COMPTE RENDU D'ACTIVITE D'ATELIER PROFESSIONNEL

Le cas MediaTek86

Application C#

Tout le nécessaire pour se connecter aux différents outils sont fournis dans le fichier « LISEZ-MOI » en racine du projet disponible sur GitHub

Par Quentin Stachowiak

Mars-2022

SOMMAIRE

Table des matières

Contexte

Mediatek86 possède un média tek de livres, dvd et de revues. L'application existante permet la consultation de ces derniers. Toutefois, Mediatek86 souhaite faire évoluer cette application en faisant appel à mes services. Un dossier documentaire m'a été fourni avec la description de l'application est des fonctionnalités existantes, ainsi que les différentes missions qui me sont attribuées dans le cadre de l'évolution.

Environnement de production

Le développement se fait en .NET en C# dans l'IDE Visual Studio Community relié à un dépôt distant Github pour assurer le versionnage, régulièrement mit à jour via les commit. Pour gérer une bonne optimisation du code, j'utilise SonarQube dans Visual Studio

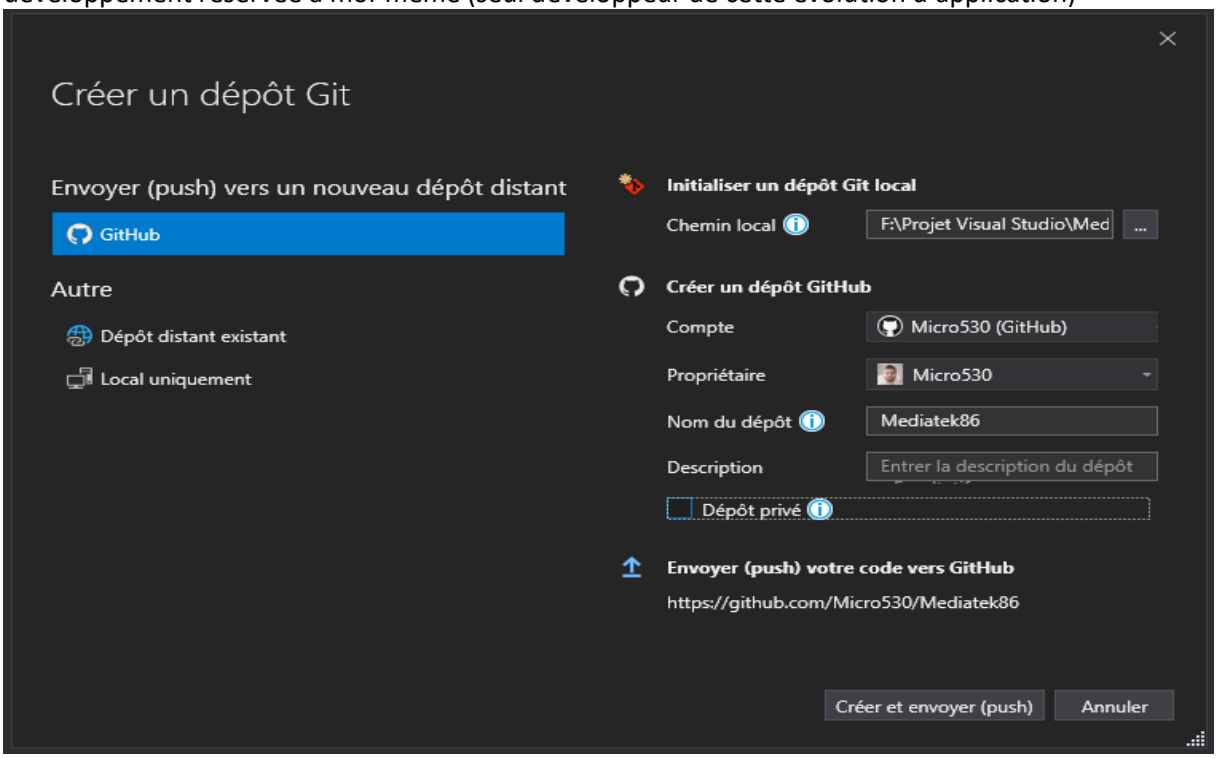
La base de données est gérée sous MySQL avec le SGBDR phpMyAdmin

L'outil de suivi de projet Trello sera également utilisé pour structurer l'avancée du projet.

En ce qui concerne l'hébergement, les services AWS seront utilisés.

Préparation de l'environnement

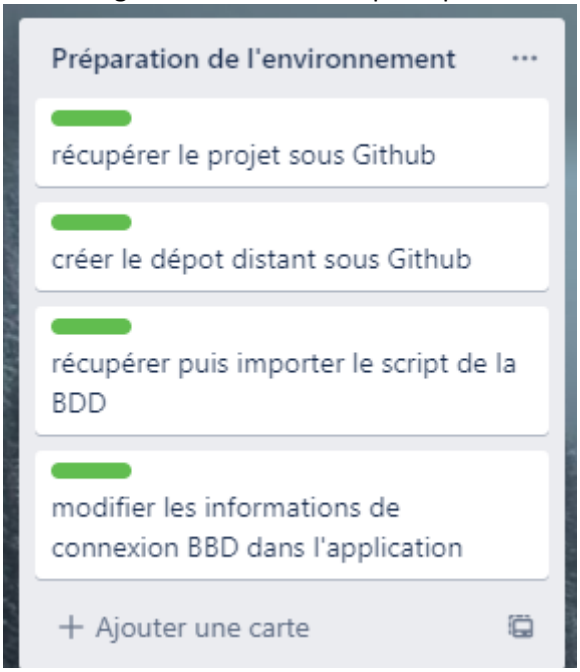
Je commence par récupérer le projet sous GitHub en zip que j'importe dans Visual Studio. À partir de là, je crée un dépôt GitHub distant pour assurer de versioning en créant également une branche développement réservée à moi-même (seul développeur de cette évolution d'application)



Je poursuis en récupérant puis important le script de la BBD dans phpMyAdmin. Je modifie les informations de connexion dans l'application et vérifie que la liaison fonctionne correctement.

```
private static readonly string server = "localhost";
private static readonly string userid = "root";
private static readonly string password = "W4F0I9Us0DJy";
private static readonly string database = "mediatek86";
private static readonly string connectionString = "server="+server+";user id="+userid+";password="+password+";database="+database+";SslMode=none";
```

Je mets également le Trello en place pour avoir un suivi d'activité des différentes missions.



Mission 1 : Gestion des documents

Supprimer un livre, un dvd ou des revues (un document)

Avant toute chose, pour respecter les contraintes d'intégrités dans la BDD, je crée un trigger sur la table livre_dvd qui est la table mère de livre et de dvd. Ce trigger va permettre en cas de suppression d'une des lignes dans livre_dvd, de supprimer également l'élément fille dans l'une des tables filles.

```
DELIMITER &&
CREATE TRIGGER BEFORE_delete_livredvd
 BEFORE DELETE ON livres_dvd
 FOR EACH ROW
BEGIN
 DELETE from livre where id = old.id;
 DELETE FROM dvd WHERE id = old.id;
END ;
&&
```

Dans la même logique, étant donné que livre_dvd est lui-même une table fille de la table document, je crée un trigger sur document qui va supprimer l'id dans ses tables filles

```
DELIMITER &&
CREATE TRIGGER BEFORE_delete_document
 BEFORE DELETE ON document
 FOR EACH ROW
BEGIN
 DELETE from livres_dvd where id = old.id;
 DELETE FROM revue WHERE id = old.id;
END ;
&&
```

À présent, dans l'application, je crée la nouvelle méthode dans DAO permettant de supprimer un document depuis un id transmis en paramètre

```
/// <summary>
/// suppression d'un element dans la table document
/// </summary>
/// <param name="id">id de l'element à supprimer</param>
3 références
public static void SupprDocument(string id)
{
    try
    {
        string req = "delete from document where id = @id";
        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@id", id}
        };
        BddMySql curs = BddMySql.GetInstance(connectionString);
        curs.ReqUpdate(req, parameters);
        curs.Close();

    }
    catch(Exception e)
    {
        Console.WriteLine(e);
    }
}
```

Cette méthode est appelée depuis le contrôleur qui renvoie le même id reçu en paramètre.

```
/// <summary>
/// Permet de supprimer le document envoyé
/// </summary>
/// <param name="id">id du document à supprimer</param>
0 références
public void SupprDocument(string id)
{
    Dao.SupprDocument(id);
}
```

Enfin, dans la vue, on commence par les livres. Nous créons deux méthodes. Une qui est l'événement du clic sur le bouton nouvellement créé « supprimer » et la seconde qui sert à demander la confirmation avant d'appeler la méthode du contrôleur.

```
/// <summary>
/// Evenement lors du clique sur le btnSupprLivre
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void btnSupprLivre_Click(object sender, EventArgs e)
{
    if(dgvLivresListe.CurrentCell.RowIndex != -1)
    {
        Livre livre = (Livre)dgvLivresListe.CurrentRow.DataBoundItem;
        if(ClickBtnSupprConfirmation("ce livre", livre.Titre, livre.Id))
        {
            dgvLivresListe.Rows.Remove(dgvLivresListe.CurrentRow);
        }
    }
}
/// <summary>
/// Méthode permettant de gérer la demande de confirmation de suppression
/// </summary>
/// <param name="message">element du message a rajouter pou personnaliser</param>
/// <param name="titre">titre de l'element</param>
/// <param name="id">id de l'element</param>
/// <returns></returns>
1 référence
private bool ClickBtnSupprConfirmation(string titre, string id, string message = "cette element")
{
    DialogResult dialogResult = MessageBox.Show($"Êtes-vous sûr de vouloir supprimer {message} nommé : '{titre}'.\n\nVoulez-vous continuer ?",
        "Confirmation de suppression", MessageBoxButtons.YesNo,MessageBoxIcon.Warning);
    if (dialogResult.Equals(DialogResult.Yes))
    {
        controle.SupprDocument(id);
        return true;
    }
    return false;

}
```

Grâce au fait que le dataGridView a été incrémenté par datasource à partir de la liste des livres, nous pouvons directement récupérer l'objet livre depuis la ligne sélectionnée. Reste plus qu'à appeler la seconde méthode en envoyant les paramètres demandés et de retirer la ligne sélectionnée si l'utilisateur confirme son choix.

