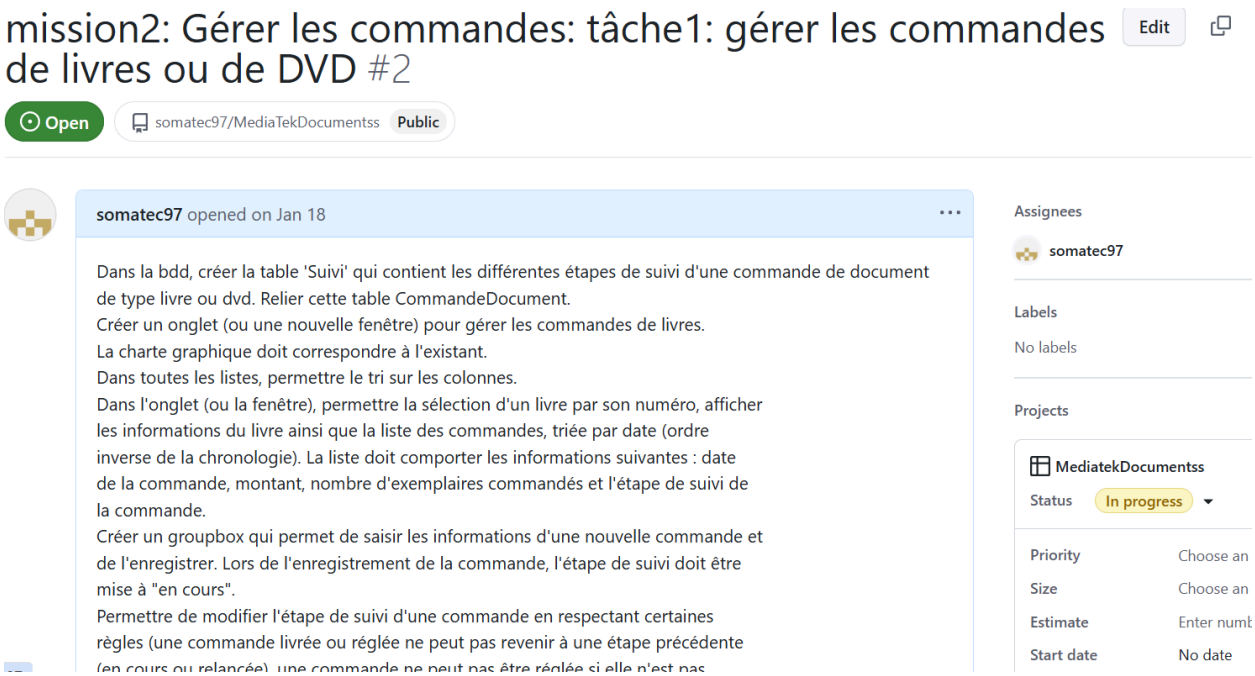
Le numéro d'exemplaire doit être séquentiel par rapport au livre concerné.

Créer un onglet pour gérer les commandes de DVD en suivant la même logique que pour les commandes de livres.

Toutes les sécurités seront mises en place pour éviter des erreurs de manipulation.

Créer le trigger qui contrôle la contrainte de partition de l'héritage sur Commande.

Temps estimé 8h - Temps mis environ 15h .



COMMANDES DE LIVRES

Je commence par créer le nouvel onglet "Commandes de livres", et je construis le design de l'interface concernant les commandes de livres, en m'inspirant du design de l'onglet "Parutions des revues".

Cet onglet doit permettre de saisir le numéro d'un livre, puis de rechercher s' il existe des commandes le concernant.

Les commandes doivent être affichées dans la datagrid, et les informations du livre doivent apparaître dans la groupbox des informations détaillées.

Cette groupbox est non-modifiable puisqu'on ne cherche pas à modifier le livre.

Les informations concernant la commande sélectionnée dans la grille, doivent être récupérées dans une groupbox "Informations de commande".

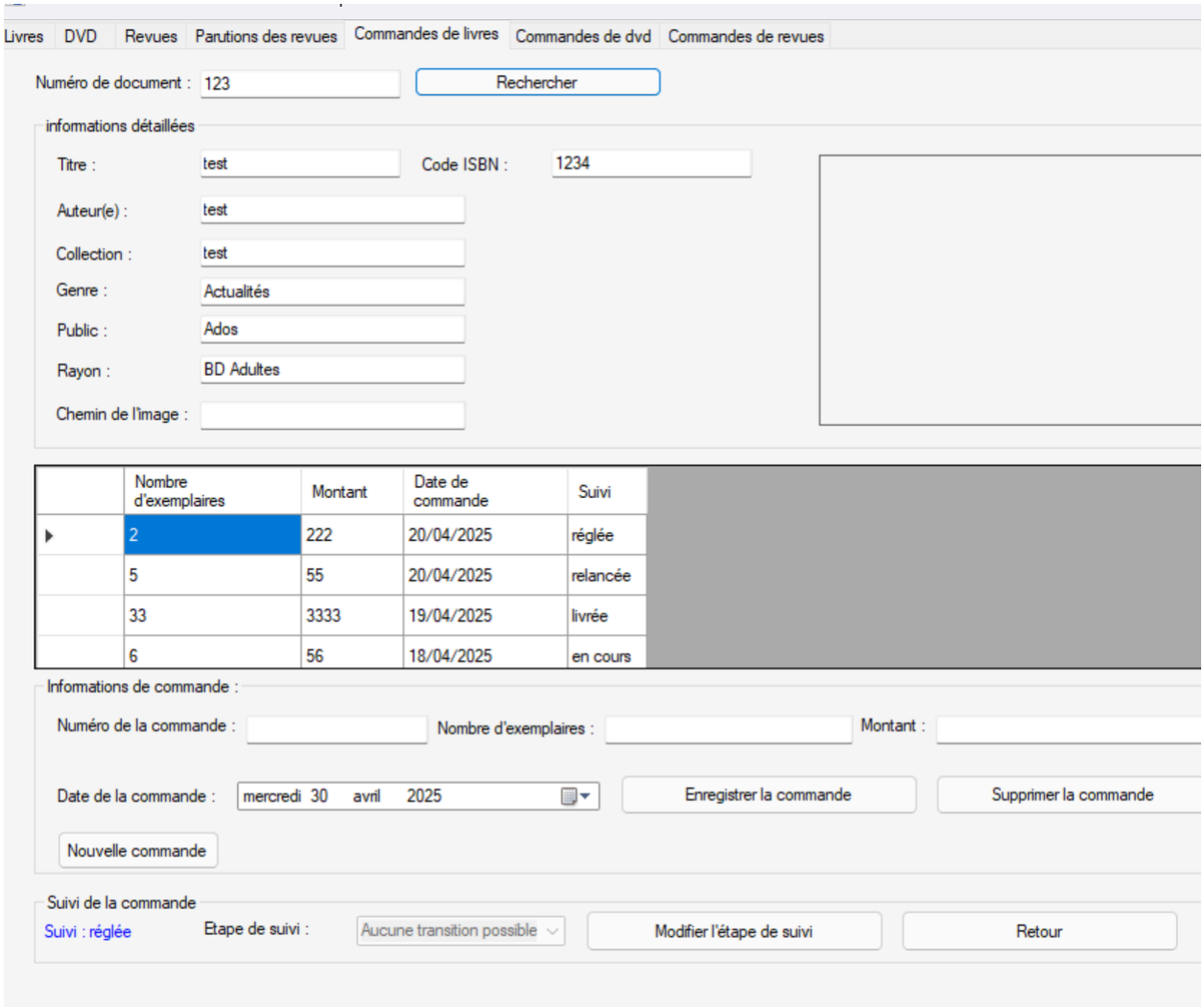
C'est dans cette groupbox que se feront toutes les configurations d'une commande.

Elle sera donc inaccessible tant que l'utilisateur n'aura pas saisi un numéro de document valide.

Pour ajouter une nouvelle commande, on a besoin d'un numéro de commande, du nombre d'exemplaires, du montant, et de la date de commande.

Je créé un bouton "Enregistrer une nouvelle commande". Il doit également être possible de supprimer une commande, donc je positionne un bouton "Supprimer la commande". Il est notamment stipulé que l'utilisateur doit pouvoir modifier l'étape de suivi d'une commande, selon certaines règles.

Je place alors un libellé "étape" qui contiendra l'étape de suivi actuelle de la commande, en dessous je construis une comboBox qui possède les étapes de suivi selon l'étape actuelle, puis j'ajoute le bouton "Modifier l'étape de suivi". Le bouton "Nouvelle commande" va permettre à l'utilisateur de gérer le suivi de la commande, et le bouton "Retour", servira à revenir dans les informations de commandes.



J'initialise l'affichage des listes en déclarant les variables qui contiendront les listes et qui permettront par la suite de remplir la datagrid ainsi que la comboBox des suivis, dans la procédure évènementielle "tabCommandesLivres_Enter".

Après avoir constitué la table "suivi" dans la base de données, j'ajoute la fonction qui va permettre de récupérer les commandes de documents selon leur id, dans l'api rest_mediatekdocuments déjà pré-conçue. J'insère donc la fonction "selectAllCommandesDocument($id)" , dans laquelle j'affecte le paramètre "$id", et qui va

sélectionner et récupérer les données des champs correspondants au livre, au suivi, à la table commande, et à la table commandedocument, triées par ordre descendant.

Requête - selectAllCommandesDocument($id)

```
public function selectAllCommandesDocument($id) {

$param = array ( "id" => $id );

$req = "select l.nbExemplaire, l.idLivreDvd, l.idSuivi, s.libelle, l.id, max(c.dateCommande) as dateCommande, sum(c.montant) as montant ";

$req .= "from commandedocument l join suivi s on s.id=l.idSuivi ";

$req .= "left join commande c on l.id=c.id ";

$req .= "where l.idLivreDvd = :id ";

$req .= "group by l.id ";

$req .= "order by dateCommande DESC";

return $this->conn->query($req, $param);

}
```

Cette fonction est ensuite appelée dans la requête "selectOne($table, $id)" de l'api Rest PHP, dans le case "commandedocument", qui va ensuite pouvoir être exploité dans la classe "Access" de l'application C#.

J'ajoute également le case "commandedocument", dans la fonction "selectAll($table)" pour récupérer toutes les commandes de document.

Pour que la datagrid affiche la liste des commandes concernant un livre recherché, je dois créer trois nouvelles méthodes dans l'application C#.

Avant de réaliser ces méthodes, il a d'abord été nécessaire de constituer les entités qui correspondent à la table "commande" et à la table "commandedocument".

Dans le dossier "Model", j'ajoute donc les classes métiers "Commande" et "CommandeDocument", j'affecte à chacune d'elle les propriétés qui leurs sont associés, j'étends la classe Commande à la classe CommandeDocument, car la classe "Commande" contient les propriétés de base de chaque commande.

Cela me permet d'initialiser les propriétés de la classe Commande. Ensuite, dans la vue FrmMediatek, je réalise la méthode qui va permettre de remplir la grille avec la liste des commandes du livre.

J'appelle cette méthode "RemplirCommandesLivresListe(List lesCommandesDocument)", et je lui affecte en paramètre la liste d'objets "CommandeDocument", récupérée dans la variable "lesCommandesDocument".

Il faut pour cela, que la liste ne soit pas nulle, les propriétés sont ensuite insérées dans des colonnes de la grille contenant les données "dgvCommandesLivres".

Les colonnes qui n'ont pas besoin de s'afficher sont masquées, et j'ai ajouté un "AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;", pour permettre l'ajustement des colonnes selon le contenu, obtenant ainsi une meilleure visibilité.

La seconde méthode consiste à mettre à jour l'affichage de la liste des commandes du livre qui sera saisi dans le champ de recherche "txbCommandesLivresNumRecherche.Text".

