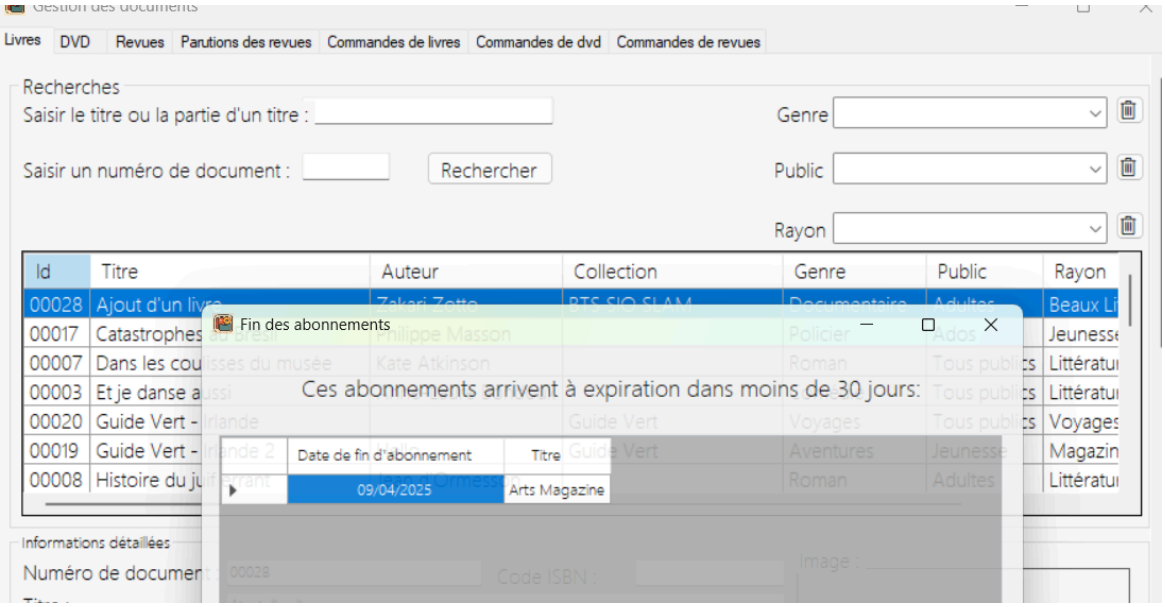



# MediaTekDocuments
# Rapport

*Atelier professionnel 2 - BTS SIO 2e année*

Paul Thorel

## Introduction/Contexte

Dans le cadre de mon BTS Services Informatiques aux Organisations (SIO), option Solutions Logicielles et Applications Métier (SLAM), j’ai effectué dans le cadre de ma deuxième année, des ateliers professionnels visant à évaluer mes compétences acquises.

Ici, il fallait reprendre le travail existant d'un développeur de "l'équipe" sur une application lourde développée en C# et utilisant une API développée en PHP, en rajoutant des fonctionnalités, et en sécurisant certains aspects.

## Ressources et outils mis en œuvre

Durant cet atelier, j'ai utilisé:

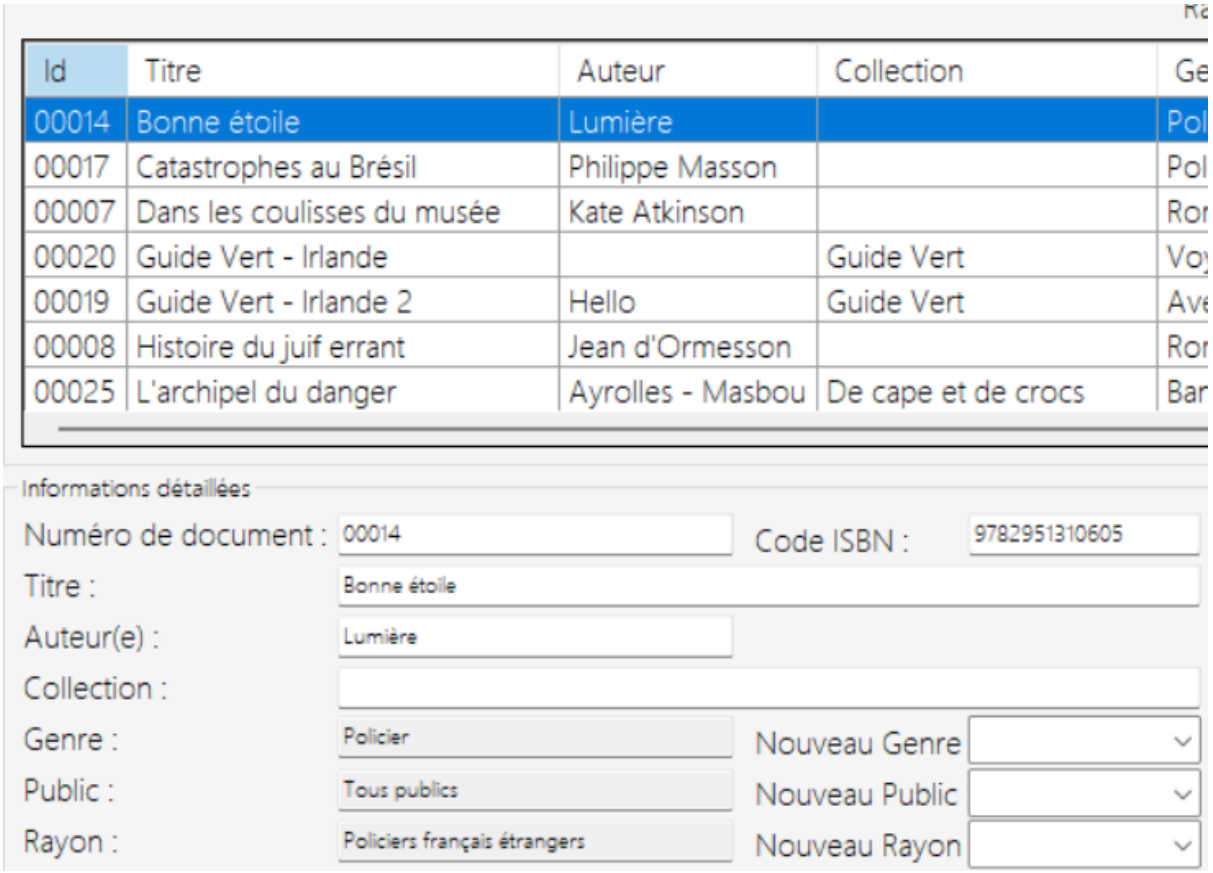
- PhpMyAdmin tout au long du développement, afin de vérifier la bonne modification et l'intégrité des données
- Visual studio 2022: développement en C# - débogage et développement
- Stackedit ainsi que google Docs pour rédiger/éditer les readmes et ce rapport
- Postman afin de tester et adapter l’api
- Et enfin sublime text pour écrire rapidement du code à la volée

## Mission 1 - Gérer les documents

La gestion des documents est le point crucial de l'application.

J'ai commencé par modifier les vues (contenant les fonctions pour les missions suivantes afin de ne pas revenir sur cette partie à chaque fois).

Ici une partie de la vue pour les livres.




Certaines fonctions du controlleur principal sont également rajoutées (ici la fonction qui gère l'événement utilisateur de modification d'un livre)