Il ne reste plus qu'à créer les deux autres boutons sur les deux autres onglets et appliquer la même logique.

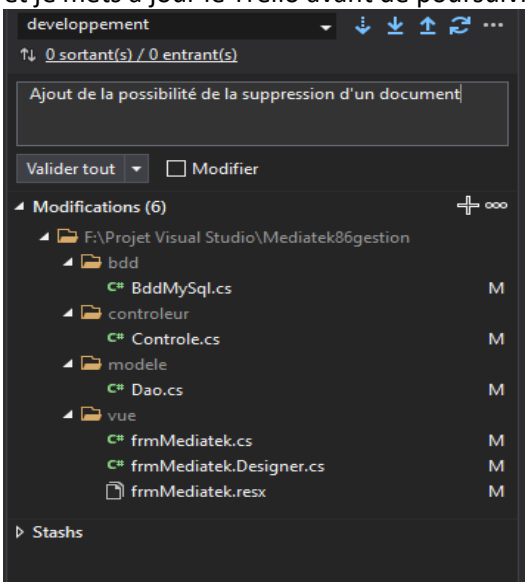
Il reste maintenant à empêcher cette suppression si ce document possède des exemplaires ou bien fait partie d'une commande. Pour ce faire nous allons créer un trigger qui va contrôler ça pour nous à la demande de suppression d'un document. Celui-ci se contente de faire des requêtes et de vérifier qu'elle retourne zéro, ce qui signifie qu'il n'y a pas d'autres références de ce document ailleurs.

```
DELIMITER &&
CREATE TRIGGER Controlepresence_before_delete
BEFORE DELETE ON document FOR EACH ROW

 BEGIN

 DECLARE nbIdExemplaire int;
 DECLARE nbIdCommande int;

 Select count(*) into nbIdExemplaire from exemplaire where exemplaire.id = old.id ;
 IF(nbIdExemplaire <> 0) THEN
 SIGNAL SQLSTATE "45000"
    SET MESSAGE_TEXT = "Opération impossible : Ce document est présent dans un exemplaire" ;
 END IF ;

 Select count(*) into nbIdCommande from commandedocument where idLivreDvd = old.id ;
 IF(nbIdCommande <> 0) THEN
 SIGNAL SQLSTATE "45000"
    SET MESSAGE_TEXT = "opération impossible : ce document est présent dans une commande" ;
 END IF ;

CALL siPeutEtreSupprime(old.id);

END;
&&
```

Le trigger déjà présent sur la table document est converti en procédure et est appelé en fin de ce trigger si aucunes erreurs ne l'a arrêté plus tôt.

```
DELIMITER &&
CREATE PROCEDURE siPeutEtreSupprime(IN idAVerifier int)
BEGIN
 DELETE from livres_dvd where id = idAVerifier;
 DELETE FROM revue WHERE id = idAVerifier;
END;
&&
```

Afin de renvoyer le message d'erreur à l'utilisateur, si jamais le document était présent quelque part, je récupère l'exception et je la mets dans un MessageBox. Je fais un commit & push sur le dépôt distant et je mets à jour le Trello avant de poursuivre.

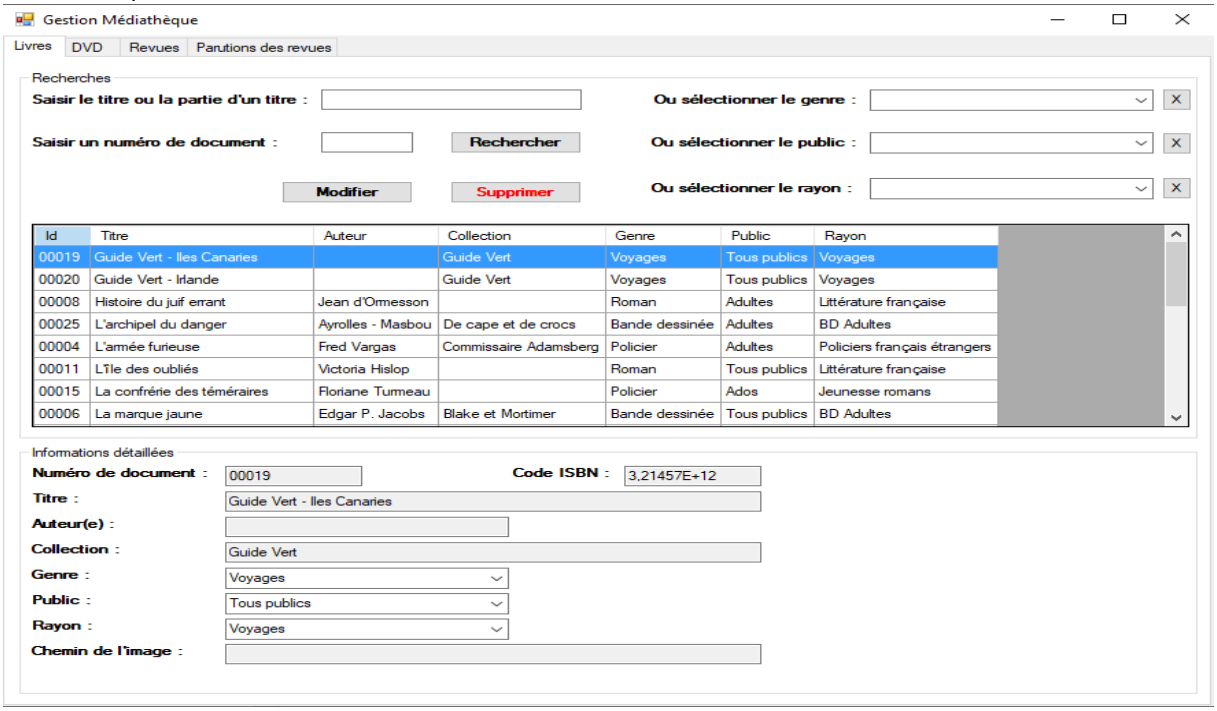


Modifier un document

Je commence par modifier la vue en retirant les zones de textes où les genres, les publics et les rayons étaient écrit pour les remplacer par des ComboBox qui pourront facilement être modifié par l'utilisateur. Dans la vue « FrmMediatek » je crée des nouveaux BindingSource pour les nouveaux ComboBox (si je réutilise les mêmes que pour les ComboBox permettant de filtrer par genre, ils seront synchronisés). Pour éviter de trop solliciter inutilement la base de données (étant donné que j'ai retiré le fait que le contrôleur gardait en mémoire les listes récupérées de la BDD ce qui n'était pas pratique pour remettre à jour après un changement, cette mémorisation se faire directement dans FrmMediatek), je crée de nouvelles listes de catégories pour mémoriser toutes ces informations. Dans AfficheLivresInfos, je fais en sorte que les ComboBox affichent les informations du livre actuellement sélectionné.

```
/// <summary>
/// Affichage des informations du livre sélectionné
/// </summary>
/// <param name="livre"></param>
1 référence
private void AfficheLivresInfos(Livre livre)
{
    txbLivresAuteur.Text = livre.Auteur;
    txbLivresCollection.Text = livre.Collection;
    txbLivresImage.Text = livre.Image;
    txbLivresIsbn.Text = livre.Isbn;
    txbLivresNumero.Text = livre.Id;

    //récupère les genre, les publics et lesrayons du livre et l'affiche dans le combobox
    Genre genreLivre = lesGenres.Cast<Genre>().Where((Genre g) => g.Libelle == livre.Genre).First();
    comboLivreGenre.SelectedItem = genreLivre;

    Public publicLivre = lesPublics.Cast<Public>().Where((Public p) => p.Libelle == livre.Public).First();
    comboLivrePublic.SelectedItem = publicLivre;

    Rayon rayonLivre = lesRayons.Cast<Rayon>().Where((Rayon r) => r.Libelle == livre.Rayon).First();
    comboLivreRayon.SelectedItem = rayonLivre;

    txbLivresTitre.Text = livre.Titre;
    string image = livre.Image;
    try
    {
        pcbLivresImage.Image = Image.FromFile(image);
    }
    catch
    {
        pcbLivresImage.Image = null;
    }
}
```

Voici le résultat de la vue (j'ai déjà commencé à placer le bouton « modifier » qui n'est pour l'instant relié à rien)



Je crée ensuite dans la vue 3 méthodes : la première est l'événement du clic sur le btnModificationLivre, la seconde est l'événement du clic sur le btnValiderLivre après avoir effectué les modifications. Cette méthode appelle la troisième en lui envoyant son objet de type Button. La troisième méthode permet, en contrôlant de quel bouton il s'agit, de créer soit un livre soit les autres possibilités que je ferais un peu après. Elle récupère alors tous les éléments des champs de saisies puis créer un objet qui est envoyé au contrôleur.

```
/// <summary>
/// Evenement lors du clique sur le btnModifierLivre
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void btnModifierLivre_Click(object sender, EventArgs e)
{
    grpLivresInfos.Enabled = true;
    grpLivresRecherche.Enabled = false;
}
/// <summary>
/// Méthode permettant de moddifier un document
/// </summary>
1 référence
private void ModificationDocument(Object sender)
{

    string nomBtn = ((Button)sender).Name;
    switch (nomBtn)
    {
        case "btnValiderModificationLivre":
            Livre unLivre = new Livre(txbLivresNumero.Text, txbLivresTitre.Text, txbLivresImage.Text, txbLivresIsbn.Text, txbLivresAuteur.Text, txbLivresCollection.Text,
        ((Genre)comboLivreGenre.SelectedValue).Id, ((Genre)comboLivreGenre.SelectedValue).Libelle, ((Public)comboLivrePublic.SelectedValue).Id,
        ((Public)comboLivrePublic.SelectedValue).Libelle, ((Rayon)comboLivreRayon.SelectedValue).Id, ((Rayon)comboLivreRayon.SelectedValue).Libelle);
            controle.ModifierLivre(unLivre);
            RemplirLivresListeComplete(controle.GetAllLivres());
            break;
    }
}
/// <summary>
/// Evenement lors du clique sur le btnValiderLivre
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void btnValiderLivre_Click(object sender, EventArgs e)
{
    ModificationDocument(sender);
}
```