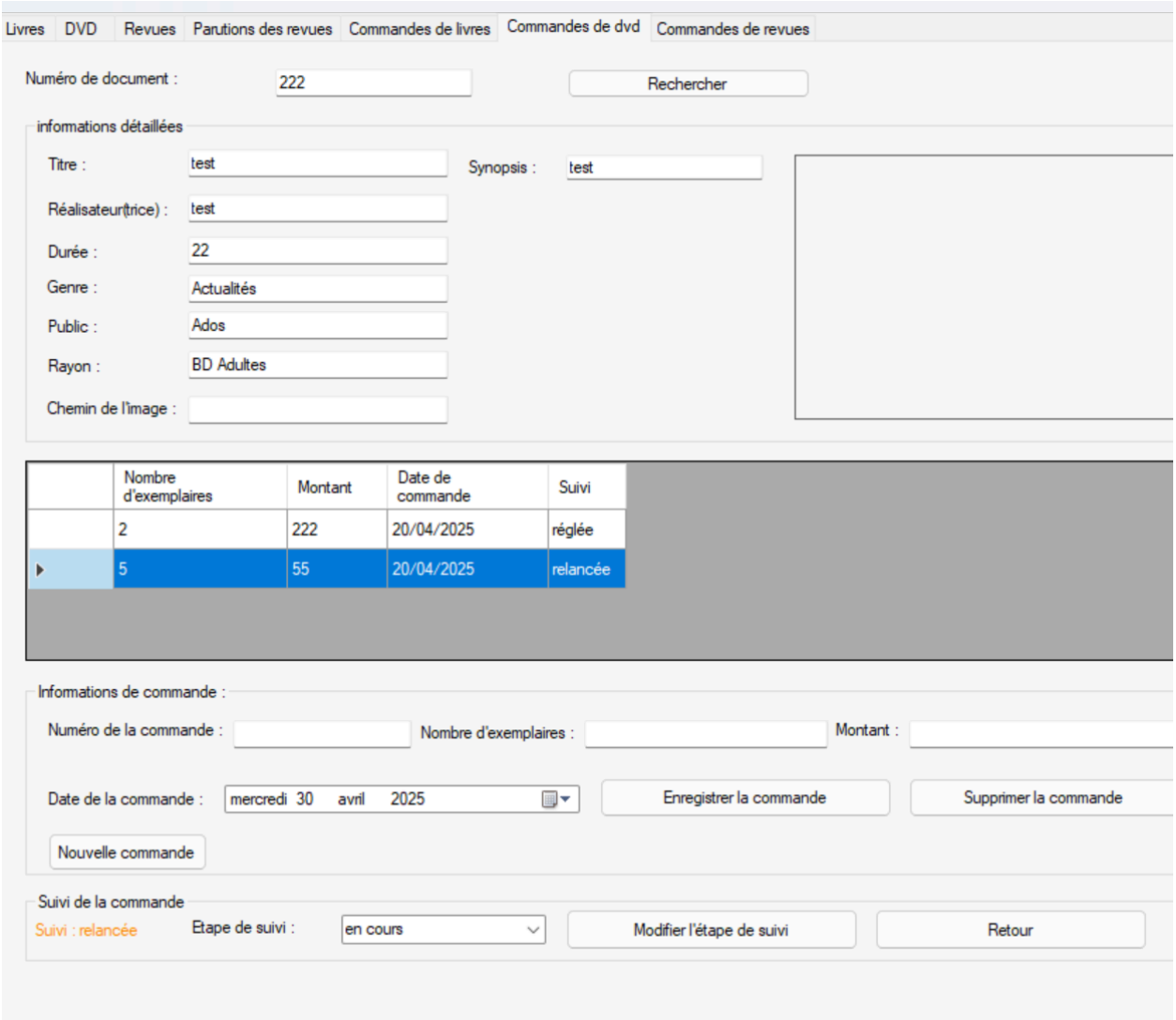
J'appelle cette méthode "AfficheReceptionCommandesLivre()", j'affecte dans la variable "idDocument", la valeur du champ de recherche du numéro de document.

Dans la variable "lesCommandesDocument", j'appelle le getter qui va récupérer la liste d'objets des commandes concernant le document, pour cela je lui affecte en paramètre la variable "idDocument" qui va contenir l'id du document saisi par l'utilisateur.

Enfin, j'appelle la méthode "RemplirCommandesLivresListe(lesCommandesDocument)" qui va mettre à jour le remplissage de la grille, selon les commandes du document concerné, puisque je lui ai affecté la variable contenant les commandes du document.

Méthode - RemplirCommandesLivresListe

Méthode - AfficheReceptionCommandesLivre



TÂCHE 2 : GÉRER LES COMMANDES DE REVUES

Créer un onglet (ou une fenêtre) pour gérer les commandes de revues : une commande représente un nouvel abonnement ou le renouvellement d'un abonnement.

Dans le cas d'un nouvel abonnement, la revue sera préalablement créée dans l'onglet Revues.

Donc, dans l'onglet des commandes de revues, il n'y a pas de distinction entre un nouvel abonnement et un renouvellement.

La charte graphique doit correspondre à l'existant.

Dans toutes les listes, permettre le tri sur les colonnes.

Permettre la sélection d'une revue par son numéro, afficher les informations de la revue ainsi que la liste des commandes (abonnements), triée par date (ordre inverse de la chronologie).

La liste doit comporter les informations suivantes : date de la commande, montant et date de fin d'abonnement.

Créer un groupbox qui permet de saisir les informations d'une nouvelle commande (nouvel abonnement ou renouvellement, le principe est identique) et de l'enregistrer.

Permettre de supprimer une commande de revue uniquement si aucun exemplaire n'est rattaché (donc, en vérifiant la date de l'exemplaire, comprise entre la date de la commande et la date de fin d'abonnement).

Pour cela, créer et utiliser la méthode 'ParutionDansAbonnement' qui reçoit en paramètre 3 dates (date commande, date fin abonnement, date parution) et qui retourne vrai si la date de parution est entre les 2 autres dates.

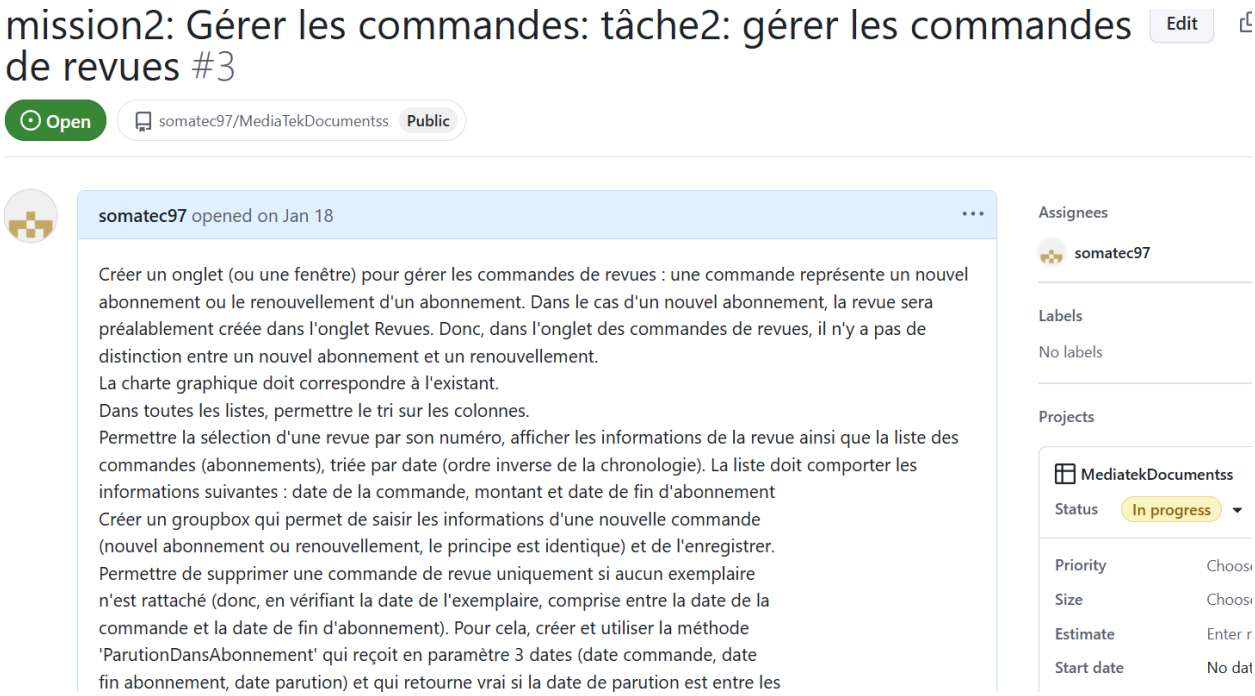
Créer le test unitaire sur cette méthode.

Toutes les sécurités seront mises en place pour éviter des erreurs de manipulation.

Créer une méthode qui permet d'obtenir la liste des revues dont l'abonnement se termine dans moins de 30 jours.

Dès l'ouverture de l'application, ouvrir une petite fenêtre d'alerte rappelant la liste de ces revues (titre et date de fin abonnement) triée sur la date dans l'ordre chronologique.

Temps estimé à 4h - Temps mis environ 15h(bug de l'api).



Je commence par créer le nouvel onglet "Commandes de revues", et je construis le design de l'interface concernant les commandes de revues, en m'inspirant du design de l'onglet "Parutions des revues".

Cet onglet doit permettre de saisir le numéro d'une revue, puis de rechercher s' il existe des commandes la concernant.

Les commandes doivent être affichées dans la datagrid, et les informations de la revue doivent apparaître dans la groupbox des informations détaillées.

Cette groupbox est non-modifiable puisqu'on ne cherche pas à modifier la revue.

Les informations concernant la commande sélectionnée dans la grille, doivent être récupérées dans une groupbox "Informations de commande".

C'est dans cette groupbox que se feront toutes les configurations d'une commande.

Elle sera donc inaccessible tant que l'utilisateur n'aura pas saisi un numéro de document valide.

Pour ajouter une nouvelle commande, on a besoin d'un numéro de commande, du montant, de la date de commande et de la date de fin de l'abonnement, car les commandes de revues fonctionnent par "abonnement".

Je créé un bouton "Ajouter une nouvelle commande" et comme il doit être possible de supprimer une commande/abonnement, je positionne un bouton "Supprimer la commande".

phto app comd revues_____

J'initialise l'affichage des listes en déclarant les variables qui contiendront les listes et qui permettront par la suite de remplir la datagrid dans la procédure évènementielle "tabCommandesRevues_Enter".

L'application C#, doit récupérer les données de la base de données, et donc exploiter les fonctions de l'api rest_mediatekdocuments fournies et déjà pré-conçues.

Je dois y ajouter la fonction qui va permettre de récupérer les abonnements d'une revue, selon leur id.

J'insère donc la fonction "selectAllAbonnementsRevues($id)" , dans laquelle j'affecte le paramètre "$id", et qui va sélectionner et récupérer les données des champs correspondants à à la table "abonnement", et à la table "commande", triées par ordre descendant.

Fonction - selectAllAbonnementsRevues($id)

public function selectAllAbonnementsRevues($id){

```
$param = array( "id" => $id );

$req = "select c.id, c.dateCommande, c.montant, a.dateFinAbonnement, a.idRevue ";

$req .= "from commande c join abonnement a ON c.id=a.id ";

$req .= "where a.idRevue= :id ";

$req .= "order by c.dateCommande DESC ";

return $this->conn->queryAll($req, $param);

}
```

Cette fonction est ensuite appelée dans la fonction "selectOne($table, $id)" de l'api, dans le case "abonnement", qui va ensuite pouvoir être exploité dans la classe "Access" de l'application C#.

Pour que la datagrid affiche la liste des abonnements concernant une revue recherchée, je dois créer trois nouvelles méthodes.

Avant de réaliser ces méthodes, il a d'abord été nécessaire de constituer l'entité qui correspond à la table "abonnement" de la base de données.

Dans le dossier "Model", j'ajoute donc la classe métier "Abonnement", je lui affecte les propriétés qui lui sont associées, et j'étends la classe Commande à la classe Abonnement, car la classe "Commande" contient les propriétés de base de chaque commande.

Cela me permet d'initialiser les propriétés de la classe Commande.

Ensuite, je réalise la méthode qui va permettre de remplir la grille avec la liste des abonnements de la revue.

J'appelle cette méthode "RemplirAbonnementsRevueListe(List lesAbonnementsRevue)", et je lui affecte en paramètre la liste d'objets "Abonnement", récupérée dans la variable "lesAbonnementsRevue".

Il faut pour cela, que la liste ne soit pas nulle, les propriétés sont ensuite insérées dans des colonnes de la grille contenant les données "dgvAbonnementsRevue".

Les colonnes qui n'ont pas besoin de s'afficher sont masquées, et j'ai ajouté un "AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;", pour permettre l'ajustement des colonnes selon le contenu, obtenant ainsi une meilleure visibilité.

Méthode - RemplirAbonnementsRevueListe

```
private void RemplirAbonnementsRevueListe(List lesAbonnementsRevue) {
if (lesAbonnementsRevue != null) {
bdgAbonnementsRevue.DataSource = lesAbonnementsRevue;
dgvAbonnementsRevue.DataSource = bdgAbonnementsRevue;
dgvAbonnementsRevue.Columns["id"].Visible = false;
dgvAbonnementsRevue.Columns["idRevue"].Visible = false;
dgvAbonnementsRevue.Columns["titre"].Visible = false;
dgvAbonnementsRevue.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
dgvAbonnementsRevue.Columns["dateCommande"].DisplayIndex = 0;
dgvAbonnementsRevue.Columns["montant"].DisplayIndex = 1;
dgvAbonnementsRevue.Columns[4].HeaderCell.Value = "Date de commande";
dgvAbonnementsRevue.Columns[0].HeaderCell.Value = "Date de fin d'abonnement";
}
```

La seconde méthode, consiste à mettre à jour l'affichage de la liste des abonnements de la revue qui sera saisie dans le champ de recherche "txbCommandesRevueNumRecherche".

J'appelle cette méthode "AfficheReceptionAbonnementsRevue", j'affecte dans la variable "idDocument", la valeur du champ de recherche du numéro de revue.

Dans la variable "lesAbonnementsRevue", j'appelle le getter qui va récupérer la liste d'objets des abonnements concernant la revue, pour cela je lui affecte en paramètre la variable "idDocument" qui va contenir l'id de la revue saisi par l'utilisateur.

Enfin, j'appelle la méthode "RemplirAbonnementsRevueListe" qui va mettre à jour le remplissage de la grille, selon les commandes de la revue concernée.