(suppression n'étant possible que si aucun exemplaire existe, bien sur)

```
private void btnReceptionLivreModifier_Click(object sender, EventArgs e)
{
    if (dgvLivresListe.SelectedRows.Count > 0)
    {
        Livre selectedLivre = (Livre)dgvLivresListe.SelectedRows[0].DataBoundItem;

        string id = selectedLivre.Id;
        string titre = txbLivresTitre.Text;
        string image = txbLivresImage.Text;
        string isbn = txbLivresIsbn.Text;
        string auteur = txbLivresAuteur.Text;
        string collection = txbLivresCollection.Text;
        string idGenre = GetIdGenreDocument(cbxAjoutModifGenreLivre.Text);
        string idPublic = GetIdPublicDocument(cbxAjoutModifPublicLivre.Text);
        string idRayon = GetIdRayonDocument(cbxAjoutModifRayonLivre.Text);

        if (!txbLivresNumero.Text.Equals("") && !txbLivresTitre.Text.Equals("") && !cbxAjoutModifGenreLi
        {
            if (controller.ModifierLivre(id, isbn, auteur, collection) && controller.ModifierDocument(ic
            {
                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.");
            }
```

Ainsi que la méthode qui s'occupe des appels à l'api pour "enregistrer" ces données:

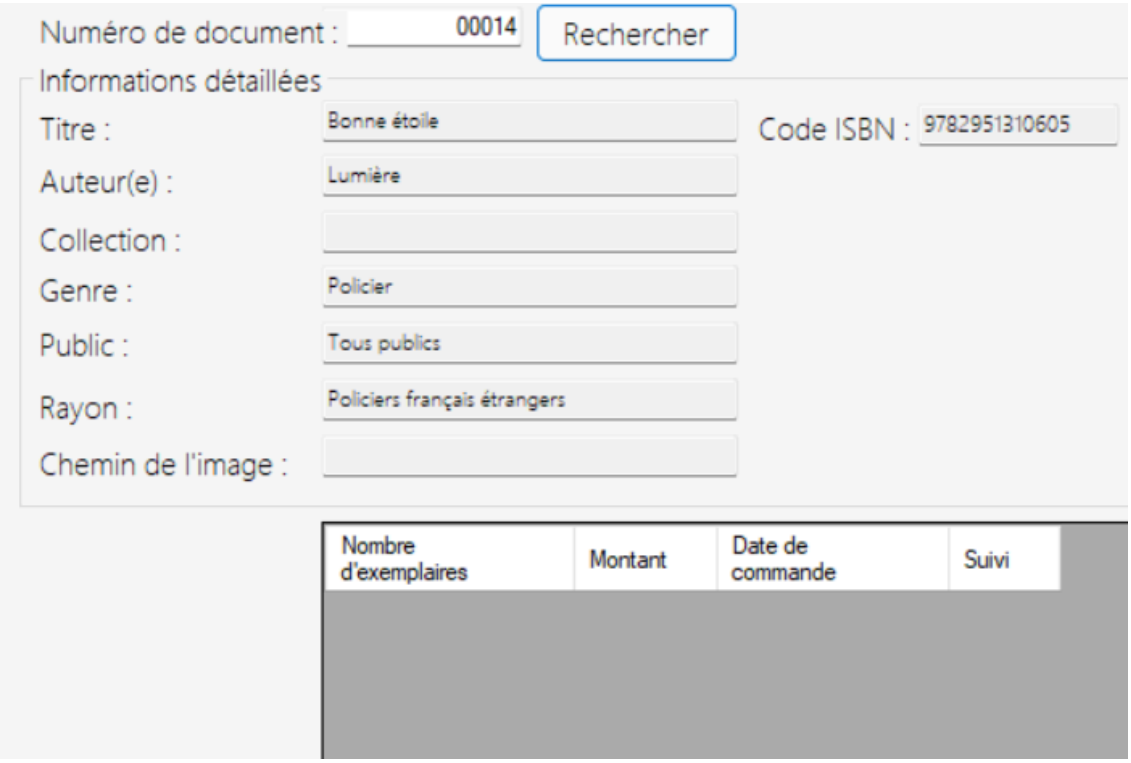
```
1 référence
public bool ModifierLivre(string Id, string Isbn, string Auteur, string Collection)
{
    String jsonModifierLivre = "{ \"id\" : \"" + Id + "\", \"isbn\" : \"" + Isbn + "\", \"auteur\" : \"" + Auteur + "\", \"
    Console.WriteLine("jsonModifierLivre" + jsonModifierLivre);
    try
    {
        // récupération soit d'une liste vide (requête ok) soit de null (erreur)
        List<Livre> liste = TraitementRecup<Livre>(PUT, "livre/" + Id + "/" + jsonModifierLivre);
        return (liste != null);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Log.Error("Access.ModifierLivre catch jsonModifierLivre={0} erreur={1} ", jsonModifierLivre, ex.Message);
    }
    return false;
}
```