Le contrôleur se contente de recevoir le livre puis de le renvoyer au DAO sur la méthode ModifierLivre. Dans le DAO, deux méthodes sont créées, une pour modifier un document et l'autre pour appeler cette première méthode puis créer un livre.

```
/// <summary>
/// Modification d'un document
/// </summary>
/// <param name="document">document à modifier</param>
/// <returns>true si la modification à pu se faire</returns>
1 référence
private static bool ModifierDocument(Document document)
{
    try
    {
        string req = "UPDATE document set titre = @titre, image = @image, idRayon = @idRayon, idPublic = @idPublic, idGenre = @idGenre where id = @id";
        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@id", document.Id},
            { "@titre", document.Titre},
            { "@image", document.Image},
            { "@idRayon", document.IdRayon},
            { "@idPublic",document.IdPublic},
            { "@idGenre",document.IdGenre}
        };
        BddMySql curs = BddMySql.GetInstance(connectionString);
        curs.ReqUpdate(req, parameters);
        curs.Close();
        return true;
    }
    catch
    {
        return false;
    }
}
/// <summary>
/// Modification d'un document
/// </summary>
/// <param name="livre">document à modifier</param>
/// <returns>true si la modification à pu se faire</returns>
1 référence
public static bool ModifierLivre(Livre livre)
{
    try
    {
        Document document = new Document(livre.Id, livre.Titre, livre.Image, livre.IdGenre, livre.Genre, livre.IdPublic,livre.Public, livre.IdRayon, livre.Rayon);
        if (!ModifierDocument(document))
        {
            throw new Exception();
        }
        string req = "UPDATE livre set ISBN = @ISBN, auteur = @auteur, collection = @collection where id = @id";
        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@id", livre.Id},
            { "@ISBN", livre.Isbn},
            { "@auteur", livre.Auteur},
            { "@collection", livre.Collection}
        };
        BddMySql curs = BddMySql.GetInstance(connectionString);
        curs.ReqUpdate(req, parameters);
        curs.Close();
        return true;
```

Dans cette logique, je crée les méthodes et les boutons nécessaires à la modification des DVD et des revues.

Ajouter un document

À présent, il ne reste plus qu'à ajouter la possibilité d'ajouter un document. La plupart de ce qui est nécessaire à déjà été créées pour la modification. Pour commencer, je crée un nouveau bouton qui permet l'ajout, le mécanisme est le même que pour la modification. Je crée aussi une variable Boolean qui est instancié à True si nous somme en cours de création (afin de pouvoir réutiliser la même méthode que pour la modification) Dans cette méthode je crée un if qui compare le Boolean puis je crée un deuxième switch case qui est très proche du premier avec les petites subtilités d'un ajout à la place d'une modification (ici, seuls les livres sont ajoutable pour l'instant).

```
switch (nomBtn)
{
    case "btnValiderModificationLivre":
        int id = this.lesLivres.Max((Livre l) => int.Parse(l.Id)) + 1;
        Livre unLivre = new Livre(conversionIDBdd(id), txbLivresTitre.Text, txbLivresImage.Text, txbLivresIsbn.Text, txbLivresAuteur.Text, txbLivresCollection.Text,
    ((Genre)comboLivreGenre.SelectedValue).Id, ((Genre)comboLivreGenre.SelectedValue).Libelle, ((Public)comboLivrePublic.SelectedValue).Id,
    ((Public)comboLivrePublic.SelectedValue).Libelle, ((Rayon)comboLivreRayon.SelectedValue).Id, ((Rayon)comboLivreRayon.SelectedValue).Libelle);
        controle.AjouterLivre(unLivre);
        RemplirLivresListeComplete(controle.GetAllLivres());
        grpLivresInfos.Enabled = false;
        grpLivresRecherche.Enabled = true;
        break;
    case "btnValiderModifDvd":
        Dvd unDvd = new Dvd(txbDvdNumero.Text, txbDvdTitre.Text, txbDvdImage.Text, int.Parse(txbDvdDuree.Text), txbDvdRealisateur.Text,
            txbDvdSynopsis.Text, ((Genre)comboDvdGenre.SelectedValue).Id, ((Genre)comboDvdGenre.SelectedValue).Libelle,
            ((Public)comboDvdPublic.SelectedValue).Id, ((Public)comboDvdPublic.SelectedValue).Libelle, ((Rayon)comboDvdRayon.SelectedValue).Id,
            ((Rayon)comboDvdRayon.SelectedValue).Libelle);
        controle.ModifierDvd(unDvd);
        RemplirDvdListeComplete(controle.GetAllDvd());
        grpDvdInfos.Enabled = false;
        grpDvdRecherche.Enabled = true;
        break;
    case "btnValideModifRevue":
        Revue revue = new Revue(txbRevuesNumero.Text, txbRevuesTitre.Text, txbRevuesImage.Text, ((Genre)comboRevueGenre.SelectedValue).Id, ((Genre)comboRevueGenre.Select
            ((Public)comboRevuePublic.SelectedValue).Id, ((Public)comboRevuePublic.SelectedValue).Libelle, ((Rayon)comboRevueRayon.SelectedValue).Id,
            ((Rayon)comboRevueRayon.SelectedValue).Libelle, chkRevuesEmpruntable.Checked, txbRevuesPeriodicite.Text, int.Parse(txbRevuesDateMiseADispo.Text));
        controle.ModifierRevue(revue);
        RemplirRevuesListeComplete(controle.GetAllRevues());
        grpRevuesInfos.Enabled = false;
        grpRevuesRecherche.Enabled = true;
        break;
}
```

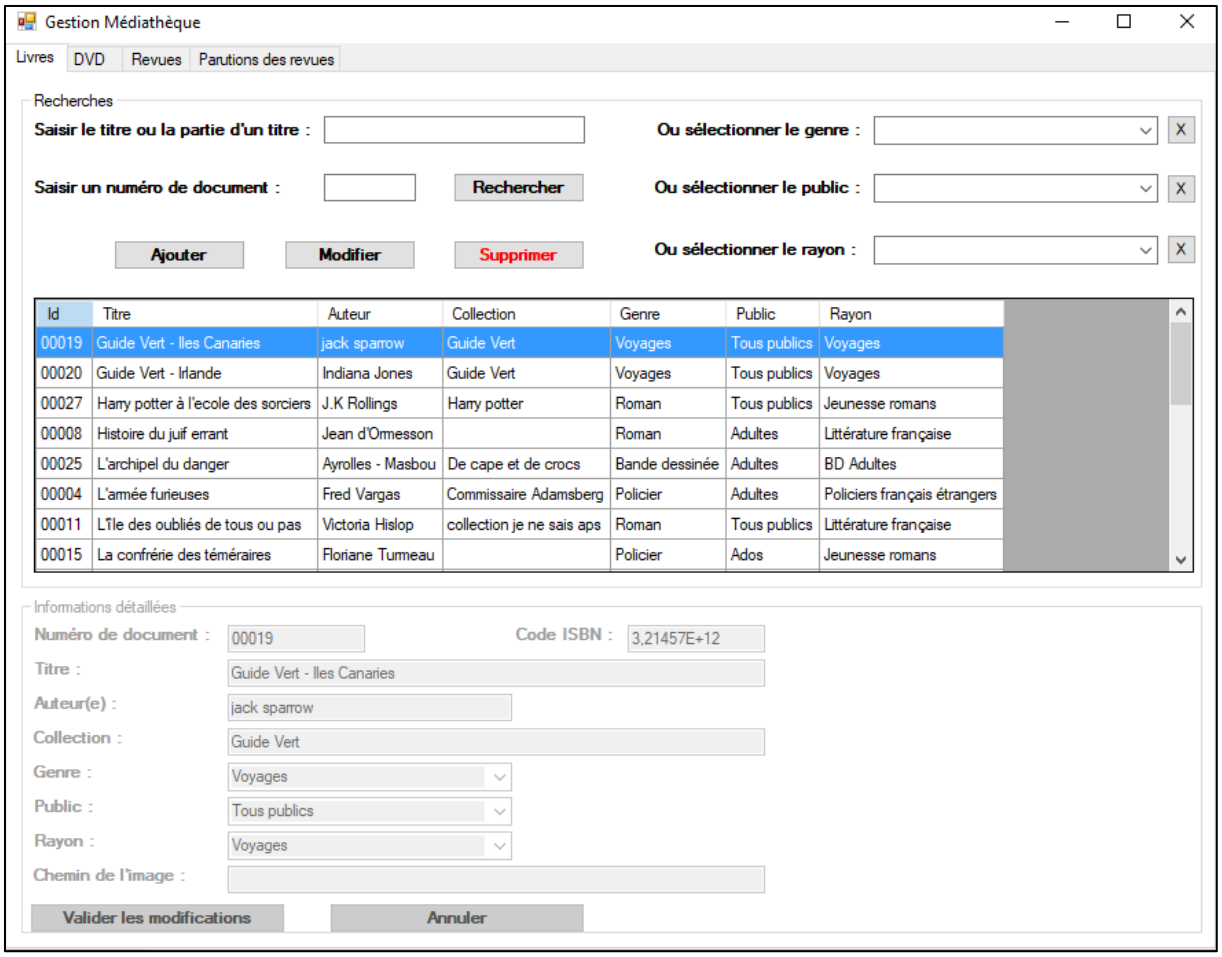
J'ai créé une petite méthode qui permet de formater l'id créé lors de l'ajout pour qu'il corresponde aux choix faits par le créateur de la BDD

```
private string conversionIDBdd(int id)
{
    if (id <= 9)
    {
        return "0000" + id;
    }else if(id <= 99)
    {
        return "000" + id;
    }else if(id <= 999)
    {
        return "00" + id;
    }else if(id <= 9999)
    {
        return "0" + id;
    }
    else
    {
        return id.ToString();
    }
}
```

À présent je crée la méthode du contrôleur qui renvoie le livre nouvellement créé au DAO. Dans ce Dao le principe est le même que pour la modification, seul la requête UPDATE se transforme en INSERT est le reste est quasiment identique.

Je continue pour faire en sorte que ce soit possible d'ajouter également des dvd et des revues.