Méthode - AfficheReceptionAbonnementsRevue

```
private void AfficheReceptionAbonnementsRevue() {

string idDocument = txbCommandesRevueNumRecherche.Text;

lesAbonnementsRevue = controller.GetAbonnementRevue(idDocument);

RemplirAbonnementsRevueListe(lesAbonnementsRevue); }
```

La troisième méthode est une procédure événementielle qui va déclencher, lors du clic sur le bouton "rechercher", l'affichage des abonnements concernant la revue qui a été recherché dans le champ "txbCommandesRevueNumRecherche", dans la datagrid.

J'appelle cette méthode "btnCommandesRevueNumRecherche_Click".

Je commence par y insérer une condition

"if (!"txbCommandesRevueNumRecherche.Text.Equals(""))", qui rend la saisie d'un numéro de revue obligatoire pour pouvoir effectuer la recherche.

Ensuite , j'utilise la méthode "Find()" pour retrouver la revue qui correspond à l'id saisi dans la champ de recherche, dans la liste "lesRevues".

Si la revue existe, j'appelle la fonction "AfficheReceptionAbonnementsRevue()", qui va afficher la liste des abonnements concernant cette revue.

J'active ensuite la zone d'affichage des informations de commande concernant cette revue, pour que l'utilisateur puisse ensuite accéder aux ajouts, modifications, ou suppression d'un abonnement.

Procédure évènementielle - btnCommandesRevueNumRecherche_Click

```
private void btnCommandesRevueNumRecherche_Click(object sender, EventArgs e) {
 if (!txbCommandesRevueNumRecherche.Text.Equals("")) {
 Revue revue = lesRevues.Find(x => x.Id.Equals(txbCommandesRevueNumRecherche.Text));
 if (revue != null) {
AfficheReceptionAbonnementsRevue();
 gbxInfosCommandeRevue.Enabled = true;
 } else {
 MessageBox.Show("Ce numéro de revue n'existe pas.");
} } else {
 MessageBox.Show("Le numéro de revue est obligatoire."); }
 }
```

Une fois que l'affichage des abonnements de la revue concernée fonctionne correctement, je crée une méthode "AfficheReceptionAbonnementsRevueInfos(Revue revue)", qui va afficher les informations détaillées concernant la revue recherchée.

Je lui affecte l'objet "Revue" en paramètre, qui permet de remplir les textBox avec les informations de la revue, grâce aux propriétés de l'objet "Revue".

J'appelle cette fonction dans la procédure "btnCommandesRevueNumRecherche_Click", pour que lors du clic sur le bouton "Recherche", les informations détaillées de la revue s'affichent dans les champs respectifs.

Méthode - AfficheReceptionAbonnementsRevueInfos

```
 private void AfficheReceptionAbonnementsRevueInfos(Revue revue) {
txbCommandeRevueTitre.Text = revue.Titre;
 txbCommandeRevuePeriodicite.Text = revue.Periodicite;
 txbCommandeRevueDelai.Text = revue.DelaiMiseADispo.ToString();
 txbCommandeRevueGenre.Text = revue.Genre;
 txbCommandeRevuePublic.Text = revue.Public;
txbCommandeRevueRayon.Text = revue.Rayon;
 txbCommandeRevueCheminImage.Text = revue.Image;
 string image = revue.Image;
 try {
pictureBox4.Image = Image.FromFile(image);
 } catch {
pictureBox4.Image = null;
} AfficheReceptionAbonnementsRevue();
}
```

Je réalise ensuite la procédure évènementielle "dgvAbonnementsRevue_RowEnter" qui va afficher les informations de l'abonnement sélectionné dans la datagrid, dans la groupBox des informations de commande.

Selon les informations contenues dans les lignes de chaque colonne, les champs doivent se remplir lors de la sélection d'une ligne de commande.

J'affecte donc les valeurs dans des variables, lesquelles seront ensuite affectées aux différents champs des informations de commande.

Procédure évènementielle - dgvAbonnementsRevue_RowEnter

```
private void dgvAbonnementsRevue_RowEnter(object sender, DataGridViewCellEventArgs e) {

DataGridViewRow row = dgvAbonnementsRevue.Rows[e.RowIndex];

string id = row.Cells["Id"].Value.ToString();

DateTime dateCommande = (DateTime)row.Cells["dateCommande"].Value;

double montant = double.Parse(row.Cells["Montant"].Value.ToString());

DateTime dateFinAbonnement = (DateTime)row.Cells["DateFinAbonnement"].Value;

txbCommandeRevueNumero.Text = id;

txbCommandeRevueMontant.Text = montant.ToString();

dtpCommandeRevue.Value = dateCommande;

dtpCommandeRevueAbonnementFin.Value = dateFinAbonnement; }
```

Enfin, je construis la procédure évènementielle "dgvAbonnementsRevue_ColumnHeaderMouseClick", qui va permettre de réaliser le tri sur les colonnes, lors du clic sur le header des colonnes, dans l'ordre inverse de la chronologie.

L'affichage des données étant réalisé, je peux dorénavant passer à l'ajout des fonctionnalités Ajouter et Supprimer, de l'onglet des commandes de revues.

Je commence par réaliser la fonctionnalité "Ajouter", qui doit permettre d'ajouter un nouvel abonnement et une nouvelle commande de revue, dans la base de données.

Pour envoyer les données dans les tables "commande" et "abonnement", je dois créer la fonction "CreerAbonnementRevue(string id, DateTime dateFinAbonnement, string idRevue)" dans la classe "Access", qui va grâce à une requête http "POST", écrire un nouvel abonnement, dans la base de données.

J'affecte à cette méthode les paramètres "id, nbExemplaire, idLivreDvd, et idSuivi", puis je créé la chaîne json correspondante. Je convertis le paramètre "dateFinAbonnement" au format Json à l'aide de la bibliothèque "Newtonsoft.Json" et des méthodes déjà pré-conçues dans l'applicationn en précisant la classe "CustomDateTimeConverter" pour permettre la conversion du format de la date au format "yyyy_MM_dd".

Il fallait nécessairement que je la transforme au format json, pour que la création de l'abonnement puisse se faire. Je réalise ensuite la requête HTTP POST, qui va envoyer les données du nouvel objet "Abonnement" sur le endpoint "abonnement/" auquel j'ajoute la chaîne json.

Méthode - CreerAbonnementRevue

```
public bool CreerAbonnementRevue(string id, DateTime dateFinAbonnement, string idRevue) {

 String jsonDateCommande = JsonConvert.SerializeObject(dateFinAbonnement, new
CustomDateTimeConverter());

 String jsonCreerAbonnementRevue = "{\"id\":\"" + id + "\", \"dateFinAbonnement\" : " +
jsonDateCommande + ", \"idRevue\" : \"" + idRevue + "\"}";

Console.WriteLine("jsonCreerAbonnementRevue" + jsonCreerAbonnementRevue);

 try {
```

```
// récupération soit d'une liste vide (requête ok) soit de null (erreur)

List liste = TraitementRecup(POST, "abonnement/" + jsonCreerAbonnementRevue);

 return (liste != null);

} catch (Exception ex) {

Console.WriteLine(ex.Message); }

return false;

 }
```

J'appelle ensuite cette méthode dans le "FrmMediatekController", puis je créé la procédure évènementielle "btnReceptionAbonnementRevueValider_Click", qui, lors du clic sur le bouton "Ajouter", va enregistrer un nouvel abonnement de revue dans la base de données.

Pour que l'enregistrement d'un nouvel abonnement de revue se fasse, il faut obligatoirement saisir un numéro de commande, un montant, la date de commande, et la date de fin de l'abonnement.

Je commence donc par créer la condition "if" qui va autoriser ou non l'insertion d'un nouvel abonnement de revue et d'une nouvelle commande dans la base de données. Je déclare les variables qui vont contenir les valeurs des champs de saisie, et j'affecte la nouvelle commande à la liste "Commande" dans la variable "commande", puis le nouvel abonnement à la liste "Abonnement" dans la variable "abonnement".

J'ajoute ensuite un second "if" qui va récupérer la méthode "CreerCommande" dans laquelle j'intègre la variable "commande" qui contient les données de la nouvelle commande.

Le controller est de nouveau appelé pour récupérer la méthode "CreerAbonnementRevue" dans laquelle j'affecte les variables du nouvel objet "Abonnement".

Puis je mets à jour l'affichage de la grille qui va contenir le nouvel enregistrement en appelant la méthode "AfficheReceptionCommandesLivre();", j'ajoute une sécurité pour que la commande ne soit pas ajoutée si l'id de la commande existe déjà.

Procédure évènementielle - btnReceptionAbonnementRevueValider_Click

```
private void btnReceptionAbonnementRevueValider_Click(object sender, EventArgs e) {

if (!txbCommandeRevueNumero.Text.Equals("") &&
!txbCommandeRevueMontant.Text.Equals("")) {

string idRevue = txbCommandesRevueNumRecherche.Text;

string id = txbCommandeRevueNumero.Text;

double montant = double.Parse(txbCommandeRevueMontant.Text);

DateTime dateCommande = dtpCommandeRevue.Value; DateTime dateFinAbonnement =
dtpCommandeRevueAbonnementFin.Value;

Commande commande = new Commande(id, dateCommande, montant);

var idCommandeRevueExistante = controller.GetCommandes(id);

var idCommandeRevueNonExistante = !idCommandeRevueExistante.Any();

if (idCommandeRevueNonExistante) {

if (controller.CreerCommande(commande)) {

controller.CreerAbonnementRevue(id, dateFinAbonnement, idRevue);

MessageBox.Show("La commande " + id + " a bien été enregistrée.", "Information");
AfficheReceptionAbonnementsRevue(); }

} else {
```

MessageBox.Show("Le numéro de la commande existe déjà, veuillez saisir un nouveau numéro.", "Erreur");

} }

J'ajoute ensuite un second "if" qui va récupérer la méthode "CreerCommande" dans laquelle j'intègre la variable "commande" qui contient les données de la nouvelle commande.

Le controller est de nouveau appelé pour récupérer la méthode "CreerAbonnementRevue" dans laquelle j'affecte les variables du nouvel objet "Abonnement".

Puis je mets à jour l'affichage de la grille qui va contenir le nouvel enregistrement en appelant la méthode "AfficheReceptionCommandesLivre();", j'ajoute une sécurité pour que la commande ne soit pas ajoutée si l'id de la commande existe déjà.

Je peux désormais réaliser la fonction "SupprimerAbonnementRevue(Abonnement abonnement)", dans la classe "Access", qui va récupérer l'id de l'objet "Abonnement" à supprimer grâce à la requête http DELETE, dans laquelle j'affecte le endpoint "abonnement/" et la chaîne au format json.

Méthode - SupprimerAbonnementRevue(Abonnement abonnement)

public bool SupprimerAbonnementRevue(Abonnement abonnement) {

String jsonSupprimerAbonnementRevue = "{\"id\":\"" + abonnement.Id + "\"}"; try {

List liste = TraitementRecup(DELETE, "abonnement/" + jsonSupprimerAbonnementRevue);

return (liste != null);

} catch (Exception ex) {

Console.WriteLine(ex.Message);

} return false; }

J'appelle ensuite cette méthode dans le controller, puis je crée la procédure évènementielle btnAbonnementRevueSupprimer_Click", qui va permettre de supprimer l'abonnement de la revue sélectionnée dans la datagrid, si elle ne comporte aucun exemplaire.

Pour cela, je créé d'abord une méthode"ParutionDansAbonnement(DateTime dateCommande, DateTime dateFinAbonnement, DateTime dateParut ion)" qui reçoit en paramètre les dates de commande, de fin d'abonnement et de parution.

Elle va permettre de vérifier la date de l'exemplaire comprise entre la date de commande et la date de fin d'abonnement.

Elle retourne "true" si la date de parution est entre la date de début et la date de fin de l'abonnement.

Méthode - ParutionDansAbonnement

public bool ParutionDansAbonnement(DateTime dateCommande, DateTime dateFinAbonnement, DateTime dateParution) { return (DateTime.Compare(dateCommande, dateParution) < 0 && DateTime.Compare(dateParution, dateFinAbonnement) < 0); }

Je crée ensuite une seconde méthode "VerificationExemplaire(Abonnement abonnement)" qui va vérifier si aucun exemplaire n'est rattaché à un abonnement de revue, et dans laquelle je vais appeler la méthode "ParutionDansAbonnement".

Méthode - VerificationExemplaire(Abonnement abonnement)

public bool VerificationExemplaire(Abonnement abonnement) { List lesExemplaires = controller.GetExemplairesRevue(abonnement.IdRevue);

bool datedeparution = false; foreach (Exemplaire exemplaire in

lesExemplaires.Where(exemplaires => ParutionDansAbonnement(abonnement.DateCommande, abonnement.DateFinAbonnement, exemplaires.DateAchat))) { datedeparution = true;

} return !datedeparution;

}

Je réalise enfin la procédure évènementielle "btnAbonnementRevueSupprimer_Click" sur le clic du bouton "Supprimer", et dans laquelle j'ajoute une condition qui va appeler la méthode "VerificationExemplaire" permettant de vérifier si un exemplaire est rattaché à l'abonnement de la revue.

