

# Compte-rendu de projet professionnel.

Contexte :

Dans ce projet, en tant que seul développeur attitré, on me demande de faire évoluer une application C# liée à une API REST PHP. Plusieurs fonctionnalités manquantes sont décrites dans le cahier des charges et doivent être implémentées : la gestion des commandes de tous types de documents, l'abonnement aux revues, ainsi que l'authentification des utilisateurs. Certains aspects doivent également être corrigés par mesure de sécurité.

Technologies et logiciels utilisés :

- .NET Framework 4.7.2
- Windows Forms
- Visual Studio 2022
- Visual Studio Code
- Apache2
- PHP 8
- MySQL
- FileZilla
- Postman
- Pencil
- Looping

Dépôt du client de gestion C# : <https://github.com/DevBlocks42/MediatekDocuments>

Dépôt de l'API PHP : [https://github.com/DevBlocks42/REST\\_MediatekDocuments](https://github.com/DevBlocks42/REST_MediatekDocuments)

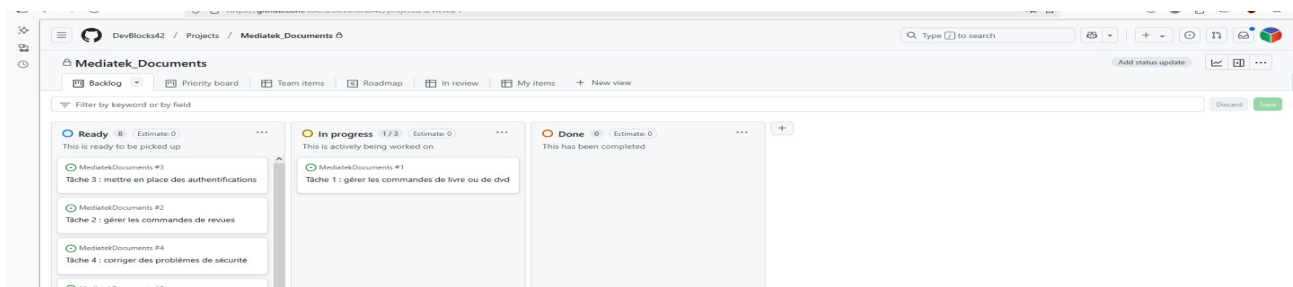
## Table des matières

Compte-rendu de projet professionnel.....	1
Tâche n°1 : gérer les commandes de livre ou dvd.....	3
Aperçu du kanban de la tâche en cours .....	3
Diagramme des cas de la tâche à réaliser.....	4
Modélisation conceptuelle des données.....	5
Prototype de la vue concernant les commandes.....	6
Implémentation du modèle.....	6
Recherche et affichage des commandes de Livre ou Dvd.....	7
Enregistrement de commandes.....	11
Modification du suivi d'une commande.....	14
Suppression de commande.....	18
Création automatisée des exemplaires de documents.....	19
Tâche n°2 : gérer les abonnements.....	19

Aperçu du kanban de la tâche en cours.....	20
Maquette de l'interface d'abonnement.....	21
Diagramme des cas d'utilisations de la tâche.....	21
Ajout de l'objet Abonnement dans le modèle.....	22
Recherche d'une revue et des abonnements associés.....	23
Enregistrement d'un nouvel abonnement.....	25
Suppression d'abonnement.....	27
Rappel des abonnements qui expirent dans moins de 30 jours.....	29
Tâche n°3 : mise en place des authentifications.....	31
Aperçu du kanban de la tâche en cours.....	31
Modèle logique de données concernant l'ajout d'authentification.....	32
Ajout d'entité dans le modèle.....	32
Classe auxiliaire de sécurisation des mots de passes utilisateurs (CryptoTools.cs).....	33
Création d'une nouvelle vue pour la fenêtre d'authentification.....	36
Restriction des Controls visibles à l'utilisateur en fonction de son service.....	38
Tâche n°4 : corriger des problèmes de sécurité.....	39
Aperçu du kanban de la tâche en cours.....	39
Issues et Pull Requests liés aux problèmes identifiés.....	40
Mise en place des correctifs.....	42
Tâche 5 : contrôler la qualité de code.....	43
Aperçu du kanban de la tâche en cours.....	43
Règles mise en évidence par SonarQube.....	44
Tâche n°6 : intégrer les logs.....	45
Kanban de la tâche en cours.....	45
Classe auxiliaire pour la gestion des logs.....	46
Journalisation des affichages console de la classe Access.....	47
Tâche n°7 : gérer les tests.....	48
Kanban de la tâche en cours.....	48
Tests unitaires sur le modèle.....	49
Tests d'intégrations sur les fonctionnalités de recherche.....	50
Collection de tests sur l'API via Postman.....	52
Plan de test intégral.....	54
Tâche n°8 : générer les documentations techniques.....	54
Aperçu du kanban de la tâche en cours.....	55
Documentation technique de l'application C#.....	55
Documentation technique pour l'API PHP.....	57
Tâche n°9 : Déployer le projet.....	58
Aperçu du kanban de la tâche en cours.....	59
Déploiement de l'API et de la base de données.....	59
Déploiement du client C#.....	60
Aperçu des étapes d'installation du logiciel.....	61
Tâche n°10 : automatiser la sauvegarde de base de données.....	64
Limitations techniques liées aux paramètres de l'hébergeur.....	65
Bilan Final.....	66