Voici le résultat de la vue après l'ajout des options d'ajouts, de modifications et de suppressions

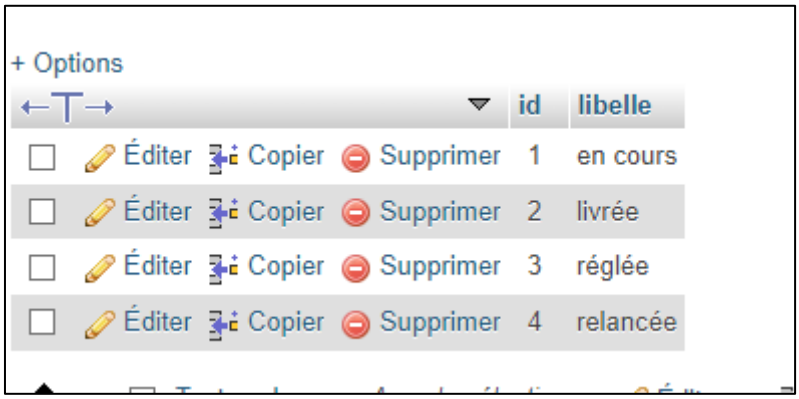


Je fais un commit & push sur GitHub, je mets à jour le Trello

Mission 2 : Gérer les commandes

Modification dans la BDD – Ajout de la table Suivi

Je commence par créer la nouvelle table demandée dans PhpMyAdmin. Puis je la relie en clé étrangère à « commandedocument »



Modification des classes métiers

À présent, il faut rajouter dans les classes métiers l'équivalent des deux tables « commandeDocument » et « document ». Dans la BDD « commande » est enfant de « commandeDcument », je vais représenter le même héritage dans les classes métiers. Les constructeurs et l'encapsulation des champs sont faits grâce aux méthodes de refactorisations

```
3 références
public class CommandeDocument
{
    private string id;
    private int nbExemplaire;
    private string idLivreDvd;
    private int idSuivi;
    private string suivi;

    1 référence
    public CommandeDocument(string id, int nbExemplaire, string idLivreDvd, int idSuivi, string suivi)
    {
        this.Id = id;
        this.NbExemplaire = nbExemplaire;
        this.IdLivreDvd = idLivreDvd;
        this.IdSuivi = idSuivi;
        this.Suivi = suivi;
    }

    1 référence
    public string Id { get => id; set => id = value; }
    1 référence
    public int NbExemplaire { get => nbExemplaire; set => nbExemplaire = value; }
    1 référence
    public string IdLivreDvd { get => idLivreDvd; set => idLivreDvd = value; }
    1 référence
    public int IdSuivi { get => idSuivi; set => idSuivi = value; }
    1 référence
    public string Suivi { get => suivi; set => suivi = value; }
}
```

```
public class Commande : CommandeDocument
{
    private DateTime dateCommande;
    private double montant;

    1 référence
    public Commande(DateTime dateCommande, double montant, string id, int nbExemplaire, string idLivreDvd, int idSuivi, string suivi)
        : base(id, nbExemplaire, idLivreDvd, idSuivi, suivi)
    {
        this.DateCommande = dateCommande;
        this.Montant = montant;
    }

    1 référence
    public DateTime DateCommande { get => dateCommande; set => dateCommande = value; }
    1 référence
    public double Montant { get => montant; set => montant = value; }
}
```

Ensuite, je crée une nouvelle méthode dans le DAO avec une requête pour récupérer l'ensemble des commandes par rapport à un idLivreDvd, cette commande est assez similaire à celle permettant de récupérer les livres par exemple.

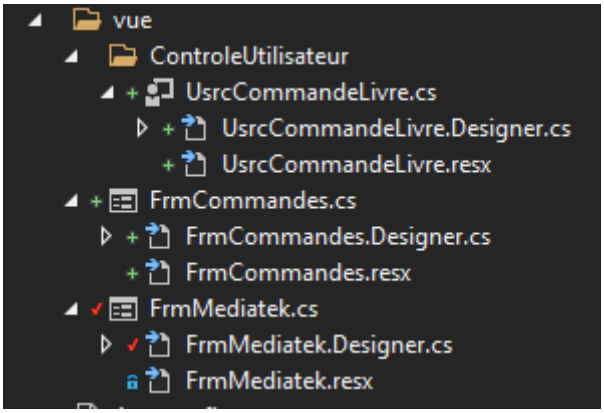
```
/// <summary>
/// Retourne toutes les commande à partir de l'id d'un livre_dvd
/// </summary>
/// <returns>Liste d'objets Commande</returns>
1 référence
public static List<Commande> GetAllCommandes(string idLivre_Dvd)
{
    List<Commande> lesCommandes = new List<Commande>();
    string req = "Select d.id, d.nbExemplaire, d.idLivreDvd, d.idSuivi, s.libelle as suivi, c.dateCommande, c.montant ";
    req += "from commandedocument d join commande c on d.id=c.id ";
    req += "join suivi s on s.id=d.idSuivi ";
    req += "where d.idLivreDvd = @idLivreDvd";
    //req += "order by dateCommande DESC";
    Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@idLivreDvd", idLivre_Dvd}
        };
    BddMySql curs = BddMySql.GetInstance(connectionString);
    curs.ReqSelect(req, parameters);

    while (curs.Read())
    {
        string id = (string)curs.Field("id");
        int nbExemplaire = (int)curs.Field("nbExemplaire");
        string idLivreDvd = (string)curs.Field("idLivreDvd");
        int idSuivi = (int)curs.Field("idSuivi");
        string suivi = (string)curs.Field("suivi");
        DateTime dateCommande = (DateTime)curs.Field("dateCommande");
        double montant = (double)curs.Field("montant");
        Commande commande = new Commande(dateCommande,montant,id,nbExemplaire,idLivreDvd,idSuivi, suivi);
        lesCommandes.Add(commande);
    }
    curs.Close();

    return lesCommandes;
}
```

Je crée également la méthode dans le constructeur permettant d'appeler cette méthode.

Création de l'interface des commandes

Ici je vais prendre des libertés par rapport aux instructions de la mission parce que je trouve la proposition pas très optimisée aussi bien qu'en performance qu'en facilité de maintenant. Ici il est demandé de créer un nouvel onglet pour gérer les commandes des livres, puis faire de même avec les dvd, etc. Le souci c'est que la vue « FrmMediatek86 » fait déjà près de 1800 lignes avec tous les onglets, objets graphiques, etc. J'ai pris la décision de créer un nouveau formulaire dédié aux commandes qui lui se contentera (en fonction de ce qu'il recevra) d'ouvrir un UserControle (ici seulement un seul UserControle « Commande » sera créer. Mais cela laissera une possibilité, dans l'optique de maintenant et d'évolution, d'ajouter plus de formulaires par la suite). Cela facilite grandement de développement et surtout la future maintenance.
Dans un souci de toujours respecter le MVC, je crée dans la vue un nouveau package ControleUtilisateur qui contiendra tous mes UserControle. Je crée aussi dans la foulée un nouveau formulaire et le premier UserControle qui géra les commandes.

Également, il est demandé dans la mission (dans le nouvel onglet qui devrait normalement être créé), de permettre la sélection d'un document par son titre, son genre, etc… À la place de récréer cette interface je vais utiliser les onglets déjà existant qui permettent cette sélection. Je vais juste ajouter un bouton qui permettra d'accéder aux commandes du document sélectionné dans le DataGridView.



Le nouveau formulaire « commande » ne contiendra aucun objet graphique, il se contentera d'appeler les UserControle qui eux contiendront les objets graphiques. Pour commencer ce formulaire aura besoin de deux choses dans sa signature, un objet (qui pourra être un livre ou n'importe quoi d'autres) et l'instance du contrôle qui l'a créé. Lors de son ouverture, il va comparer l'objet reçu pour ouvrir le bon UserControle.

```
private UsrcCommandeLivre usrcCommandeLivre;
private Livre unLivre;
private Dvd unDvd;
private Revue uneRevue;
private Object unObjet;
private Controle controle;
1 référence
public FrmCommandes(Object unObjet, Controle controle)
{
    InitializeComponent();
    this.unObjet = unObjet;
    this.controle = controle;

}

1 référence
private void Commandes_Load(object sender, EventArgs e)
{
    if (unObjet is Livre)
    {
        unLivre = (Livre)unObjet;
        ConsUsrcCommandeLivre();
    }
    else if (unObjet is Dvd)
    {

    }
    else
    {

    }
}
```

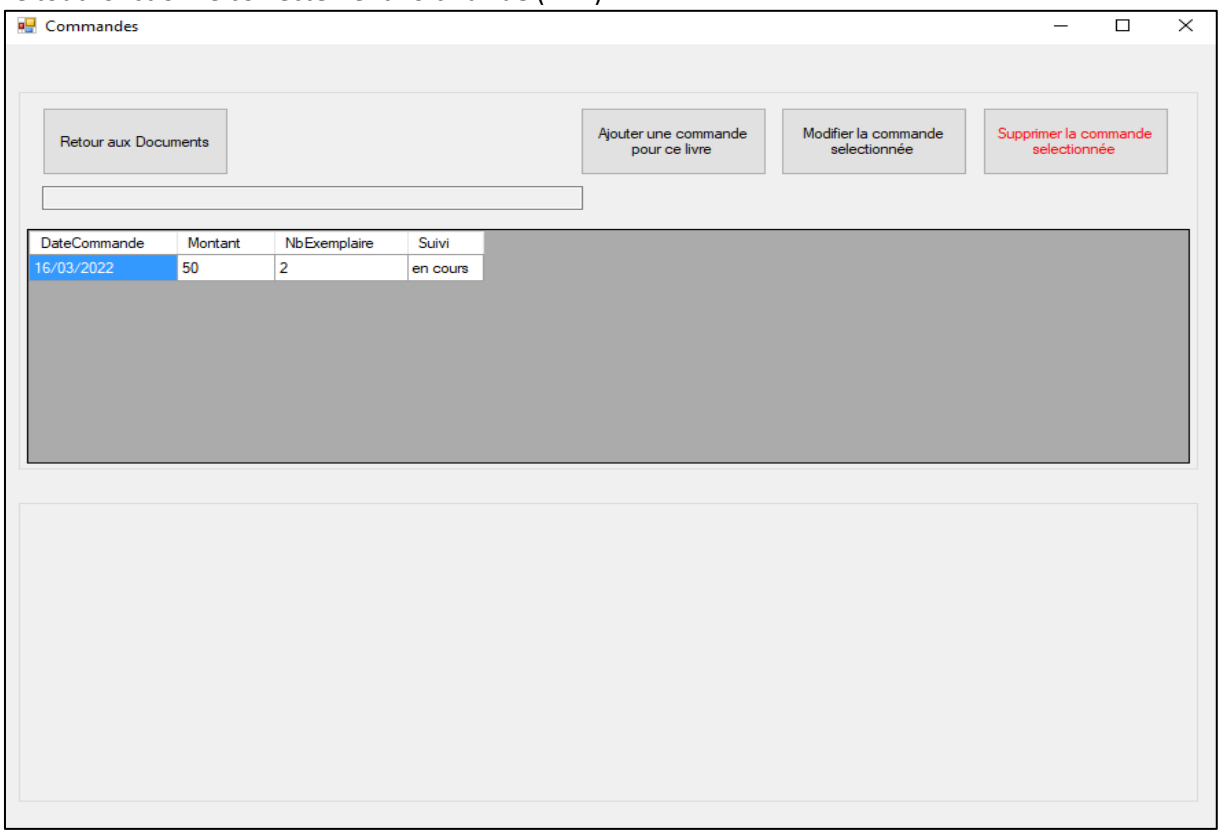
```
private void ConsUsrcCommandeLivre()
{
    usrcCommandeLivre = new UsrcCommandeLivre(this, unLivre);
    usrcCommandeLivre.Location = new Point(0, 0);
    this.Controls.Add(usrcCommandeLivre);
}
1 référence
public List<Commande> GetAllCommandes(string idLivre_Dvd)
{
    return controle.GetAllCommandes(idLivre_Dvd);
}
```