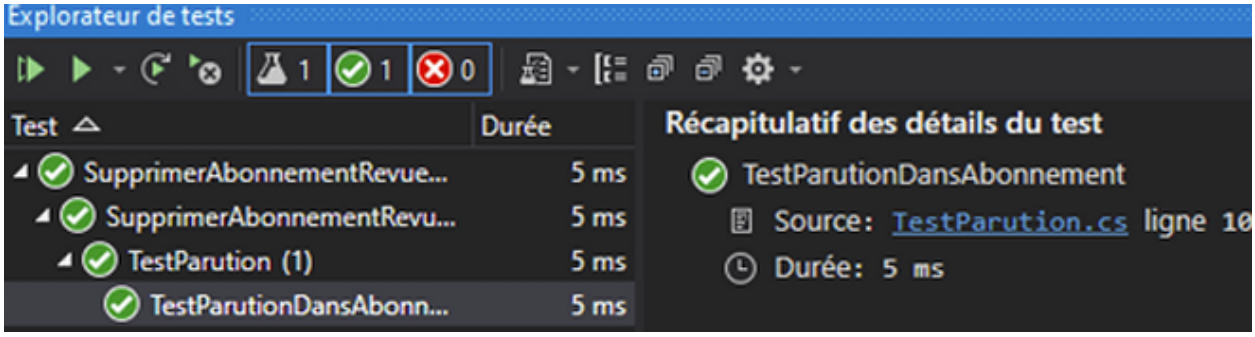
J'y ajoute une MessageBox qui va préciser à l'utilisateur que l'abonnement contient des exemplaires et ne peut donc pas être supprimé, lorsqu'il tentera de supprimer un abonnement qui en contient.

Je dois maintenant créer le test unitaire sur la méthode "ParutionDansAbonnement", pour cela j'ajoute un nouveau projet de type "test unitaire" au projet actuel, que j'appelle Controller-ParutionTest".

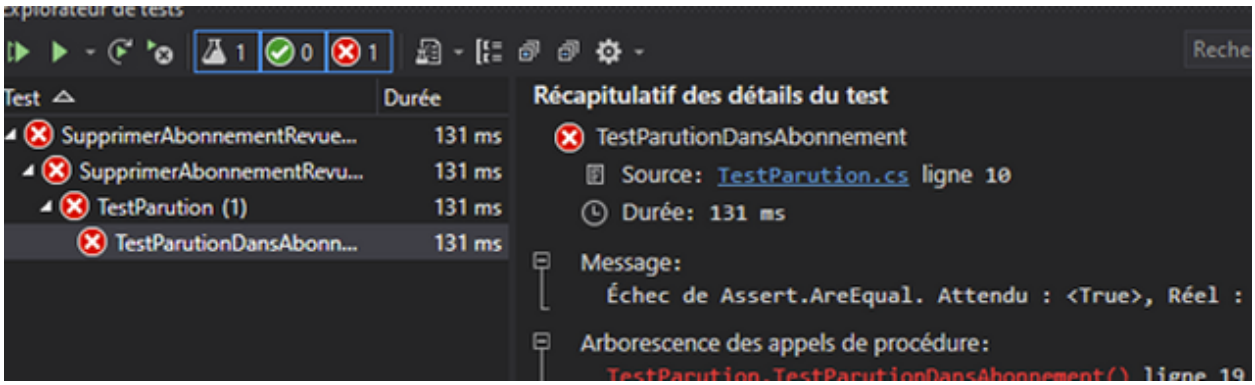
Dans la classe de test "ParutionTest", je créé la méthode "TestParutionDansAbonnement", et j'affecte des valeurs de type "DateTime", dans les variables "dateCommande", "dateFinAbonnement" et "dateParution".

J'affecte au résultat attendu la valeur "true", puis j'affecte le résultat qui va apparaître après l'exécution de la fonction, dans la variable "resultatActuel".

J'ajoute ensuite la méthode "ParutionDansAbonnement" utilisée dans le projet "MediaTekDocuments", puis j'exécute le test. Lorsque la date de parution est comprise entre la date de commande et la date de fin d'abonnement, le test est réussi.



Lorsqu'elle n'est pas comprise entre les deux dates, le test échoue.



Test unitaire - TestParutionDansAbonnement()

```
public class TestParution
{
[TestMethod]
public void TestParutionDansAbonnement()
{
DateTime dateCommande = new DateTime(2022, 10, 3);
DateTime dateFinAbonnement = new DateTime(2022, 11, 3);
```

```
DateTime dateParution = new DateTime(2022, 10, 4);

 bool resultatAttendu = true;

bool resultatActuel = ParutionDansAbonnement(dateCommande, dateFinAbonnement,
dateParution);

Assert.AreEqual(resultatAttendu, resultatActuel);

 }

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

 return (DateTime.Compare(dateCommande, dateParution) < 0 &&
DateTime.Compare(dateParution, dateFinAbonnement) < 0); } }
```

Une fenêtre d'alerte doit s'afficher à l'ouverture de l'application, lorsque les abonnements se terminent dans moins de 30 jours, et présenter la liste de ces abonnements.

Je crée une nouvelle "Form" dans le dossier "view", que j'appélle "FrmAlerteFinAbonnement", et je réalise le design de l'interface.

J'opte pour une fenêtre toute simple, contenant un titre qui va stipuler que les abonnements suivants vont arriver à expiration d'ici 30 jours, une datagrid qui va afficher la liste des abonnements arrivants à échéance, et un bouton "ok" pour fermer l'alerte.

Pour récupérer la liste des abonnements qui se terminent d'ici 30 jours, je dois créer une nouvelle fonction dans l'api Rest PHP, que j'appelle "selectAllAbonnementsEcheance();" et dans laquelle j'affecte la requête.

La requête va commencer par sélectionner la date de fin de l'abonnement, l'id de la revue et le titre du document.

Je fais ensuite une jointure entre la table revue et la table abonnement grâce à l'id de la revue, puis une seconde jointure entre la table document et la table revue pour pouvoir ainsi récupérer le titre de la revue.

Enfin, je réalise le calcul de la différence entre les dates entre la date de fin d'abonnement et la date d'aujourd'hui, avec la requête "WHERE DATEDIFF(CURRENT_DATE(), a.dateFinAbonnement) < 30 ", puis je trie les résultats par ordre croissant pour que l'abonnement arrivant le plus rapidement à échéance apparaisse en premier.

Méthode - selectAllAbonnementsEcheance()

```
public function selectAllAbonnementsEcheance(){
$req ="SELECT a.dateFinAbonnement, a.idRevue, d.titre ";
$req .="FROM abonnement a ";
$req .="JOIN revue r ON a.idRevue = r.id ";
$req .="JOIN document d ON r.id = d.id ";
$req .="WHERE DATEDIFF(CURRENT_DATE(), a.dateFinAbonnement) < 30 ";
$req .="ORDER BY a.dateFinAbonnement ASC; ";
return
$this->conn->queryAll($req);
}
```

Enfin, j'appelle cette fonction dans un nouveau "case", dans la fonction "selectAll($table)", auquel j'affecte le endpoint "abonnementsecheance".

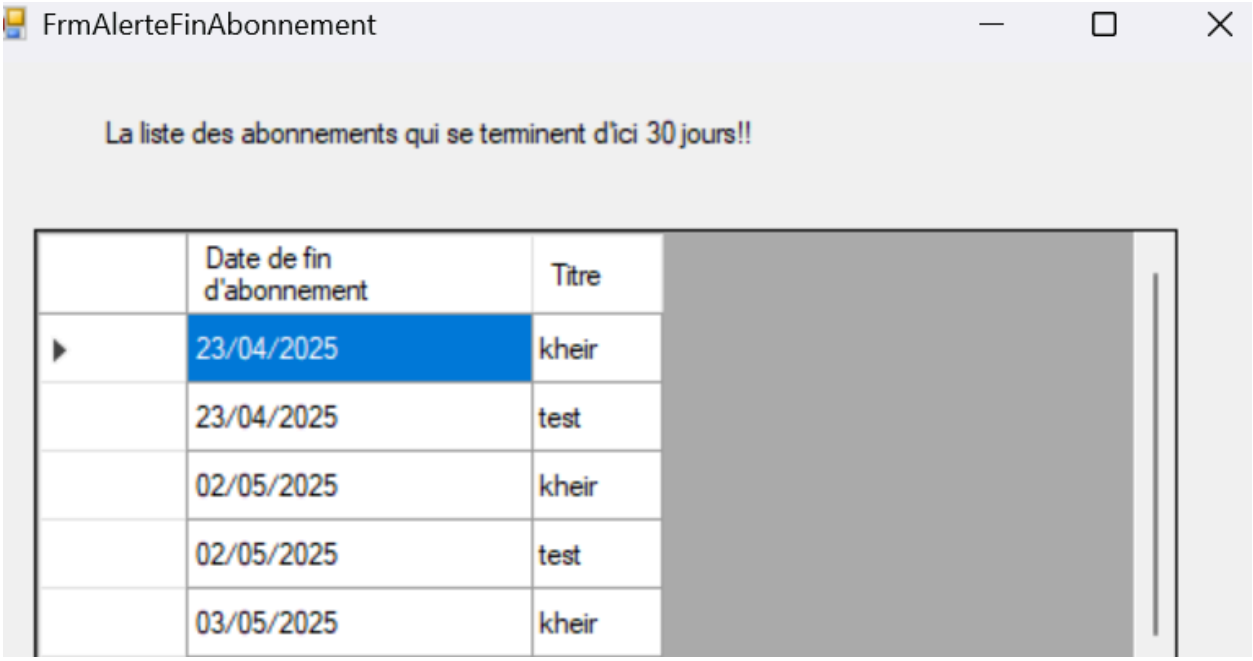
Je me rends ensuite dans la classe "Access" de l'application C#, puis je crée le getter "GetAbonnementsEcheance()" qui va récupérer la liste des abonnements arrivant à échéance sur le endpoint "abonnementsecheance".

Je l'ajoute également au frmMediatekController, puis pour que la "FrmMediatekDocument" prenne en compte l'apparition de la "FrmAlerteFinAbonnement", j'ajoute la fenêtre dans le constructeur de la vue "FrmMediatekDocument", puis je lui précise ".ShowDialog()" pour réaliser l'affichage de cette fenêtre.

De retour dans le code de la vue "AlerteFinAbonnement", je crée la méthode "RemplirAbonnementsAEcheance(List lesAbonnementsAEcheance)" qui va permettre de remplir la grille avec la liste des abonnements arrivant à leur terme.

J'affiche uniquement les colonnes contenant le titre et la date de fin d'abonnement Je réalise ensuite la méthode "dgvAbonnementsAEcheance_ColumnHeaderMouseClick" qui va permettre d'effectuer le tri sur les colonnes .

Puis je crée une petite procédure évènement "" sur le clic du bouton "ok", qui va fermer la fenêtre d'alerte.



MISSION 3. GÉRER L'ÉTAT DES EXEMPLAIRES

Agrandir la fenêtre en hauteur.

Dans l'onglet Livres, partie basse, ajouter la liste des exemplaires du livre sélectionné.

Cette liste d'exemplaires doit contenir les colonnes suivantes : numéro d'exemplaire, date achat et libellé de l'état.

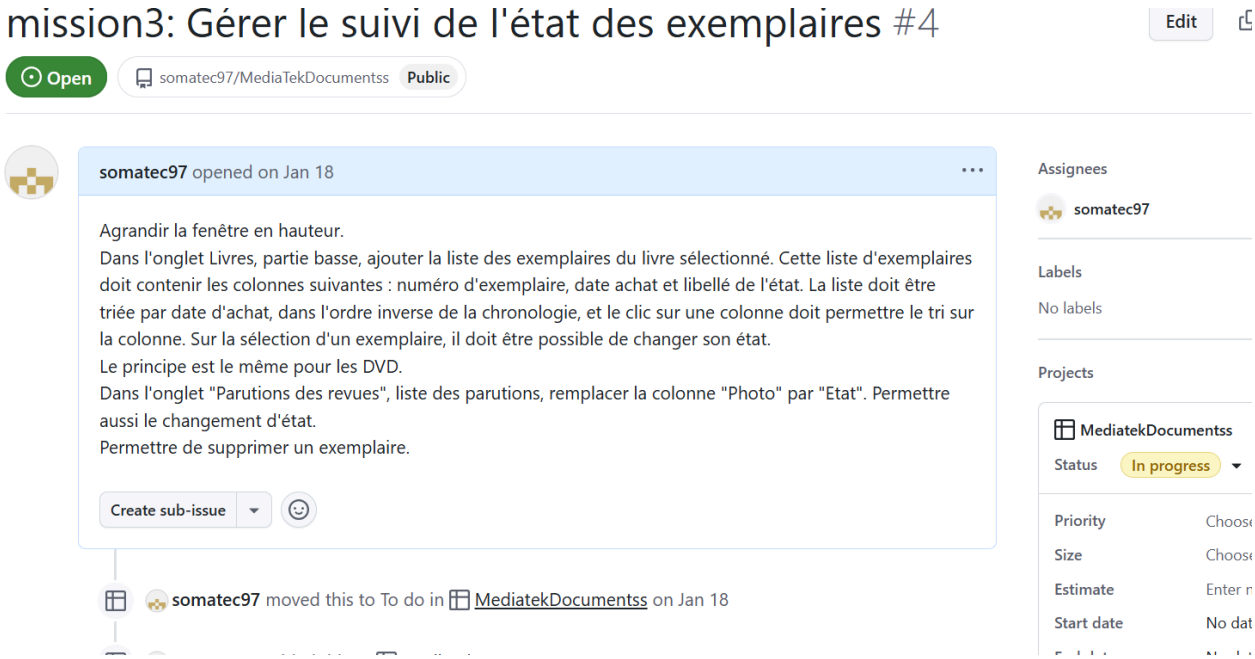
La liste doit être triée par date d'achat, dans l'ordre inverse de la chronologie, et le clic sur une colonne doit permettre le tri sur la colonne.