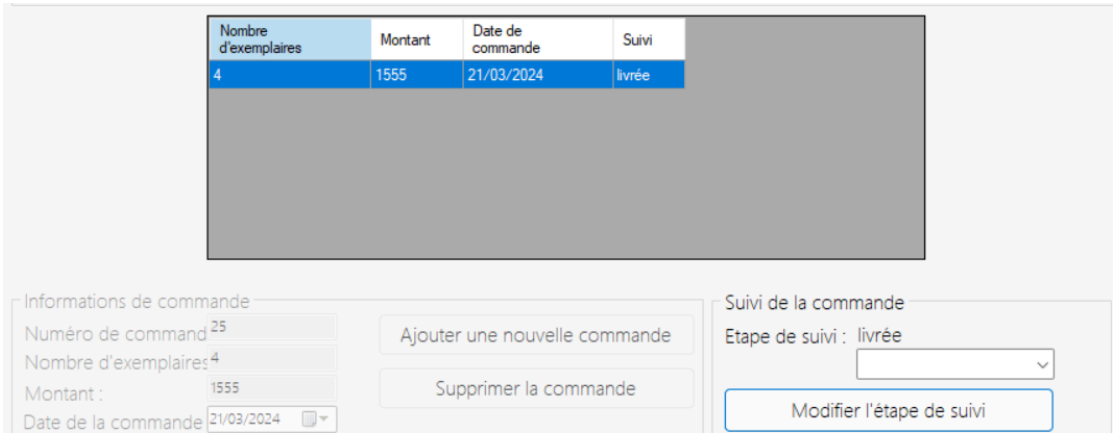
L'utilisation de postman m'a permis ici de vérifier que les modifications fonctionnaient bien, ainsi que de la bonne utilisation des verbes HTTP.
Enfin, il faut également modifier l'api pour gérer ces fonctionnalité (en PHP du coup):

```
                case "dvd" :
                    return $this->selectAllDvd();
                case "revue" :
                    return $this->selectAllRevues();
```

## Mission 2 - Gérer les commandes

Ensuite, il faut rajouter la possibilité de gérer les commandes de documents
J'ai notamment créé des nouveaux éléments sur la vue qui permettent de saisir les informations d'une nouvelle commande et de l'enregistrer.




Et on peut également modifier le suivi d'une commande.

On doit donc également l'appel aux éventuelles interactions sur ces valeurs dans le controlleur et sur les vues:

```
private void btnReceptionCommandeLivreValider_Click(object sender, EventArgs e)
{
    if (!txbCommandeLivreNumero.Text.Equals("") && !txbCommandeLivreNbExemplaires.Text.E
    {
        string id = txbCommandeLivreNumero.Text;
        int nbExemplaire = int.Parse(txbCommandeLivreNbExemplaires.Text);
        double montant = double.Parse(txbCommandeLivreMontant.Text);
        DateTime dateCommande = dtpCommandeLivre.Value;
        string idLivreDvd = txbCommandesLivresNumRecherche.Text;
        string idSuivi = lesSuivis[0].Id;

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

        var idCommandeLivreExistante = controller.GetCommandes(id);
        var idCommandeLivreNonExistante = !idCommandeLivreExistante.Any();
```

```
public bool CreerCommandeDocument(string id, int nbExemplaire, string idLivreDvd, string idSuivi)
{
    return access.CreerCommandeDocument(id, nbExemplaire, idLivreDvd, idSuivi);
}
```