En même temps, je crée la méthode qui appelle celle du contrôleur pour récupérer les commandes (ici le formulaire va agir comme un mini contrôleur pour les UserControle qui ne pourront pas directement accéder au contrôleur)

Dans le UserControle, je commence à tester l'ensemble de ce qui a été créé plus tôt en valorisant un DataGridView.

```
4 références
public partial class UsrcCommandeLivre : UserControl
{
    private FrmCommandes frmCommandes;
    private BindingSource bdgCommandesListe = new BindingSource();
    private Livre unLivre;
    1 référence
    public UsrcCommandeLivre(FrmCommandes frm, Livre livre)
    {
        InitializeComponent();
        frmCommandes = frm;
        unLivre = livre;
        remplirDgvListCommande(frmCommandes.GetAllCommandes(unLivre.Id));
    }
    1 référence
    public void remplirDgvListCommande(List<Commande> commandes)
    {
        bdgCommandesListe.DataSource = commandes;
        dgvListCommande.DataSource = bdgCommandesListe;
        dgvListCommande.Columns["id"].Visible = false;
        dgvListCommande.Columns["idLivreDvd"].Visible = false;
        dgvListCommande.Columns["idSuivi"].Visible = false;
        dgvListCommande.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
        dgvListCommande.Columns["id"].DisplayIndex = 0;
    }
}
```

Le tout fonctionne correctement voici la vue (WIP)



Ajout d'une nouvelle commande de Livre/DVD

Je continue de remplir le formulaire avec les objets graphiques manquant pour pouvoir afficher toutes les infos requises des commandes. Je crée deux boutons : un pour valider un ajout ou une modification et l'autre pour annuler l'action. Lors du clic sur le bouton une série de champs et de zones s'activent et se désactivent afin de contrôler les possibilités de l'utilisateur. Lors du clic sur le bouton valider, une commande est créée puis envoyé au formulaire commande qui l'envoie au contrôleur qui l'envoie à

son tour au DAO. Notez qu'une variable Boolean est géré pour savoir si c'est un ajout ou une modification, c'est le formulaire commande qui par un if le détermine.

```
2 références
private void btnAnnulerModif_Click(object sender, EventArgs e)
{
    grbListCommande.Enabled = true;
    grbInfoCommande.Enabled = false;
    ajoutEnCours = false;
    txtMontant.Enabled = true;
    txtNbExemplaire.Enabled = true;
    CalendrierCommandeLivre.Enabled = true;
    VideCommandeInfos();
    remplirDgvListCommande(frmCommandes.GetAllCommandes(unLivre.Id));
}

1 référence
private void btnValiderModif_Click(object sender, EventArgs e)
{
    Suivi suivi = (Suivi)comboSuivi.SelectedItem;
    if(CommandeSelectionne.IdSuivi <= suivi.Id)
    {
        Commande commande = new Commande(ConversionDateBdd(CalendrierCommandeLivre.SelectionStart), double.Parse(txtMontant.Text),
            CommandeSelectionne.Id, int.Parse(txtNbExemplaire.Text), CommandeSelectionne.IdLivreDvd, suivi.Id, suivi.Libelle);
        frmCommandes.ModifierAjouterCommande(commande, ajoutEnCours);
        btnAnnulerModif_Click(null, null);
    }
    else
    {
        MessageBox.Show("Changement du statut de la commande incohérent, vérifiez les modifications", "Statut incohérent", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }


}
```

Dans le DAO les choses se compliquent, en effet pour qu'une commandeDocument puisse exister, il faut d'abord qu'une commande possède l'id qu'elle va utiliser. Pour ce faire, je crée une autre méthode qui va se charger de créer la commande avant celle de commandeDocument (c'est cette dernière qui va l'appeler avant d'exécuter ses propres actions).

```
/// <summary>
/// créer une commande de document
/// </summary>
/// <param name="commande"></param>
/// <returns>true si l'insertion a pu se faire</returns>
1 référence
public static bool CreerCommandeLivreDvd(Commande commande)
{
    string id = CreerCommande(commande);
    try
    {
        string req = "insert into commandeDocument values (@id, @nbExemplaire, @idLivreDvd, @idSuivi)";
        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@id", id},
            { "@nbExemplaire", commande.NbExemplaire},
            { "@idLivreDvd", commande.IdLivreDvd},
            { "@idSuivi", commande.IdSuivi}
        };
        BddMySql curs = BddMySql.GetInstance(connectionString);
        curs.ReqUpdate(req, parameters);
        curs.Close();
        return true;
    }
    catch
    {
        return false;
    }
}
```

C'est dans la méthode créerCommande que les choses deviennent compliqués. Pour créer une commande, il faut connaître son id parce qu'il n'est pas en auto incrémente dans la BDD parce qu'il est en string. Pour ça, je fais une requête ou je récupère tous les id des commandes dans une liste, puis je convertis chacun de ces id dans une autre liste en int avec le int.Parse. Enfin je peux exécuter la

méthode Max() sur la liste de int pour en déterminer la valeur la plus élevée. Je fais + 1 à cette valeur, je possède maintenant l'id dont j'ai besoins. Je l'utilise pour créer ma commande.

```
public static string CreerCommande(Commande commande)
{
    string idString = "";
    try
    {
        List<string> lesId = new List<string>();
        List<int> lesIdInt = new List<int>();
        string req = "Select id ";
        req += "from commande";
        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@montant", commande.Montant},
            { "@dateCommande", commande.DateCommande}
        };

        BddMySql curs = BddMySql.GetInstance(connectionString);
        curs.ReqSelect(req, parameters);
        while (curs.Read())
        {
            string id = (string)curs.Field("id");
            lesId.Add(id);
        }
        curs.Close();
        foreach(string id in lesId)
        {
            lesIdInt.Add(int.Parse(id));
        }
        idString = (lesIdInt.Max() + 1).ToString();
    }
    catch(Exception e)
    {
        Console.WriteLine(e.Message);
    }
    try
    {
        string req = "insert into commande values (@id, @dateCommande, @montant)";
        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@id", idString},
            { "@montant", commande.Montant},
            { "@dateCommande", commande.DateCommande}
        };
        BddMySql curs = BddMySql.GetInstance(connectionString);
        curs.ReqUpdate(req, parameters);
        curs.Close();
    }
    catch(Exception e)
    {
        Console.WriteLine(e.Message);
    }
    return idString;
}
```

Notez que cette méthode retourne un string, il s'agit de l'id qui a été créé, il est renvoyé pour être utilisé dans la méthode précédente afin de pouvoir créer la commandeDocument avec ce même id.

À présent, la modification se base sur le même principe, à ceci près qu'il est plus simple de gérer la partie commande puisqu'on connaît déjà l'id vu que c'est une modification.

```
public static bool ModifierCommande(Commande commande)
{
    try
    {
        string req = "UPDATE commande set dateCommande = @dateCommande, montant = @montant where id = @id";
        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@id", commande.Id},
            { "@dateCommande", commande.DateCommande},
            { "@montant", commande.Montant}
        };
        BddMySql curs = BddMySql.GetInstance(connectionString);
        curs.ReqUpdate(req, parameters);
        curs.Close();
        return true;
    }
    catch
    {
        return false;
    }
}
/// <summary>
/// Modification d'un document
/// </summary>
/// <param name="document">document à modifier</param>
/// <returns>true si la modification à pu se faire</returns>
1 référence
public static bool ModifierCommandeLivreDvd(Commande commande)
{
    ModifierCommande(commande);
    try
    {
        string req = "UPDATE commandedocument set nbExemplaire = @nbExemplaire, idLivreDvd = @idLivreDvd, idSuivi = @idSuivi where id = @id";
        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@id", commande.Id},
            { "@nbExemplaire", commande.NbExemplaire},
            { "@idSuivi", commande.IdSuivi},
            { "@idLivreDvd", commande.IdLivreDvd}
        };
        BddMySql curs = BddMySql.GetInstance(connectionString);
        curs.ReqUpdate(req, parameters);
        curs.Close();
        return true;
    }
```

Dans le reste du code il y a assez peu de modification puisque que toutes les méthodes ont été créées pour gérer les deux situations dans le formulaire ainsi que dans le UserControle. Les nouvelles méthodes créées sont celles du contrôleur qui sont basées sur les mêmes principes que déjà vus.