Sur la sélection d'un exemplaire, il doit être possible de changer son état.

Le principe est le même pour les DVD.

Dans l'onglet "Parutions des revues", liste des parutions, remplacer la colonne "Photo" par "Etat". Permettre aussi le changement d'État. Permettre de supprimer un exemplaire.

Temps estimé 5h - Temps mis environ 7h



ETAT EXEMPLAIRES - ONGLET LIVRES

Je commence par ajouter une groupbox dans l'interface de l'onglet livres, qui va contenir la grille de la liste des exemplaires du livre sélectionné.

J'insère une seconde groupbox qui va permettre d'afficher les informations de l'exemplaire, et de gérer le suivi de l'état de l'exemplaire, j'aoute donc le numéro d'exemplaire, la date d'achat, et un libelle qui va afficher l'état actuel de l'exemplaire, puis une combobox dans laquelle l'utilisateur pourra sélectionner le nouvel état.

J'y insère également un bouton "Modifier l'état de l'exemplaire", pour effectuer la modification une fois la sélection de l'état réalisée, et un bouton "Supprimer l'exemplaire" pour réaliser la suppression d'un exemplaire.

—----------onglet photo livre expl!!!!!!!!!!!!!!!!!!!!!!

Une fois les modifications de l'interface terminées, je vais dans le code de l'interface region "livre", et j'ajoute la méthode "RemplirExemplairesLivre(List lesExemplaires)" qui va réaliser le remplissage de la grille des exemplaires.

J'y affecte les différentes propriétés de la classe "Exemplaire" du dossier "Model", dans chacune des colonnes, et je masque les colonnes qui ne sont pas nécessaires. Il n'y a donc que les colonnes "Numéro", "Date d'achat" et "Etat" qui seront visibles pour l'utilisateur.

Méthode - RemplirExemplairesLivre

```
private void RemplirExemplairesLivre(List lesExemplaires) {

if (lesExemplaires != null) {

bdgExemplairesLivre.DataSource = lesExemplaires;

dgvExemplairesLivre.DataSource = bdgExemplairesLivre;

dgvExemplairesLivre.Columns["photo"].Visible = false;

dgvExemplairesLivre.Columns["idEtat"].Visible = false;
```

```
dgvExemplairesLivre.Columns["id"].Visible = false;

dgvExemplairesLivre.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;

dgvExemplairesLivre.Columns[0].HeaderCell.Value = "Numéro";

dgvExemplairesLivre.Columns[2].HeaderCell.Value = "Date d'achat";

dgvExemplairesLivre.Columns[5].HeaderCell.Value = "Etat";

 } else {

 dgvExemplairesLivre.DataSource = null; }

 }
```

Pour récupérer les données concernant les exemplaires d'un livre, je dois me rendre dans l'api Rest PHP, puis construire une nouvelle fonction contenant les requêtes qui vont permettre de sélectionner la liste des documents d'un livre selon son id.

J'appelle cette fonction "selectAllExemplairesDocument($id)", et lui affecte le paramètre "id". Cette requête va sélectionner les informations contenues dans les champs de la table "exemplaire", et dans le champ "libelle" de la table "etat".

Je réalise une jointure entre la table exemplaire et la table etat, permettant ainsi de récupérer l'id de l'état de l'exemplaire, enfin, je trie la réception de la sélection par ordre décroissant sur la date d'achat.

J'appelle cette fonction dans un nouveau case de la méthode "select One", o* les données seront récupérées sur le endpoint "exemplairesdocument".

Méthode - selectAllExemplairesDocument($id)

```
 public function selectAllExemplairesDocument($id){

$param = array( "id" => $id );
```

```
$req = "SELECT ex.id, ex.numero, ex.dateAchat, ex.photo, ex.idEtat, et.libelle ";

$req .= "FROM exemplaire ex JOIN etat et ON ex.idEtat = et.id ";

$req .= "WHERE ex.id = :id "; $req .= "ORDER BY ex.dateAchat DESC";

return $this->conn->query($req, $param);

}
```

Je retourne dans l'application C#, et dans la classe "Access", j'insère le getter "GetExemplairesDocument(string idDocument)", qui va récupéré les donnés avec la méthode http GET, sur le endpoint "exemplairesdocument/idDocument", où l'idDocument sera remplacé par celui du document sélectionné.

J'appelle cette méthode dans le controller, puis je créé la méthode "AfficheExemplairesLivres()" dans la vue de l'onglet "livre".

J'affecte à la variable "idDocument", la valeur contenue dans le champ "txbLivresNumRecherche.Test", puis la liste des exemplaires contenus dans le document sélectionné sera affectée dans la variable "lesExemplairesDocument".

J'appelle la méthode "RemplirExemplairesLivre" dans laquelle j'intègre la variable, permettant ainsi d'afficher la liste des exemplaires concernant le document. La méthode "AfficheExemplairesLivres()" est ensuite insérée dans la procédure événementielle du clic sur le bouton "rechercher", pour que la liste s'affiche dès que l'utilisateur a recherché un document.

J'ajoute ensuite la procédure évènementielle "dgvExemplairesLivre_ColumnHeaderMouseClick", qui va permettre à l'utilisateur de trier les colonnes, et de trier la date d'achat dans l'ordre inverse de la chronologie.

Procédure évènementielle - dgvExemplairesLivres_ColumnHeaderMouseClick

```
private void dgvExemplairesLivres_ColumnHeaderMouseClick(object sender,
DataGridViewCellMouseEventArgs e) {
```

```
string titreColonne = dgvExemplairesLivre.Columns[e.ColumnIndex].HeaderText;

 List sortedList = new List();

switch (titreColonne) {

 case "Date d'achat": sortedList = lesExemplairesDocument.OrderBy(o =>
o.DateAchat).Reverse().ToList();

break;

case "Numéro": sortedList = lesExemplairesDocument.OrderBy(o => o.Numero).ToList();

 break;

 case "Etat": sortedList = lesExemplairesDocument.OrderBy(o => o.Libelle).ToList();

break;

 }

 RemplirExemplairesLivre(sortedList);

}
```

Pour remplir la combobox de suivi de l'état d'un exemplaire, je crée les mêmes méthodes que lors du suivi de l'étape d'une commande.

Je commence par faire la méthode "RemplirCbxEtatLibelleExemplaireLivre(string etatExemplaireLivre), qui va remplir la combobox selon les états de l'exemplaire possible et le libelle correspondant.

Je crée la condition "if" dans laquelle j'affecte le libelle correspondant à un des états d'exemplaire, et dans laquelle j'ajoute les libelle des états possibles.

Méthode - RemplirCbxEtatLibelleExemplaireLivre(string etatExemplaireLivre)

```
 private void RemplirCbxEtatLibelleExemplaireLivre(string etatExemplaireLivre) {
```

```
cbxEtatLibelleExemplaireLivre.Items.Clear();
if (etatExemplaireLivre == "neuf") {
cbxEtatLibelleExemplaireLivre.Items.Add("usagé");
cbxEtatLibelleExemplaireLivre.Items.Add("détérioré");
cbxEtatLibelleExemplaireLivre.Items.Add("inutilisable");
 } else if (etatExemplaireLivre == "usagé") {
cbxEtatLibelleExemplaireLivre.Text = "";
 cbxEtatLibelleExemplaireLivre.Items.Add("neuf");
cbxEtatLibelleExemplaireLivre.Items.Add("détérioré");
cbxEtatLibelleExemplaireLivre.Items.Add("inutilisable");
 } else if (etatExemplaireLivre == "détérioré") {
cbxEtatLibelleExemplaireLivre.Text = "";
 cbxEtatLibelleExemplaireLivre.Items.Add("neuf");
cbxEtatLibelleExemplaireLivre.Items.Add("usagé");
cbxEtatLibelleExemplaireLivre.Items.Add("inutilisable");
} else if (etatExemplaireLivre == "inutilisable") {
 cbxEtatLibelleExemplaireLivre.Text = "";
 cbxEtatLibelleExemplaireLivre.Items.Add("neuf");
cbxEtatLibelleExemplaireLivre.Items.Add("usagé");
cbxEtatLibelleExemplaireLivre.Items.Add("détérioré");
```

}

J'appelle cette méthode dans la procédure évènementielle " lblEtatExemplaireLivre_TextChanged", qui va affecter à la variable "etatExemplaireLivre", la valeur contenue dans le label "EtatExemplaireLivre.Text".

Selon l'état contenu dans cette variable, le remplissage de la combobox se fera selon les conditions imposées précédemment.

Je dois ensuite créer la méthode "GetIdEtat(string libelle)" pour récupérer le l'id de l'état d'un exemplaire selon son libelle.

Je créé donc un getter "GetAllEtatsDocument" dans la classe Access, que je récupère dans le controller, et que j'affecte à l'objet Etat dans la variable "lesEtats".

J'ajoute une boucle qui va retourner l'id correspondant à chaque libelle d'état.

Méthode - GetIdEtat(string libelle)

```
private string GetIdEtat(string libelle) {

 List lesEtats = controller.GetAllEtatsDocument();

foreach (Etat unEtat in lesEtats) {

 if (unEtat.Libelle == libelle) {

 return unEtat.Id;

 } } return null;

}
```

Lorsque l'utilisateur sélectionne un exemplaire dans la grille, les informations sur son état et les modifications d'état possibles doivent s'afficher dans la groupbox des informations d'exemplaire, et des informations sur l'état de l'exemplaire. Pour cela, je créé la méthode "dgvExemplairesLivre_RowEnter", dans laquelle j'affecte la valeur contenu dans chaque cellule

de la colonne sélectionnée, dans les champs correspondants, et j'active la groupbox qui permet de sélectionner un nouvel état puis de le modifier.

Méthode - dgvExemplairesLivre_RowEnter

```
 private void dgvExemplairesLivre_RowEnter(object sender, DataGridViewCellEventArgs e) {

DataGridViewRow row = dgvExemplairesLivre.Rows[e.RowIndex];

string id = row.Cells["Id"].Value.ToString();

 string numero = row.Cells["Numero"].Value.ToString();

 DateTime dateAchat = (DateTime)row.Cells["dateAchat"].Value;

 string libelle = row.Cells["Libelle"].Value.ToString();

txbExemplaireLivresNumero.Text = numero;

dtpDateAchatExemplaireLivre.Value = dateAchat;

lblEtatExemplaireLivre.Text = libelle;

gbxEtatExemplaireLivre.Enabled = true;

}
```

Je peux maintenant construire la fonctionnalité "Modifier l'état d'un exemplaire sélectionné", pour cela, je me rends dans l'api Rest PHP, pour y modifier la méthode "updateOne($table, $id, $champs)".

Je modifie cette méthode car ce que je souhaite obtenir c'est la modification d'un numéro d'exemplaire spécifique à un document, la méthode précédente ne me permettait pas de modifier l'état d'un numéro, à la place elle modifiait l'état de tous les numéros concernant un document.

Dans cette méthode, j'ajoute donc un case "exemplairedocument", pour définir la table et les champs "exemplaire", et réaliser un "where id et numero" , afin de récupérer l'id du document ainsi que le numéro d'exemplaire. Le case default permet de modifier les tables globales.

Fonction - updateOne

```
public function updateOne($table, $id, $champs){

 if($this->conn != null && $champs != null){

switch($table){

 case "exemplairesdocument":

 $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=:$key,"; }

$requete = substr($requete, 0, strlen($requete)-1);

$requete .= " WHERE id=:id AND numero=:numero;"; $champsExemplaire['numero'] = $id;

 $updateExemplaire = $this->conn->execute($requete, $champsExemplaire);

if(!$updateExemplaire){ return null; } default: $champs['id'] = $id; $requete = "UPDATE $table SET
";

 foreach ($champs as $key => $value) { $requete .= "$key=:$key,"; }

$requete = substr($requete, 0, strlen($requete)-1);

 $requete .= " WHERE id=:id;"; return $this->conn->execute($requete, $champs); }

 } else {
```

```
 return null;
}
```

Je retourne dans la classe "Access" de l'application C#, et je construis la méthode "ModifierEtatExemplaireDocument(Exemplaire exemplaire)" qui va permettre de modifier l'état de l'exemplaire en base de données, grâce à la méthode http PUT, où la chaîne de données concernant la table "exemplaire", au format json, sera ajoutée au endpoint "exemplairesdocument/", au lieu de préciser l'id du document, je précise le numéro de l'exemplaire, pour obtenir la modification sur un exemplaire unique.

J'appelle cette méthode dans le "FrmMediatekController", puis je créé la procédure évènementielle "btnEtatExemplaireLivreModifier_Click" qui va se déclencher lors du clic sur le bouton "Modifier l'état de l'exemplaire".

J'affecte les valeurs des champs dans des variables respectives, puis un nouvel objet "Exemplaire" dans la variable "exemplaire".

J'appelle ensuite la méthode du controller "ModifierEtatExemplaireDocument(exemplaire)" qui va effectuer la modification selon les nouvelles valeurs contenues dans l'objet exemplaire.