Tout en modifiant l'api de communication avec la bdd

```
public bool CreerCommandeDocument(string id, int nbExemplaire, string idLivreDvd, string idSuivi)
{
    String jsonCreerCommandeDocument = "{ \"id\" : \"" + id + "\", \"nbExemplaire\" : \"" + nbExempl
    Console.WriteLine("jsonCreerCommandeDocument" + jsonCreerCommandeDocument);
    try
    {
        // récupération soit d'une liste vide (requête ok) soit de null (erreur)
        List<CommandeDocument> liste = TraitementRecup<CommandeDocument>(POST, "commandedocument/" +
        return (liste != null);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Log.Error("Access.CreerCommandeDocument catch jsonCreerCommandeDocument={0} erreur={1} ", js
    }
    return false;
```

Et enfin le(s) modèle(s), qui sont pour certains créés ou modifiés (tout au long du développement, pas juste sur cette étape)

```
using System;

namespace MediaTekDocuments.model
{
    9 références
    public class Suivi
    {
        4 références
        public string Id { get; set; }

        2 références
        public string Libelle { get; set; }

        0 références
        public Suivi(string id, string libelle)
        {
            this.Id = id;
            this.Libelle = libelle;
        }
    }
```

```
public class Commande
{

    public string Id { get; set; }


    public DateTime DateCommande { get; set; }


    public double Montant { get; set; }


    public Commande(string id, DateTime dateCommande, double montant)
    {
        this.Id = id;
        this.DateCommande = dateCommande;
        this.Montant = montant;
    }
}
```

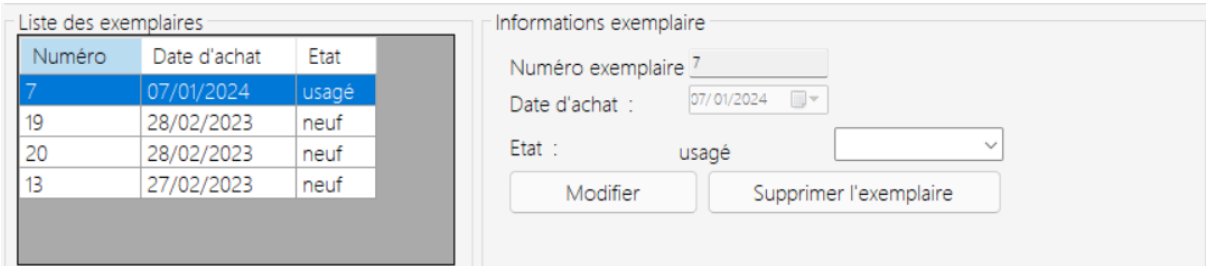
## Mission 3 - Gérer l’état des documents

Ensuite, il fallait rajouter la possibilité de suivre et de gérer l’état des documents (“onglet” déjà visible dans les captures d’écrans des vues)

On doit pouvoir accéder (et j’ai donc créé des variables dans la classe du controlleur contenant l’id du document, de manière dynamique sur chaque click/changement:

```
private void affichageExemplairesLivreClick(object sender = null, DataGridViewCellEventArgs e = null)
{
    int rowIndex;

    if (e != null)
    {
        rowIndex = e.RowIndex;
    }
    else if(dgvLivresListe.Rows.Count > 0)
    {
        rowIndex = 0;
    }
    else {

        return;
    }


    if (rowIndex >= 0)
    {
        // Récupérer la valeur de la "troisième" colonne (colonne ID)
        string id = (string)dgvLivresListe.Rows[rowIndex].Cells[3].Value;

        if (id != null)
        {
            this.INDEXLIVRE = id;
            AfficheExemplairesLivres(id);
        }
```

Ces variables permettent d’automatiquement remplir le “tableau” contenant les exemplaires et leur état, dans certains cas.