Maintenant que nous pouvons ajouter et modifier les commandes d'un livre, il faut gérer les restrictions qui concernent c'est dernier. Dans la base de données il faut gérer via un trigger le fait que si une commande passe à l'état livré, la table exemplaire ce rempli avec les nouveaux livres en incrémentant avec un numéro chaque livre (en fonction du nombre de livre de la commande) et le code état « neuf»

```
DELIMITER &&
CREATE TRIGGER after_update_commandedocument AFTER UPDATE on commandedocument FOR EACH ROW
BEGIN
 DECLARE nbExemplaireLivre int;
 DECLARE fin int DEFAULT 0;
 DECLARE numeroExemp int;
 DECLARE dateCommandeAchat DateTime;
 DECLARE photoDocument Text;
 if(new.idsuivi = 3)THEN
    select nbExemplaire into nbExemplaireLivre from commandedocument where id = new.id;
    SELECT Max(numero) into numeroExemp from exemplaire where id = new.id;
    IF(numeroExemp is null) THEN
        set numeroExemp = 1;
    ELSE
        set numeroExemp = numeroExemp + 1;
    END IF;
    SELECT dateCommande into dateCommandeAchat from commande where id = new.id;
    SELECT image into photoDocument from document where id = new.idLivreDvd;
    WHILE(fin<>nbExemplaireLivre) DO
        INSERT into exemplaire VALUES (new.idLivreDvd, numeroExemp, dateCommandeAchat, photoDocument, '00001');
        set fin = fin + 1;
        set numeroExemp = numeroExemp + 1;
    END WHILE;

 END IF ;
END
&&
```

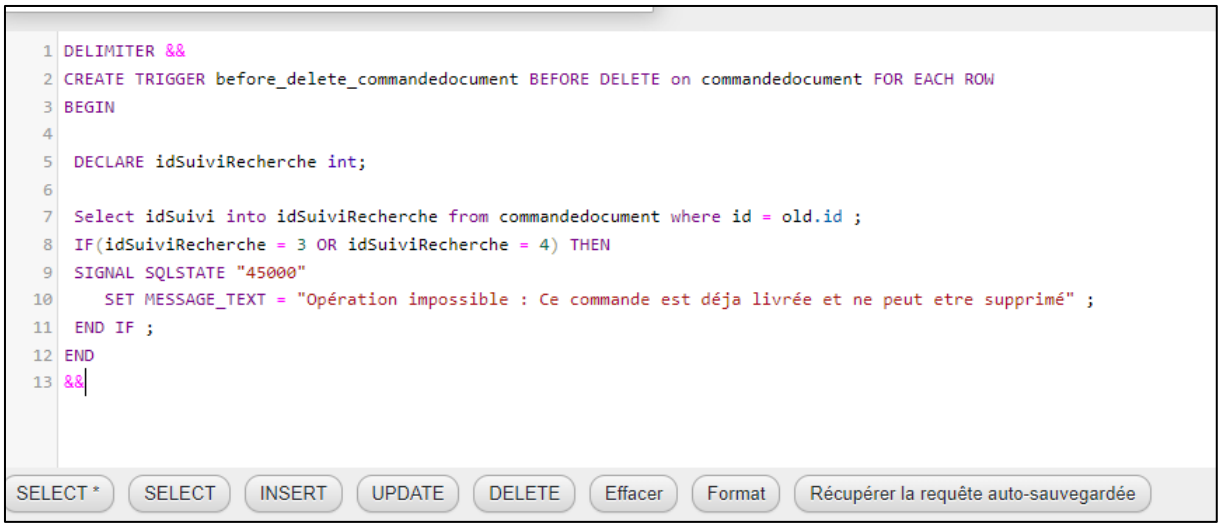
La partie pour gérer les commandes d'un DVD sont identique, l'interface s'ajuste via des contrôles sur le type d'objet lors de l'appel du UserControle

Suppression d'une commande de Livre/DVD

Je commence par créer la méthode dans le dao, cette méthode sera appelée via le contrôleur qui sera sollicité par la vue

```
/// <summary>
/// suppression d'une commande LivreDvd
/// </summary>
/// <param name="commande">commande à supprimer</param>
1 référence
public static bool SupprCommandeLivreDvd(CommandeDoc commande)
{
    try
    {
        string req = "delete from commandedocument where id = @id";
        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@id", commande.Id}
        };
        BddMySql curs = BddMySql.GetInstance(connectionString);
        curs.ReqUpdate(req, parameters);
        curs.Close();
        return true;
    }
    catch (Exception e)
    {
        return false;
    }
}
```

Toutefois, il ne faut pas pouvoir supprimer une commande, si celle-ci est déjà passée à l'état de suivi livré ou réglé. Afin de faire ce contrôle, je crée un trigger dans la BDD.



```
DELIMITER &&
CREATE TRIGGER before_delete_commandedocument BEFORE DELETE on commandedocument FOR EACH ROW
BEGIN

 DECLARE idSuiviRecherche int;

 Select idSuivi into idSuiviRecherche from commandedocument where id = old.id ;
 IF(idSuiviRecherche = 3 OR idSuiviRecherche = 4) THEN
 SIGNAL SQLSTATE "45000"
    SET MESSAGE_TEXT = "Opération impossible : Ce commande est déja livrée et ne peut etre supprimé" ;
 END IF ;
END
&&
```


Ce trigger bloquera la suppression au cas où le suivi ne le permettrait pas.
Également, ici on se concentre sur la suppression d'une commandeDocument mais il faut aussi supprimer la ligne correspondant à cette commande dans la table commande. Encore une fois un trigger sera plus simple à gérer.


```
DELIMITER &&
CREATE TRIGGER AFTER_delete_commandedocument AFTER DELETE on commandedocument FOR EACH ROW
BEGIN
 DELETE FROM commande where id = old.id ;
END
&&
```

Notez qu'il s'agit cette fois d'un trigger AFTER DELETE contrairement au premier qui était un BEFORE DELETE.
Encore une fois la suppression sur un DVD est strictement identique.

Ajout d'un nouvel abonnement sur les revues

Pour gérer les commandes d'abonnement de revue, j'utilise toujours le même UserControle en faisant des contrôles sur le type d'objet.

```
public UsrcCommandeLivre(FrmCommandes frm, object objet)
{
    if(objet is Livre)
    {
        unLivre = (Livre)objet;
        idDocument = unLivre.Id;
    }
    else if (objet is Dvd)
    {
        unDvd = (Dvd)objet;
        idDocument = unDvd.Id;
    }
    else
    {
        uneRevue = (Revue)objet;
        idDocument = uneRevue.Id;
    }
    InitializeComponent();
    frmCommandes = frm;
    Init();
}
```

Également, je crée la nouvelle classe métier Abonnement et commandeAbo qui est enfant de cette dernière.
Le principe est assez similaire que juste avant, seulement cette fois c'est une commandeAbo que je crée et que j'envoie depuis la vue.

```
if (CommandeSelectionneAbo != null)
{

    CommandeAbo commande = new CommandeAbo(ConversionDateBdd(DateTime.Now), double.Parse(txtMontant.Text), CommandeSelectionneAbo.Id,
        ConversionDateBdd(calendrierCommandeLivre.SelectionStart), idDocument);
    frmCommandes.ModifierAjouterCommande(commande, ajoutEnCours);
    btnAnnulerModif_Click(null, null);

}
else
```

Dans le DAO, je crée la nouvelle méthode qui va se charger de créer l'abonnement et la commande correspondante

```
/// <summary>
/// créer une commande de document
/// </summary>
/// <param name="commandeAbo"></param>
/// <returns>true si l'insertion a pu se faire</returns>
1 référence
public static bool CreerCommandeRevue(CommandeAbo commandeAbo)
{
    CommandeDoc commande = new CommandeDoc(commandeAbo.DateCommande, commandeAbo.Montant, commandeAbo.Id, 0, null, 0, null);
    string id = CreerCommande(commande);
    try
    {
        string req = "insert into abonnement values (@id, @dateFinCommande, @idRevue)";
        Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@id", id},
            { "@dateFinCommande", commandeAbo.DateFinAbonnement},
            { "@idRevue", commandeAbo.IdRevue}
        };
        BddMySql curs = BddMySql.GetInstance(connectionString);
        curs.ReqUpdate(req, parameters);
        curs.Close();
        return true;
    }
    catch
    {
        return false;
    }
}
```

Suppression d'un abonnement

La suppression est encore une fois très similaire à ce qui a déjà été fait, la seule différence est qu'elle peut se faire uniquement si aucun exemplaire de revue n'a été récupéré pendant la période qui souhaite être supprimée (date de l'ouverture d'abonnement juste qu'à la date de fin d'abonnement). Pour ce faire, deux méthodes ont été créées, une qui sert à contrôler la présence d'une date entre deux autres, et une autre qui va appeler cette méthode sur une collection d'exemplaires. Le principe est que la collection d'exemplaires est la liste des revues où l'id de la revue est le même que l'id de la revue à qui appartient l'abonnement qu'on souhaite supprimer. La deuxième méthode va appeler la première sur chaque date d'achat de l'exemplaire, si lors des appels, la première méthode retourne vrai, la boucle for s'arrête. La capture sera plus claire

```
public bool verifRevueCommande(List<Exemplaire> lesExemplaires)
{
    foreach(Exemplaire exemp in lesExemplaires)
    {
        if(ParutionDansAbonnement(DateTime.Parse(CommandeSelectionneAbo.DateCommande), DateTime.Parse(CommandeSelectionneAbo.DateFinAbonnement), exemp.DateAchat))
        {
            return true;
        }
    }
    return false;
}
1 référence
public bool ParutionDansAbonnement(DateTime dateCommande, DateTime dateFinAbonnement, DateTime dateParution)
{
    if(dateParution >= dateCommande)
    {
        if(dateParution <= dateFinAbonnement)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
}
```

Cette méthode « verifRevueCommande» est appelée lors d'un if pour valider ou non la suppression. La méthode « ParutionDansAbonnement » est d'ailleurs placée dans un test unitaire.