Procédure évènementielle - btnEtatExemplaireLivreModifier_Click

```
private void btnEtatExemplaireLivreModifier_Click(object sender, EventArgs e) {
string idDocument = txbLivresNumRecherche.Text;
int numero = int.Parse(txbExemplaireLivresNumero.Text);
 DateTime dateAchat = dtpDateAchatExemplaireLivre.Value;
 string photo = ""; string idEtat = GetIdEtat(cbxEtatLibelleExemplaireLivre.Text);
string libelle = cbxEtatLibelleExemplaireLivre.SelectedItem.ToString();
Exemplaire exemplaire = new Exemplaire(numero, dateAchat, photo, idEtat, idDocument, libelle);
```

```
if (MessageBox.Show("Voulez-vous modifier l'état de l'exemplaire " + exemplaire.Numero + "
en " + libelle + " ?", "Confirmation", MessageBoxButtons.YesNo) == DialogResult.Yes) {

controller.ModifierEtatExemplaireDocument(exemplaire);

MessageBox.Show("L'état de l'exemplaire " + exemplaire.Numero + " a bien été modifié.",
"Information");

 AfficheExemplairesLivres();

} }
```

Pour réaliser la suppression d'un exemplaire de document dans la base de données, je créé la méthode "SupprimerExemplaireDocument(Exemplaire exemplaire)" dans la classe "Access".

J'y affecte la chaîne au format json, permettant de supprimer un exemplaire de document grâce à l'id de l'exemplaire (correspondant à l'id du document), et le numéro de l'exemplaire.

J'effectue ensuite la requête http "DELETE", dans laquelle je précise le endpoint "exemplaire/" auquel j'ajoute la chaîne json.

Méthode - SupprimerExemplaireDocument(Exemplaire exemplaire)

```
 public bool SupprimerExemplaireDocument(Exemplaire exemplaire) {

String jsonSupprimerExemplaireDocument = "{\"id\":\"" + exemplaire.Id + "\",\"numero\":\"" +
exemplaire.Numero + "\"}";

Console.WriteLine("jsonSupprimerExemplaireDocument" + jsonSupprimerExemplaireDocument);

try {

 // récupération soit d'une liste vide (requête ok) soit de null (erreur)

List liste = TraitementRecup(DELETE, "exemplaire/" + jsonSupprimerExemplaireDocument);
return (liste != null);
```

```
} catch (Exception ex) {

Console.WriteLine(ex.Message)

; } return false;

 }
```

J'appelle cette méthode dans le controller "FrmMediatekController", puis je crée la procédure évènementielle "btnExemplaireLivreSupprimer_Click", qui va permettre à l'utilisateur de réaliser la suppression de l'exemplaire du livre sélectionné.

Je créé une condition "if" qui oblige l'utilisateur à sélectionner une ligne dans la grille des exemplaires, puis j'affecte à la variable "exemplaire", l'objet "Exemplaire" sélectionné dans la liste.

J'appelle la méthode "SupprimerExemplaireDocument(exemplaire)" du controller, pour enregistrer la suppression de l'exemplaire dans la base de données, et j'affiche la liste des exemplaires du livre mise à jour après la suppression.

Procédure évènementielle - btnExemplaireLivreSupprimer_Click

```
private void btnExemplaireLivreSupprimer_Click(object sender, EventArgs e) { if (dgvExemplairesLivre.SelectedRows.Count > 0) { Exemplaire exemplaire = (Exemplaire)bdgExemplairesLivre.List[bdgExemplairesLivre.Position]; if (MessageBox.Show("Voulez-vous supprimer l'exemplaire " + exemplaire.Numero + " du livre " + exemplaire.Id + " ?", "Confirmation de suppression", MessageBoxButtons.YesNo) == DialogResult.Yes) { controller.SupprimerExemplaireDocument(exemplaire); MessageBox.Show("L'exemplaire " + exemplaire.Numero + " a bien été supprimé.", "Information"); AfficheExemplairesLivres(); } } else { MessageBox.Show("Une ligne doit être sélectionnée.", "Information"); } }
```

ETAT EXEMPLAIRES - ONGLET DVD

Je commence par ajouter une groupbox dans l'interface de l'onglet dvd, qui va contenir la grille de la liste des exemplaires du dvd sélectionné.

J'insère une seconde groupbox qui va permettre d'afficher les informations de l'exemplaire, et de gérer le suivi de l'état de l'exemplaire, j'aoute donc le numéro d'exemplaire, la date d'achat, et un libelle qui va afficher l'état actuel de l'exemplaire, puis une combobox dans laquelle l'utilisateur pourra sélectionner le nouvel état.

J'y insère également un bouton "Modifier l'état de l'exemplaire", pour effectuer la modification une fois la sélection de l'état réalisée, et un bouton "Supprimer l'exemplaire" pour réaliser la suppression d'un exemplaire.

_____ongle dvd exempl—----------

Une fois les modifications de l'interface terminées, je vais dans le code de l'interface région "dvd", et j'ajoute la méthode "RemplirExemplairesDvd(List lesExemplaires)" qui va réaliser le remplissage de la grille des exemplaires.

J'y affecte les différentes propriétés de la classe "Dvd" du dossier "Model", dans chacune des colonnes, et je masque les colonnes qui ne sont pas nécessaires. Il n'y a donc que les colonnes "Numéro", "Date d'achat" et "Etat" qui seront visibles pour l'utilisateur.

Méthode - RemplirExemplairesDvd(List lesExemplaires)

```
private void RemplirExemplairesDvd(List lesExemplaires) { if (lesExemplaires != null) { bdgExemplairesDvd.DataSource = lesExemplaires; dgvExemplairesDvd.DataSource = bdgExemplairesDvd; dgvExemplairesDvd.Columns["photo"].Visible = false; dgvExemplairesDvd.Columns["idEtat"].Visible = false; dgvExemplairesDvd.Columns["id"].Visible = false; dgvExemplairesDvd.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells; dgvExemplairesDvd.Columns[0].HeaderCell.Value = "Numéro"; dgvExemplairesDvd.Columns[2].HeaderCell.Value = "Date d'achat"; dgvExemplairesDvd.Columns[5].HeaderCell.Value = "Etat"; } else { dgvExemplairesDvd.DataSource = null; } }
```

Ensuite, je créé la méthode "AfficheExemplairesDvd()" dans la vue de l'onglet "dvd".

J'affecte à la variable "idDocument", la valeur contenue dans le champ "txbDvdNumRecherche.Text", puis la liste des exemplaires contenu dans le document sélectionné sera affectée dans la variable "lesExemplairesDocument".

J'appelle la méthode "RemplirExemplairesDvd" dans laquelle j'intègre la variable, permettant ainsi d'afficher la liste des exemplaires concernant le document.

La méthode "AfficheExemplairesDvd()" est ensuite insérée dans la procédure évènementielle du clic sur le bouton "rechercher", pour que la liste s'affiche dès que l'utilisateur a recherché un document. J'ajoute ensuite la procédure évènementielle "dgvExemplairesDvd_ColumnHeaderMouseClick", qui va permettre à l'utilisateur de trier les colonnes, et de trier la date d'achat dans l'ordre inverse de la chronologie.

Procédure évènementielle - dgvExemplairesLivres_ColumnHeaderMouseClick

```
private void dgvExemplairesDvd_ColumnHeaderMouseClick(object sender,
DataGridViewCellMouseEventArgs e) { string titreColonne =
dgvExemplairesDvd.Columns[e.ColumnIndex].HeaderText; List sortedList = new List(); switch
(titreColonne) { case "Date d'achat": sortedList = lesExemplairesDocument.OrderBy(o =>
o.DateAchat).Reverse().ToList(); break; case "Numéro": sortedList =
lesExemplairesDocument.OrderBy(o => o.Numero).ToList(); break; case "Etat": sortedList =
lesExemplairesDocument.OrderBy(o => o.Libelle).ToList(); break; }

 RemplirExemplairesDvd(sortedList); }
```

Pour remplir la combobox de suivi de l'état d'un exemplaire, je créé les mêmes méthodes que lors du suivi de l'état d'un livre.

Je commence par faire la méthode "RemplirCbxEtatLibelleExemplaireDvd(string etatExemplaireDvd), qui va remplir la combobox selon les états de l'exemplaire possible et le libelle correspondant. Je créé la condition "if" dans laquelle j'affecte le libelle correspondant à un des états d'exemplaire, et dans laquelle j'ajoute les libelle des états possibles.

Méthode - RemplirCbxEtatLibelleExemplaireDvd(string etatExemplaireDvd)

```
private void RemplirCbxEtatLibelleExemplaireDvd(string etatExemplaireDvd) {
cbxEtatLibelleExemplaireDvd.Items.Clear(); if (etatExemplaireDvd == "neuf") {
cbxEtatLibelleExemplaireDvd.Items.Add("usagé");
cbxEtatLibelleExemplaireDvd.Items.Add("détérioré");
cbxEtatLibelleExemplaireDvd.Items.Add("inutilisable"); } else if (etatExemplaireDvd == "usagé") {
cbxEtatLibelleExemplaireDvd.Text = ""; cbxEtatLibelleExemplaireDvd.Items.Add("neuf");
cbxEtatLibelleExemplaireDvd.Items.Add("détérioré");
cbxEtatLibelleExemplaireDvd.Items.Add("inutilisable"); } else if (etatExemplaireDvd ==
"détérioré") { cbxEtatLibelleExemplaireDvd.Text = "";
cbxEtatLibelleExemplaireDvd.Items.Add("neuf");
cbxEtatLibelleExemplaireDvd.Items.Add("usagé");
cbxEtatLibelleExemplaireDvd.Items.Add("inutilisable"); } else if (etatExemplaireDvd ==
"inutilisable") { cbxEtatLibelleExemplaireDvd.Text = "";
cbxEtatLibelleExemplaireDvd.Items.Add("neuf");
cbxEtatLibelleExemplaireDvd.Items.Add("usagé");
cbxEtatLibelleExemplaireDvd.Items.Add("détérioré"); } }
```

J'appelle cette méthode dans la procédure évènementielle " lblEtatExemplaireDvd_TextChanged", qui va affecter à la variable "etatExemplaireDvd", la valeur contenue dans le label "EtatExemplaireDvd.Text".

Selon l'état contenu dans cette variable, le remplissage de la combobox se fera selon les conditions imposées précédemment.

Lorsque l'utilisateur sélectionne un exemplaire dans la grille, les informations sur son état et les modifications d'état possibles doivent s'afficher dans la groupbox des informations d'exemplaire. Pour cela, je créé la méthode "dgvExemplairesDvd_RowEnter", dans laquelle j'affecte la valeur contenu dans chaque cellule de la colonne sélectionnée, dans les champs correspondants, et j'active la groupbox qui permet de sélectionner un nouvel état puis de le modifier.

Méthode - dgvExemplairesDvd_RowEnter

```
private void dgvExemplairesDvd_RowEnter(object sender, DataGridViewCellEventArgs e) {
DataGridViewRow row = dgvExemplairesDvd.Rows[e.RowIndex]; string id =
row.Cells["Id"].Value.ToString(); string numero = row.Cells["Numero"].Value.ToString(); DateTime
dateAchat = (DateTime)row.Cells["dateAchat"].Value; string libelle =
row.Cells["Libelle"].Value.ToString(); txbExemplaireDvdNumero.Text = numero;
dtpDateAchatExemplaireDvd.Value = dateAchat; lblEtatExemplaireDvd.Text = libelle;
gbxEtatExemplaireDvd.Enabled = true; }
```

Je procède maintenant à la création de la fonctionnalité "Modifier l'état d'un exemplaire", en construisant la procédure évènementielle "btnEtatExemplaireDvdModifier_Click" qui va se déclencher lors du clic sur le bouton "Modifier l'état de l'exemplaire".

J'affecte les valeurs des champs dans des variables respectives, puis un nouvel objet "Exemplaire" dans la variable "exemplaire".

J'appelle ensuite la méthode du controller "ModifierEtatExemplaireDocument(exemplaire)" qui va effectuer la modification selon les nouvelles valeurs contenues dans l'objet exemplaire.

Procédure évènementielle - btnEtatExemplaireDvdModifier_Click

```
private void btnEtatExemplaireDvdModifier_Click(object sender, EventArgs e) { string
idDocument = txbDvdNumRecherche.Text; int numero =
int.Parse(txbExemplaireDvdNumero.Text); DateTime dateAchat =
dtpDateAchatExemplaireDvd.Value; string photo = ""; string idEtat =
GetIdEtat(cbxEtatLibelleExemplaireDvd.Text); try { string libelle =
cbxEtatLibelleExemplaireDvd.SelectedItem.ToString(); Exemplaire exemplaire = new
Exemplaire(numero, dateAchat, photo, idEtat, idDocument, libelle); if
(MessageBox.Show("Voulez-vous modifier l'état de l'exemplaire " + exemplaire.Numero + " en " +
libelle + " ?", "Confirmation", MessageBoxButtons.YesNo) == DialogResult.Yes) {
controller.ModifierEtatExemplaireDocument(exemplaire); MessageBox.Show("L'état de
l'exemplaire " + exemplaire.Numero + " a bien été modifié.", "Information");
```

AfficheExemplairesDvd(); } } catch (NullReferenceException) { MessageBox.Show("Le nouvel état de l'exemplaire doit être sélectionné.", "Information"); } }

Pour réaliser la suppression d'un exemplaire de dvd, je créé la procédure évènementielle "btnExemplaireDvdSupprimer_Click", qui va permettre à l'utilisateur de réaliser la suppression de l'exemplaire du dvd sélectionné.