En plus du controlleur et de l'application C# elle même, il faut aussi modifier l'API afin d'intéragir avec ces données.

```php
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);
}
```

## Mission 4 - Authentification et sécurisation

On doit maintenant pouvoir sécuriser l'application

J'ai décidé de stocker directement les identifiants et les mots de passe, ainsi que les "autorisations" dans la BDD

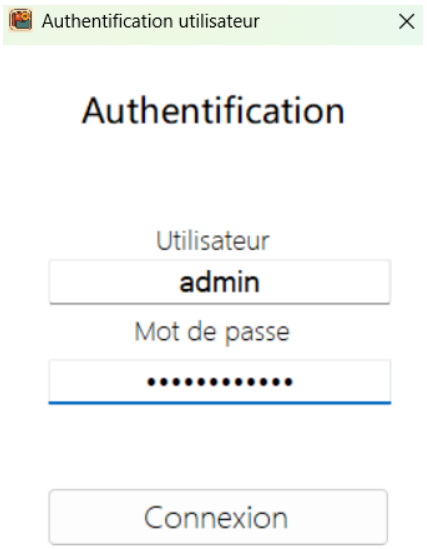
Les mots de passe sont stocké de manière cryptée (de cette manière) , et l'api permet de récupérer ces données comme pour les autres tables (pour l'instant), qu'on vérifiera ensuite durant la connexion

```sql
INSERT INTO `utilisateur`(`id`, `login`, `password`, `idService`) VALUES ('00002','admin',SHA2("1adminpass!",256),'1')
INSERT INTO `utilisateur`(`id`, `login`, `password`, `idService`) VALUES ('00001','pret',SHA2("paul",256),'3')
INSERT INTO `utilisateur`(`id`, `login`, `password`, `idService`) VALUES ('00003','nocompte',SHA2("mdp",256),'3')
```

Il faut également créer le modèle qui représente notre “utilisateur”:

```
public Utilisateur(string login, string password, string idService
{
    this.Login = login;
    this.Password = password;
    this.IdService = idService;
    this.Libelle = libelle;
}
```

Ensuite, j’ai créé le formulaire d’authentification:




Lorsque l’utilisateur se connecte, on vérifie son service/ses habilitations, et la vue est modifiée en conséquence, suivant que l’utilisateur soit administrateur ou non:

```
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;
```

C’est sans doute la partie qui m’a le plus donné du fil à retordre, et j’ai du regarder la documentation (de mysql ainsi que dotnet ), à plusieurs reprises, pour réussir à obtenir le même résultat de “cryptage” entre les deux

## Mission 6 - Documentation textuelle et video

J'ai enregistré et monté une vidéo de démonstration de l'application, ainsi que la page de portfolio, disponible à cette adresse:

https://github.com/Manerr/mediatekdocuments/

manerr.github.io/mediatekdocuments

## Mission 7 - déploiement

Afin de déployer l'application, il suffit de copier la source de l'application rest sur un serveur disposant de wampserver (ou de php et mysql).
Il faut exécuter composer install dans le répertoire racine du dossier, pour installer les dépendances.
Ensuite, il suffit juste d'importer le script fourni (au format sql) dans phymyadmin ou adminer, et l'api est prête à fonctionner.

Pour utiliser l'application cliente, il suffit d'exécuter le fichier *.exe fournis, qui installera l'application directement, avec tous les fichiers/dépendances requises.

Je n'ai pas pu faire fonctionner l'hébergement avec azure, comme pour le premier atelier, les outils proposés par github action copiant bien les fichiers sur le "serveur", mais certains d'entre eux étant supprimés au passage…

## Compétences acquises:

Lors de cet atelier, j'ai eu l'occasion d'approfondir mes connaissances, que ce soit en PHP ou en C#, en utilisation/modification d'un API et tous les tests que cela implique (avec postman, que j'ai particulièrement apprécié pour sa très grande flexibilité et sa belle interface 💔 ) et surtout en programmation objet, avec le concept de modèle que j'ai eu l'occasion d'approfondir.

## Conclusion:

Cet atelier m'a apporté une vision plus professionnelle dans le cadre du développement d'applications, et plus largement m'a permis de m'enrichir dans plusieurs domaines, que ce soit en utilisant plusieurs langages de programmation, par exemple