Message utilisateurs sur les abonnements de moins de 30 jours

Pour ce faire une procédure stockée est créée dans la BDD afin qu'elle fasse le contrôle. Cette procédure va retourner l'id et le titre de toutes les revues à qui il leur reste moins de 30 jours d'abonnement.

```
1 DELIMITER &&
2 CREATE PROCEDURE abonnementMoins30Jours()
3 BEGIN
4 select idRevue, titre, (DateDiff(dateFinAbonnement, Now())) as delai
5 from abonnement inner join document on document.id = abonnement.idRevue
6 HAVING delai < 30;
7 END
8 &&
```

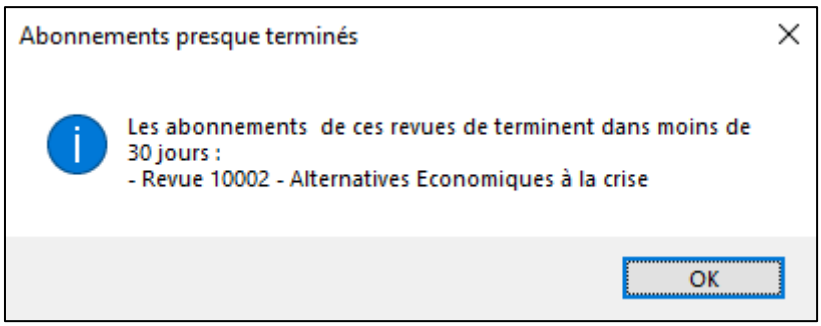
Dans le DAO, cette méthode pour appeler la procédure est appelée

```
/// <summary>
/// Retourne les revues avec un abonnement de moins de 30 jours
/// </summary>
/// <returns>Liste d'objets Revues</returns>
1 référence
public static List<Revue> GetAbonnementMoinsTrenteJours()
{
    List<Revue> lesRevues = new List<Revue>();
    string req = "CALL `abonnementMoins30Jours`()";

    BddMySql curs = BddMySql.GetInstance(connectionString);
    curs.ReqSelect(req, null);

    while (curs.Read())
    {
        string idRevue = (string)curs.Field("idRevue");
        string titre = (string)curs.Field("titre");
        Revue revue = new Revue(idRevue, titre, null, null, null, null, null, null, null, false, null, 0);
        lesRevues.Add(revue);
    }
    curs.Close();

    return lesRevues;
}
```

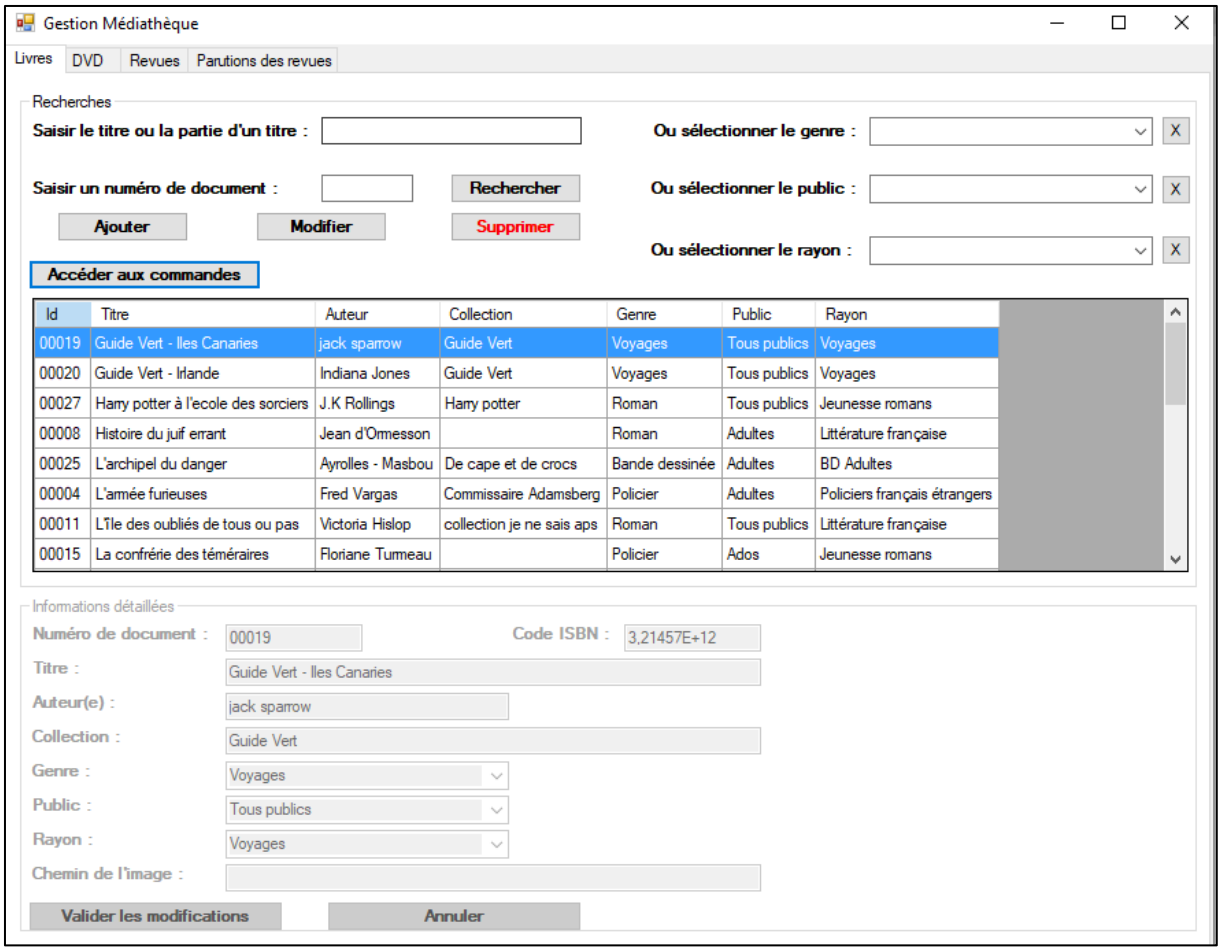
Ensuite, lors du démarrage du formulaire Mediatek86 j'appelle cette méthode via le contrôleur dans une méthode qui affichera un MessageBox si au moins un objet a été retourné.

```
internal FrmMediatek(Controle controle)
{
    InitializeComponent();
    this.controle = controle;
    lesGenres = controle.GetAllGenres();
    lesPublics = controle.GetAllPublics();
    lesRayons = controle.GetAllRayons();
    messageRevuesMoinsTrenteJours(controle.GetAbonnementMoinsTrenteJours());
}


#region modules communs

1 référence
private void messageRevuesMoinsTrenteJours(List<Revue> lesRevues)
{
    if(lesRevues.Count() > 0)
    {
        string message = "Les abonnements  de ces revues de terminent dans moins de 30 jours :\n";
        foreach (Revue revue in lesRevues)
        {
            message += $"- Revue {revue.Id} - {revue.Titre}\n";
        }
        MessageBox.Show(message, "Abonnements presque terminés", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }

}
```

Pour ce résultat lors de l'ouverture



La vue ressemble à ça avec l'ajout du bouton dans chaque onglet



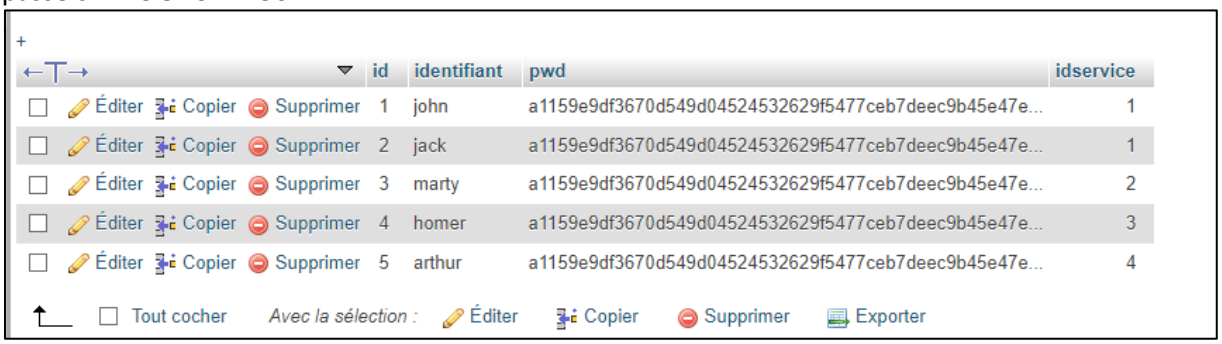
Je nettoie le code, j'ajoute les commentaires normalisés partout et je fais un commit & push vers GitHub.

Mission 3 : gérer le suivi de l'état des documents

Cette mission est facultative et ne sera pas traitée ici

Mission 4 : Mettre en place des authentifications

Je commence par ajouter les deux nouvelles tables dans la base de données, la table utilisateur qui contiendra l'identifiant, le mot de passe et l'id du service, et la table service qui contient les différents services. Je remplis les tables en créant, pour les utilisateurs, des utilisateurs tests avec un mot de passe chiffré en SHA256.



	id	identifiant	pwd	idservice
Éditer Copier Supprimer	1	john	a1159e9df3670d549d04524532629f5477ceb7deec9b45e47e...	1
Éditer Copier Supprimer	2	jack	a1159e9df3670d549d04524532629f5477ceb7deec9b45e47e...	1
Éditer Copier Supprimer	3	marty	a1159e9df3670d549d04524532629f5477ceb7deec9b45e47e...	2
Éditer Copier Supprimer	4	homer	a1159e9df3670d549d04524532629f5477ceb7deec9b45e47e...	3
Éditer Copier Supprimer	5	arthur	a1159e9df3670d549d04524532629f5477ceb7deec9b45e47e...	4

Tout cocher *Avec la sélection :* Éditer Copier Supprimer Exporter

Dans l'application, je crée un nouveau formulaire nommé FrmAuthentification dans lequel je place deux Textbox ainsi qu'un bouton de connexion.
Dans l'événement du clic sur le bouton de connexion, j'appelle la méthode « authentification du contrôleur » (méthode que l'on va créer juste après). Cette méthode retourne le nom du service de l'utilisateur cherchant à se connecter. Si ce service est null, cela signifie que la connexion n'a pas pu se faire et que les informations d'identifications sont erronées, également si celui-ci correspond à « culture», il faut bloquer la connexion puisque ces personnes n'ont pas accès à cette application.

```
public partial class FrmAuthentification : Form
{
    Controle controle;
    string nomService;
    1 référence
    public FrmAuthentification(Controle controle)
    {
        this.controle = controle;
        InitializeComponent();
    }

    1 référence
    private void btnConnexion_Click(object sender, EventArgs e)
    {
        nomService = controle.Authentification(txtIdentifiant.Text, txtPwd.Text);
        if(nomService is null)
        {
            MessageBox.Show("Identifiant ou mot de passe érroné", "Échec de l'authentification", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        else if(nomService.Equals("culture"))
        {
            MessageBox.Show("Vos privilèges ne vous permettent pas d'accéder à cette application", "Échec de l'authentification",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
            controle.OuvertureFrmMediatek(nomService);
        }
    }
}
```