Je créé une condition "if" qui oblige l'utilisateur à sélectionner une ligne dans la grille des exemplaires, puis j'affecte à la variable "exemplaire", l'objet "Exemplaire" sélectionné dans la liste.

J'appelle la méthode "SupprimerExemplaireDocument(exemplaire)" du controller, pour enregistrer la suppression de l'exemplaire dans la base de données, et j'affiche la liste des exemplaires du dvd mise à jour après la suppression.

Procédure évènementielle - btnExemplaireDvdSupprimer_Click

private void btnExemplaireDvdSupprimer_Click(object sender, EventArgs e) { if (dgvExemplairesDvd.SelectedRows.Count > 0) { Exemplaire exemplaire = (Exemplaire)bdgExemplairesDvd.List[bdgExemplairesDvd.Position]; if (MessageBox.Show("Voulez-vous supprimer l'exemplaire " + exemplaire.Numero + " du dvd " + exemplaire.Id + " ?", "Confirmation de suppression", MessageBoxButtons.YesNo) == DialogResult.Yes) { controller.SupprimerExemplaireDocument(exemplaire); MessageBox.Show("L'exemplaire " + exemplaire.Numero + " a bien été supprimé.", "Information"); AfficheExemplairesDvd(); } } else { MessageBox.Show("Une ligne doit être sélectionnée.", "Information"); } }

ETAT EXEMPLAIRES - ONGLET PARUTION REVUES

Je commence par ajouter une groupbox dans l'interface de l'onglet parution, qui va permettre d'afficher les informations de l'exemplaire, et de gérer le suivi de l'état de l'exemplaire, j'ajoute donc le numéro d'exemplaire, la date d'achat, et un libelle qui va afficher l'état actuel de l'exemplaire, puis une combobox dans laquelle l'utilisateur pourra sélectionner le nouvel état.

J'y insère également un bouton "Modifier l'état de l'exemplaire", pour effectuer la modification une fois la sélection de l'état réalisée, et un bouton "Supprimer l'exemplaire" pour réaliser la suppression d'un exemplaire.

_____photo onglet revues exmpl—----------

Une fois les modifications de l'interface terminée, je vais dans le code de la classe "Access" et je créé une nouvelle méthode "CreerExemplaireRevue(Exemplaire exemplaire)" car la méthode initiale "CreerExemplaire" ne fonctionne plus lorsque je souhaite ajouter un exemplaire à une revue, je n'ai pas réussi à résoudre ce problème, d'où la nouvelle méthode .

Je modifie le code de la vue dans l'onglet "parution" en conséquence.

Méthode - CreerExemplaireRevue(Exemplaire exemplaire)

public bool CreerExemplaireRevue(string id, int numero, DateTime dateAchat, string photo, string idEtat) { String jsonDateAchat = JsonConvert.SerializeObject(dateAchat, new CustomDateTimeConverter()); String jsonCreerExemplaireRevue = "{\"id\":\"" + id + "\", \"numero\":\"" + numero + "\", \"dateAchat\" : " + jsonDateAchat + ", \"photo\" : \"" + photo + "\" , \"idEtat\" : \"" + idEtat+ "\"}"; try { List liste = TraitementRecup(POST, "exemplaire/" + jsonCreerExemplaireRevue); return (liste != null); }

Je modifie ensuite la colonne "Photo" en "Etat", pour qu'elle affiche l'état de l'exemplaire de la revue sélectionnée, dans la méthode "RemplirReceptionExemplairesListe(List exemplaires)" , et je passe la colonne "photo" en "false" pour qu'elle soit masquée.

Dans la procédure évènementielle "dgvExemplairesListe_ColumnHeaderMouseClick", je modifie le case "Photo" en "Etat", et lui affecte "Libelle" pour obtenir le libelle de l'état des exemplaires.

Pour remplir la combobox de suivi de l'état d'un exemplaire, je créé les mêmes méthodes que lors du suivi de l'état d'un livre et d'un dvd.

Je commence par faire la méthode "RemplirCbxEtatLibelleExemplaireRevue(string etatExemplaireRevue), qui va remplir la combobox selon les états de l'exemplaire possible et le libelle correspondant.

Je créé la condition "if" dans laquelle j'affecte le libelle correspondant à un des états d'exemplaire, et dans laquelle j'ajoute les libelle des états possibles.

Méthode - RemplirCbxEtatLibelleExemplaireDvd(string etatExemplaireRevue)

```
private void RemplirCbxEtatLibelleExemplaireRevue(string etatExemplaireRevue) {
cbxEtatLibelleExemplaireRevue.Items.Clear(); if (etatExemplaireRevue == "neuf") {
cbxEtatLibelleExemplaireRevue.Items.Add("usagé");
cbxEtatLibelleExemplaireRevue.Items.Add("détérioré");
cbxEtatLibelleExemplaireRevue.Items.Add("inutilisable"); } else if (etatExemplaireRevue ==
"usagé") { cbxEtatLibelleExemplaireRevue.Text = "";
cbxEtatLibelleExemplaireRevue.Items.Add("neuf");
cbxEtatLibelleExemplaireRevue.Items.Add("détérioré");
cbxEtatLibelleExemplaireRevue.Items.Add("inutilisable"); } else if (etatExemplaireRevue ==
"détérioré") { cbxEtatLibelleExemplaireRevue.Text = "";
cbxEtatLibelleExemplaireRevue.Items.Add("neuf");
cbxEtatLibelleExemplaireRevue.Items.Add("usagé");
cbxEtatLibelleExemplaireRevue.Items.Add("inutilisable"); } else if (etatExemplaireRevue ==
"inutilisable") { cbxEtatLibelleExemplaireRevue.Text = "";
cbxEtatLibelleExemplaireRevue.Items.Add("neuf");
cbxEtatLibelleExemplaireRevue.Items.Add("usagé");
cbxEtatLibelleExemplaireRevue.Items.Add("détérioré"); } }
```

J'appelle cette méthode dans la procédure évènementielle " lblEtatExemplaireRevue_TextChanged", qui va affecter à la variable "etatExemplaireRevue", la valeur contenue dans le label "EtatExemplaireRevue.Text".

Selon l'état contenu dans cette variable, le remplissage de la combobox se fera selon les conditions imposées précédemment.

Lorsque l'utilisateur sélectionne un exemplaire dans la grille, les informations sur son état et les modifications d'état possibles doivent s'afficher dans la groupbox des informations d'exemplaire.

Pour cela, je créé la méthode "dgvReceptionExemplairesListe_RowEnter", dans laquelle j'affecte la valeur contenu dans chaque cellule de la colonne sélectionnée, dans les champs correspondants, et j'active la groupbox qui permet de sélectionner un nouvel état puis de le modifier.

Je modifie ensuite la colonne "Photo" en "Etat", pour qu'elle affiche l'état de l'exemplaire de la revue sélectionnée, dans la méthode "RemplirReceptionExemplairesListe(List exemplaires)" , et je passe la colonne "photo" en "false" pour qu'elle soit masquée. Dans la procédure évènementielle "dgvExemplairesListe_ColumnHeaderMouseClick", je modifie le case "Photo" en "Etat", et lui affecte "Libelle" pour obtenir le libelle de l'état des exemplaires.

Méthode - dgvReceptionExemplairesListe_RowEnter

```
private void dgvReceptionExemplairesListe_RowEnter(object sender, DataGridViewCellEventArgs e) { DataGridViewRow row = dgvReceptionExemplairesListe.Rows[e.RowIndex]; string id = row.Cells["Id"].Value.ToString(); string numero = row.Cells["Numero"].Value.ToString(); DateTime dateAchat = (DateTime)row.Cells["dateAchat"].Value; string libelle = row.Cells["Libelle"].Value.ToString(); gbxEtatExemplaireRevue.Enabled = true; lblNumeroExemplaireRevue.Text = numero; dtpDateAchatExemplaireRevue.Value = dateAchat; lblEtatExemplaireRevue.Text = libelle; }
```

Je procède maintenant à la création de la fonctionnalité "Modifier l'état d'un exemplaire", en construisant la procédure évènementielle "btnEtatExemplaireRevueModifier_Click" qui va se déclencher lors du clic sur le bouton "Modifier l'état de l'exemplaire".

J'affecte les valeurs des champs dans des variables respectives, puis un nouvel objet "Exemplaire" dans la variable "exemplaire".

J'appelle ensuite la méthode du controller "ModifierEtatExemplaireDocument(exemplaire)" qui va effectuer la modification selon les nouvelles valeurs contenues dans l'objet exemplaire.

Procédure évènementielle - btnEtatExemplaireRevueModifier_Click

```
private void btnEtatExemplaireRevueModifier_Click(object sender, EventArgs e) { string idDocument = txbReceptionRevueNumero.Text; int numero = int.Parse(lblNumeroExemplaireRevue.Text); DateTime dateAchat = dtpDateAchatExemplaireRevue.Value; string photo = ""; string idEtat = GetIdEtat(cbxEtatLibelleExemplaireRevue.Text); try { string libelle = cbxEtatLibelleExemplaireRevue.SelectedItem.ToString(); Exemplaire exemplaire = new Exemplaire(numero, dateAchat, photo, idEtat, idDocument, libelle); if (MessageBox.Show("Voulez-vous modifier l'état de l'exemplaire " + exemplaire.Numero + " en " + libelle + " ?", "Confirmation", MessageBoxButtons.YesNo) == DialogResult.Yes) { controller.ModifierEtatExemplaireDocument(exemplaire); MessageBox.Show("L'état de l'exemplaire " + exemplaire.Numero + " a bien été modifié.", "Information"); AfficheReceptionExemplairesRevue(); } } catch (NullReferenceException) { MessageBox.Show("Le nouvel état de l'exemplaire doit être sélectionné.", "Information"); } }
```

Pour réaliser la suppression d'un exemplaire de revue, je créé la procédure évènementielle "btnExemplaireRevueSupprimer_Click", qui va permettre à l'utilisateur de réaliser la suppression de l'exemplaire de la revue sélectionné. Je créé une condition "if" qui oblige l'utilisateur à sélectionner une ligne dans la grille des exemplaires, puis j'affecte à la variable "exemplaire", l'objet "Exemplaire" sélectionné dans la liste.

J'appelle la méthode "SupprimerExemplaireDocument(exemplaire)" du controller, pour enregistrer la suppression de l'exemplaire dans la base de données, et j'affiche la liste des exemplaires de la revue mise à jour après la suppression.

Procédure évènementielle - btnExemplaireRevueSupprimer_Click

```
private void btnExemplaireRevueSupprimer_Click(object sender, EventArgs e) { if (dgvReceptionExemplairesListe.SelectedRows.Count > 0) { Exemplaire exemplaire = (Exemplaire)bdgExemplairesListe.List[bdgExemplairesListe.Position]; if (MessageBox.Show("Voulez-vous supprimer l'exemplaire " + exemplaire.Numero + " de la revue " + exemplaire.Id + " ?", "Confirmation de suppression", MessageBoxButtons.YesNo) ==
```

DialogResult.Yes) { controller.SupprimerExemplaireDocument(exemplaire); MessageBox.Show("L'exemplaire " + exemplaire.Numero + " a bien été supprimé.", "Information"); AfficheReceptionExemplairesRevue(); } } else { MessageBox.Show("Une ligne doit être sélectionnée.", "Information"); } }

MISSION 4. METTRE EN PLACE DES AUTHENTIFICATIONS

Dans la base de données, ajouter une table Utilisateur et une table Service, sachant que chaque utilisateur ne fait partie que d'un service.

Pour réaliser les tests, remplir les tables d'exemples. Ajouter une première fenêtre d'authentification.

Faire en sorte que l'application démarre sur cette fenêtre.

Suivant le type de personne authentifiée, empêcher certains accès en rendant invisibles ou inactifs certains onglets ou objets graphiques.

Dans le cas du service Culture qui n'a accès à rien, afficher un message précisant que les droits ne sont pas suffisants pour accéder à cette application, puis fermer l'application.

Faire en sorte que l'alerte de fin d'abonnement n'apparaisse que pour les personnes concernées (qui gèrent les commandes).

Temps estimé 4h - Temps mis environ 3h

AJOUTER UNE TABLE "UTILISATEUR" - BDD

Il faut commencer par créer une nouvelle table "utilisateur" dans la base de données, qui doit contenir des identifiants d'authentification d'utilisateurs, appartenant à différents services.

Je me rends donc dans le phpmyadmin du serveur d'hébergement de ma base de données, et j'ajoute la commande mysql suivante pour créer la nouvelle table : CREATE TABLE `utilisateur` (`id` varchar(5) NOT NULL, `login` varchar(50) DEFAULT NULL, `password` varchar(50) DEFAULT NULL, `idService` varchar(5) DEFAULT NULL) ENGINE=MyISAM DEFAULT CHARSET=utf8mb3 COLLATE=utf8mb3_unicode_ci;

Je créé également une nouvelle table "service" qui va contenir les différents services de la médiathèque. Pour cela, je crée les champs "id" et "idService", j'insère les 4 services de la médiathèque "administratif", "prêts", "culture". et "administrateur". INSERT INTO service (id, libelle) VALUES (1, 'administratif'), (2, 'prêts'), (3, 'culture'), (0, 'administrateur');

J'insère également des comptes utilisateurs fictifs dans la table "utilisateur" : INSERT INTO utilisateur (id, login, password, idService) VALUES ('00001', 'CharlotteOFraise',

'CharlotteOFraise230', '0'), ('00002', 'ZakariZotto', 'ZakariZotto430', '1'), ('00003', 'BobLeponge', 'BobLeponge530', '2'), ('00004', 'RenaultMegane', 'RenaultMegane630', '3');

Enfin, je précise les clés correspondantes à la table "utilisateur" et "service"

FONCTIONNALITÉ D'AUTHENTIFICATION

Je retourne dans l'application C#, et j'ajoute une nouvelle vue "FrmAuthentification", dans le dossier "view", de type "Form".

Je place un panel qui va contenir le titre "Authentification", un label "Utilisateur" avec en dessous la textbox qui contiendra l'identifiant utilisateur, un label "Mot de passe", avec en dessous la textbox qui contiendra le mot de passe utilisateur, puis un bouton "Connexion" pour permettre la connexion à l'application.

_____photo authentificationè_____

Pour que la fenêtre d'authentification s'affiche avant la fenêtre de l'application, je vais dans la classe "Program.cs", et je remplace le "Application.Run" par "Application.Run(new FrmAuthentification());" dans la fonction "Main()" .

L'application va ainsi s'exécuter sur cette première fenêtre d'affichage.

Je créé ensuite un nouveau controller "FrmAuthentificationController", dans lequel je vais pouvoir récupérer l'identifiant de l'utilisateur et son mot de passe. Je créé également une nouvelle classe "Service" pour pouvoir récupérer les propriétés de la table "service", et une nouvelle classe "Utilisateur" pour récupérer les propriétés de la table "utilisateur".

Pour récupérer les données des tables "utilisateur" et "service", permettant ainsi l'authentification, je retourne dans l'api rest_mediatekdocuments et je créé la méthode "selectUtilisateur($id)" qui va récupérer la liste des utilisateurs ainsi que le service auquel ils sont affectés.

Dans cette méthode j'affecte le paramètre "id" pour récupérer un utilisateur spécifique, ensuite je fais un "Select" qui va récupérer le login utilisateur, le mot de passe utilisateur, l'id du service

de l'utilisateur, et le libelle du service. Enfin, j'effectue une jointure entre la table utilisateur et la table service, pour récupérer l'id du service.

J'appelle cette méthode dans un nouveau case de la méthode "selectOne($table, $id)", auquel j'affecte le endpoint "utilisateur". Méthode - selectUtilisateur($id) public function selectUtilisateur($id) { $param = array("id" => $id); $req = "select u.login, u.password , u.idService, s.libelle "; $req .= "from utilisateur u join service s on s.id=u.idService "; $req .= "where u.login =:id "; return $this->conn->queryAll($req, $param); }

Une fois la méthode construite dans l'api Rest, je vais dans la classe "Access" de l'application C#, et je créé la méthode "GetUtilisateur(string login, string password)" qui va permettre de récupérer l'utilisateur.

Je dois récupérer le login et le mot de passe correspondant aux identifiants de l'utilisateur, grâce à une requête http GET qui va récupérer la fonction "selectUtilisateur" de l'api REST, sur le endpoint "utilisateur/" , et j'y ajoute le "login".

Si le login existe, la fonction va récupérer l'utilisateur, sinon elle ne récupère rien du tout.

Méthode - GetUtilisateur(string login, string password)

public Utilisateur GetUtilisateur(string login, string password) { try { List liste = TraitementRecup(GET, "utilisateur/" + login); if (liste == null || liste.Count == 0) { return null; } Utilisateur utilisateur = liste[0]; if (utilisateur.Password != password) { return null; } return utilisateur;

Je peux maintenant récupérer les identifiants de l'utilisateur, pour cela je vais dans le "FrmAuthentificationController", et je créé la méthode d'authentification "GetUtilisateur(string login, string password)" .

J'appelle la méthode de récupération d'un utilisateur "GetUtilisateur(login, password)" de la classe "Access", puis, si l'utilisateur existe et possède un mot de passe, l'id du service et le libelle vont récupérer les valeurs de l'id du service d'un utilisateur ainsi que le libelle.

Méthode - GetUtilisateur(string login, string password)

public Service GetUtilisateur(string login, string password) { Utilisateur utilisateur = access.GetUtilisateur(login, password); if (utilisateur != null && utilisateur.Password.Equals(password)) { return new Service(utilisateur.IdService, utilisateur.Libelle); } return null; }

Je réalise ensuite la procédure évènementielle du clic sur le bouton "Connexion", que j'appelle "btnConnexion_Click", dans le "FrmAuthentification".

J'affecte dans les variables "utilisateur" et "password", les informations qui seront saisies par l'utilisateur dans les champs respectifs.

J'ajoute une condition "if" pour vérifier si les textbox sont vides, si elles ne le sont pas, j'appelle la méthode du controller "GetUtilisateur(utilisateur,password)" dans laquelle les valeurs saisies seront enregistrées, si les valeurs saisies ne correspondent pas à des identifiants valides dans la base de données, un message signale une erreur d'authentification.

Si l'utilisateur inscrit les bonnes informations de connexion, la fenêtre de l'application va s'afficher.

Procédure évènementielle - btnConnexion_Click

private void btnConnexion_Click(object sender, EventArgs e) { string login = txbLogin.Text; string password = txbPassword.Text; if (!txbLogin.Text.Equals("") && !txbPassword.Text.Equals("")) { Service service = controller.GetUtilisateur(login, password); if (service == null) { MessageBox.Show("Erreur d'authentification", "Alerte"); txbPassword.Text = ""; } else if(service.Libelle == "culture") { MessageBox.Show("Vous n'avez pas les droits d'accès à l'application.", "Alerte"); Application.Exit(); } else { MessageBox.Show("Vous êtes connecté", "Information"); FrmMediatek frmMediatek = new FrmMediatek(service); frmMediatek.ShowDialog(); } } }

Je dois mettre en place les habilitations d'accès, selon les différents services.

Pour cela, je vais dans le code du "FrmMediatek", et j'y inscris les différentes conditions d'affichage.

Les employés du service administratif et l'administrateur ont accès à toutes les fonctionnalités de l'application.

Étant donné qu'ils doivent gérer les commandes, j'ajoute une condition "if (Service.Libelle == "administratif || Service.Libelle == "administrateur", dans laquelle je vais afficher la fenêtre d'alerte.

Je place cette condition dans la procédure évènementielle "FrmAlerteFinAbonnement_Shown". Les employés du service de prêts peuvent seulement consulter le catalogue (livres, dvd, revues), je vais donc créer une condition qui masque les onglets de commandes lorsque le libelle du service est "prêts", et je vais désactiver tous les composants qui permettent de gérer le catalogue.

Pour cela, j'ai créé une méthode "AutorisationsAcces(Service service)", dans laquelle je récupère l'objet service, si le service est associé au libelle "prêts", alors les champs de gestion et les onglets de commandes seront masqués ou supprimés.

Les employés du service "culture" n'ont accès à aucune fonctionnalité j'ajoute donc un "else if(Service.Libelle == "culture") " dans la FrmAuthentification, méthode "btnConnexion_Click", dans lequel j'affiche un message à l'utilisateur pour l'informer, et unApplication.Exit() pour fermer l'application.

Je dois mettre en place les habilitations d'accès, selon les différents services.

Pour cela, je vais dans le code du "FrmMediatek", et j'y inscris les différentes conditions d'affichage.

Méthode - AutorisationsAcces

```
private void AutorisationsAcces(Service service) { if (service.Libelle == "prêts") {
tabOngletsApplication.TabPages.Remove(tabCommandesLivres);
tabOngletsApplication.TabPages.Remove(tabCommandesDvd);
tabOngletsApplication.TabPages.Remove(tabCommandesRevues); grpLivresInfos.Enabled =
false; txbExemplaireLivresNumero.Enabled = false; dtpDateAchatExemplaireLivre.Enabled =
```

false; cbxEtatLibelleExemplaireLivre.Enabled = false; btnEtatExemplaireLivreModifier.Enabled = false; btnExemplaireLivreSupprimer.Enabled = false; grpDvdInfos.Enabled = false; txbExemplaireDvdNumero.Enabled = false; dtpDateAchatExemplaireDvd.Enabled = false; cbxEtatLibelleExemplaireDvd.Enabled = false; btnEtatExemplaireDvdModifier.Enabled = false; btnExemplaireDvdSupprimer.Enabled = false; grpRevuesInfos.Enabled = false; txbReceptionExemplaireNumero.Enabled = false; dtpReceptionExemplaireDate.Enabled = false; txbReceptionExemplaireImage.Enabled = false; btnReceptionExemplaireImage.Enabled = false; btnReceptionExemplaireValider.Enabled = false; dtpDateAchatExemplaireRevue.Enabled = false; btnExemplaireRevueSupprimer.Enabled = false; cbxEtatLibelleExemplaireRevue.Enabled = false; btnEtatExemplaireRevueModifier.Enabled = false; } }

Les employés du service "culture" n'ont accès à aucune fonctionnalité j'ajoute donc un "else if(Service.Libelle == "culture") " dans la FrmAuthentification, méthode "btnConnexion_Click", dans lequel j'affiche un message à l'utilisateur pour l'informer, et unApplication.Exit() pour fermer l'application.

MISSION 5 : ASSURER LA SÉCURITÉ, LA QUALITÉ ET INTÉGRER DES LOGS

TÂCHE 1 - CORRIGER DES PROBLÈMES DE SÉCURITÉ

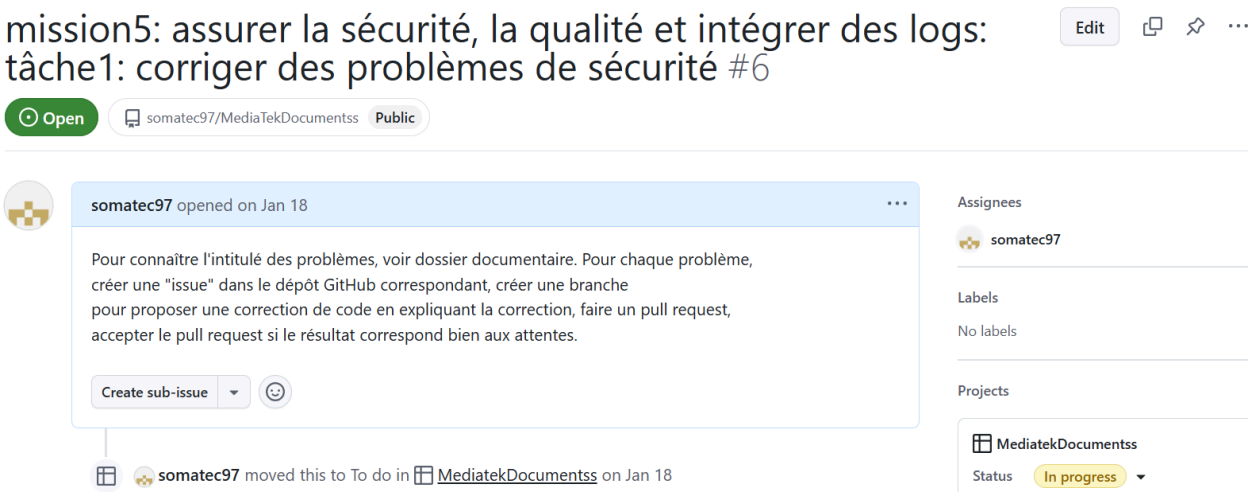
Pour connaître l'intitulé des problèmes, voir dossier documentaire.

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

Mémoriser le couple "login:pwd" dans App.config au lieu de le laisser en dur directement dans Access.php.

Modifier le fichier htacces pour prendre en compte une route vide et en tenir compte dans le fichier mediatekdocuments.php

Temps estimé 3h - Temps mis environ 2h



Pour résoudre le premier problème, je dois d'abord modifier le fichier App.config, pour y ajouter le nom de la chaîne de connexion et permettre à la connexion vers la base de données, de s'établir. J'y intègre le couple "admin:adminpwd", pour qu'il n'apparaisse plus en "dur" dans la classe Access. Ensuite, je retourne dans la classe "Access", et j'ajoute la propriété "connectionName", à laquelle j'affecte le nom de la connexion du fichier App.config, puis je créé un getter "GetConnectionStringByName(string name)", pour récupérer la chaîne de connexion. et je vérifie si l'objet "ConnectionStringSettings" n'est pas null, pour être sûre que la chaîne existe dans le fichier de configuration. Enfin, dans la propriété "Access()", j'insère la chaîne d'authentification dans le getter, dans la variable permettant l'authentification. Méthode - Access() private Access() { String authenticationString; try { authenticationString = GetConnectionStringByName(connectionName); api = ApiRest.GetInstance(uriApi, authenticationString); } catch (Exception e) { Console.WriteLine(e.Message); Environment.Exit(0); } } Je créé une nouvelle branche "problem", puis j'effectue un pull request afin de comparer les deux branches, et je fais un merge.