Il ne reste plus qu'à créer la méthode dans le DAO pour permettre l'authentification. Dans le DAO, je fais une simple requête selecte sur le nom du service en fonction des paramètres reçus. Également pour chiffrer le mot de passe en SHA256 pour la comparaison, j'utilise une méthode qui s'en occupe.

```
/// <summary>
/// Retourne le service de l'utilisateur
/// </summary>
/// <returns>service de l'utilisteur</returns>
1 référence
public static string Authetification(string identifiant, string pwd)
{
    string req = "select service.nom " +
        "from utilisateur " +
        "inner join service on service.id = utilisateur.idservice " +
        "where identifiant = @identifiant " +
        "and pwd = @pwd";
    Dictionary<string, object> parameters = new Dictionary<string, object>
        {
            { "@identifiant", identifiant},
            { "@pwd", GetStringSha256Hash(pwd)}
        };
    BddMySql curs = BddMySql.GetInstance(connectionString);
    curs.ReqSelect(req, parameters);

    curs.Read();
    string service = (string)curs.Field("nom");
    curs.Close();

    return service;
}
```

```
/// <summary>
/// Transformation d'une chaîne avec SHA256 (pour le pwd)
/// </summary>
/// <param name="text"></param>
/// <returns></returns>
1 référence
internal static string GetStringSha256Hash(string text)
{
    if (string.IsNullOrEmpty(text))
        return string.Empty;

    using (var sha = new System.Security.Cryptography.SHA256Managed())
    {
        byte[] textData = System.Text.Encoding.UTF8.GetBytes(text);
        byte[] hash = sha.ComputeHash(textData);
        return BitConverter.ToString(hash).Replace("-", string.Empty);
    }
}
#endregion
```

À présent, il ne reste plus qu'à mettre à jour le contrôleur pour ajouter l'appel de cette méthode et également pour qu'il démarre sur le formulaire d'authentification à la place du formulaire Mediatek.

```
private FrmAuthentification frmAuthentification;

/// <summary>
/// Ouverture de la fenêtre
/// </summary>
1 référence
public Controle()
{
    frmAuthentification = new FrmAuthentification(this);
    frmAuthentification.ShowDialog();
}
/// <summary>
/// Permet d'ouvrir le formulaire mediatek
/// </summary>
/// <param name="service">le service de l'utilisateur connecté</param>
1 référence
public void OuvertureFrmMediatek(string service)
{
    FrmMediatek frmMediatek = new FrmMediatek(this, service);
    frmMediatek.ShowDialog();
    frmAuthentification.Dispose();
}
```

Maintenant que l'authentification fonctionne, il faut ajouter les restrictions sur les fenêtres. On sait que le service culture est bloqué dès la connexion et que les services administration et administrateur ont accès à toute l'application. Il ne reste que le service prêt qui ne doit pouvoir que consulter et pas pouvoir modifier les livres, dvd et revues.
Dans le FrmMediatek, j'effectue quelques réglages, comme le fait que pour le service prêt il n'y a pas le message lors de l'ouverture sur les abonnements de moins de 30 jours, ainsi qu'à chaque ouverture d'onglet, je bloque les boutons de modification, de suppression et d'accès aux commandes.

```
/// <summary>
/// Ouverture de l'onglet Revues :
/// appel des méthodes pour remplir le datagrid des revues et des combos (genre, rayon, public)
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void tabRevues_Enter(object sender, EventArgs e)
{
    lesRevues = controle.GetAllRevues();
    RemplirComboCategorie(lesGenres, bdgGenres, cbxRevuesGenres);
    RemplirComboCategorie(lesGenres, bdgGenresChoix, comboRevueGenre);
    RemplirComboCategorie(lesPublics, bdgPublics, cbxRevuesPublics);
    RemplirComboCategorie(lesPublics, bdgPublicsChoix, comboRevuePublic);
    RemplirComboCategorie(lesRayons, bdgRayons, cbxRevuesRayons);
    RemplirComboCategorie(lesRayons, bdgRayonsChoix, comboRevueRayon);
    RemplirRevuesListeComplete();
    if (service.Equals("prêt"))
    {
        btnAccederCommandeRevue.Enabled = false;
        btnModifierRevue.Enabled = false;
        btnSupprRevues.Enabled = false;
    }
}
```

Voici le résultat de la vue pour l'authentification

La mission 4 est terminé, je commit & push.

Mission 5 et 6 : Préparation du projet pour la mise à disposition

Nettoyage du code :

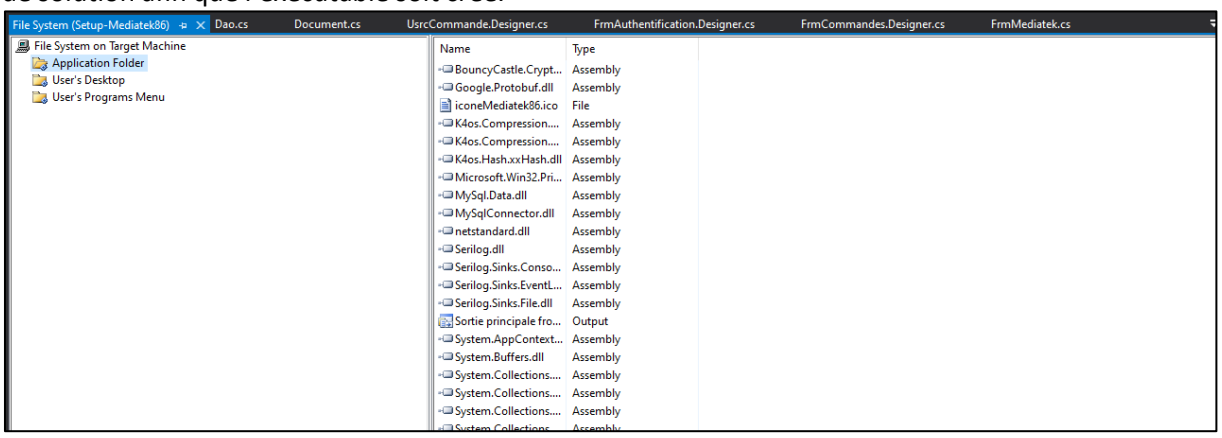
Dans une optique d'optimisation du code et de le rendre le plus propre possible, j'utilise SonarQube dans Visual Studio afin de m'aider dans la recherche d'anomalie. Également dans les propriétés du projet j'active la création de la documentation technique afin qu'également que toutes les propriétés ainsi que les méthodes où j'ai oublié de mettre les commentaires normalisés apparaissent en warning. Après avoir nettoyé le code, je crée la documentation technique.

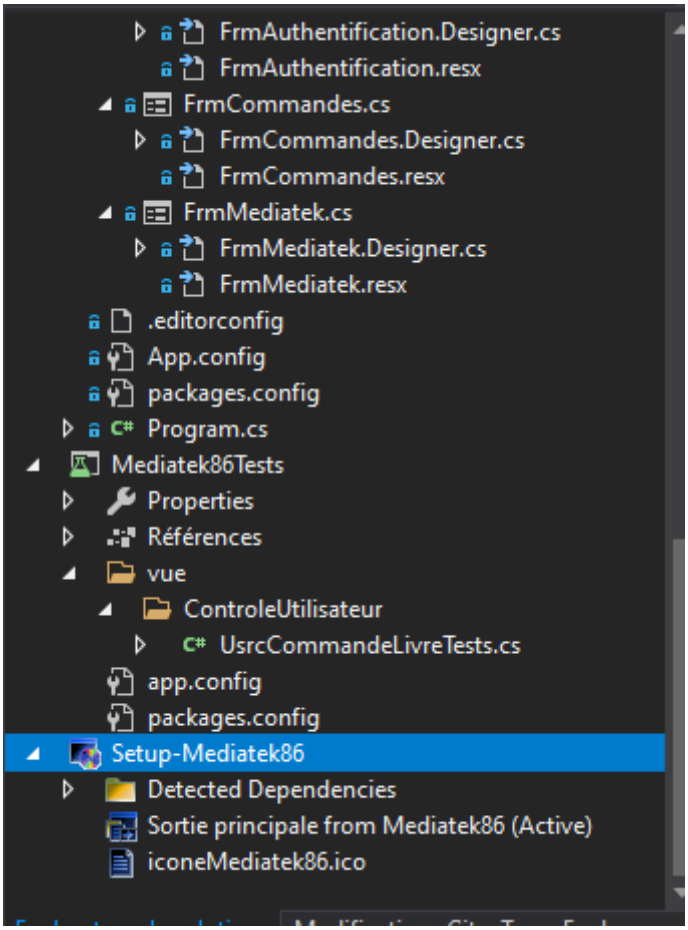
Exportation de la BDD :

Pour l'exportation de la base de données, je vais utiliser les services d'hébergement d'Amazon AWS. Je récupère le scripte dans la base de données local, puis je l'importe dans le serveur AWS. J'ai dû modifier dans le scripte le créateur des triggers et des procédures pour retirer « root » et mettre « admin » (profile utiliser sur AWS). Également j'ai modifié les propriétés du serveur pour qu'il accepte les triggers (désactivé par défaut chez Amazon). Je modifie les identifiants de la chaîne de connexion pour que l'application passe sur la base distante.

Préparation de l'exécutable :

Afin de préparer l'exécutable, j'ajoute un nouveau projet dans la solution. Il s'agit de setup wizard. Il va me permettre de créer l'exécutable en ajustant les paramètres qu'il m'offre. J'ai créé également une image au format .ico en 128x128 que j'ajoute au projet. Enfin je fais « générer » dans l'explorateur de solution afin que l'exécutable soit créé.





Conclusion

L'application est à présent fonctionnelle avec une base de données distante et un fichier exécutable pour l'installation. Le développement c'est fait dans le respect de la structure MVC et de la charte graphique déjà existante.
L'application peut maintenant ajouter, modifier et supprimer un document seulement si l'utilisateur connecté possède les droits. Également, il est possible de suivre les commandes d'un document, il est possible d'en créer une, de la modifier et de la supprimer.
Afin de contrôler que toutes ces modifications de données n'entraînent pas des incohérences dans les contraintes d'intégrités de la base de données, des triggers et des procédures stockées ont été mis en place pour gérer ces éventualités ainsi que de faciliter le traitement de certaines actions permettant d'alléger le développement de l'application.