# Tâche n°1 : gérer les commandes de livre ou dvd.

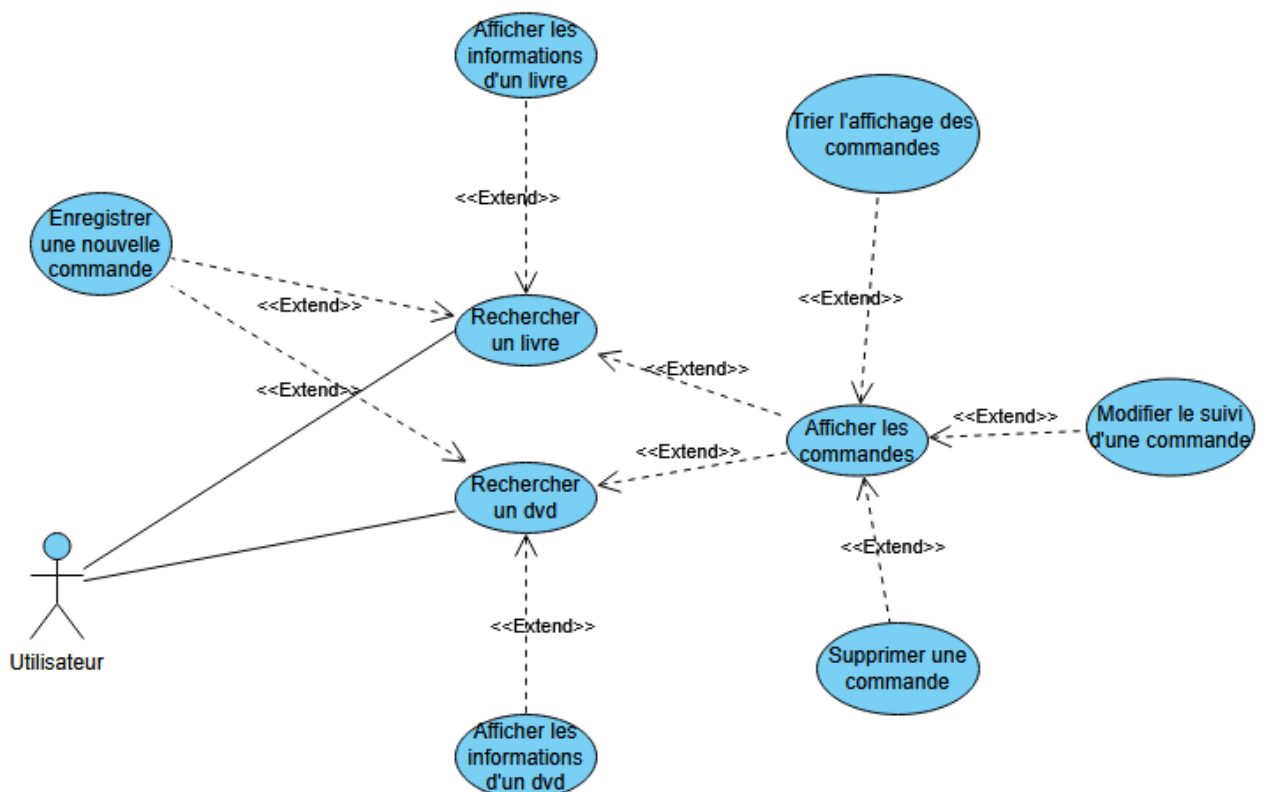
## Aperçu du kanban de la tâche en cours



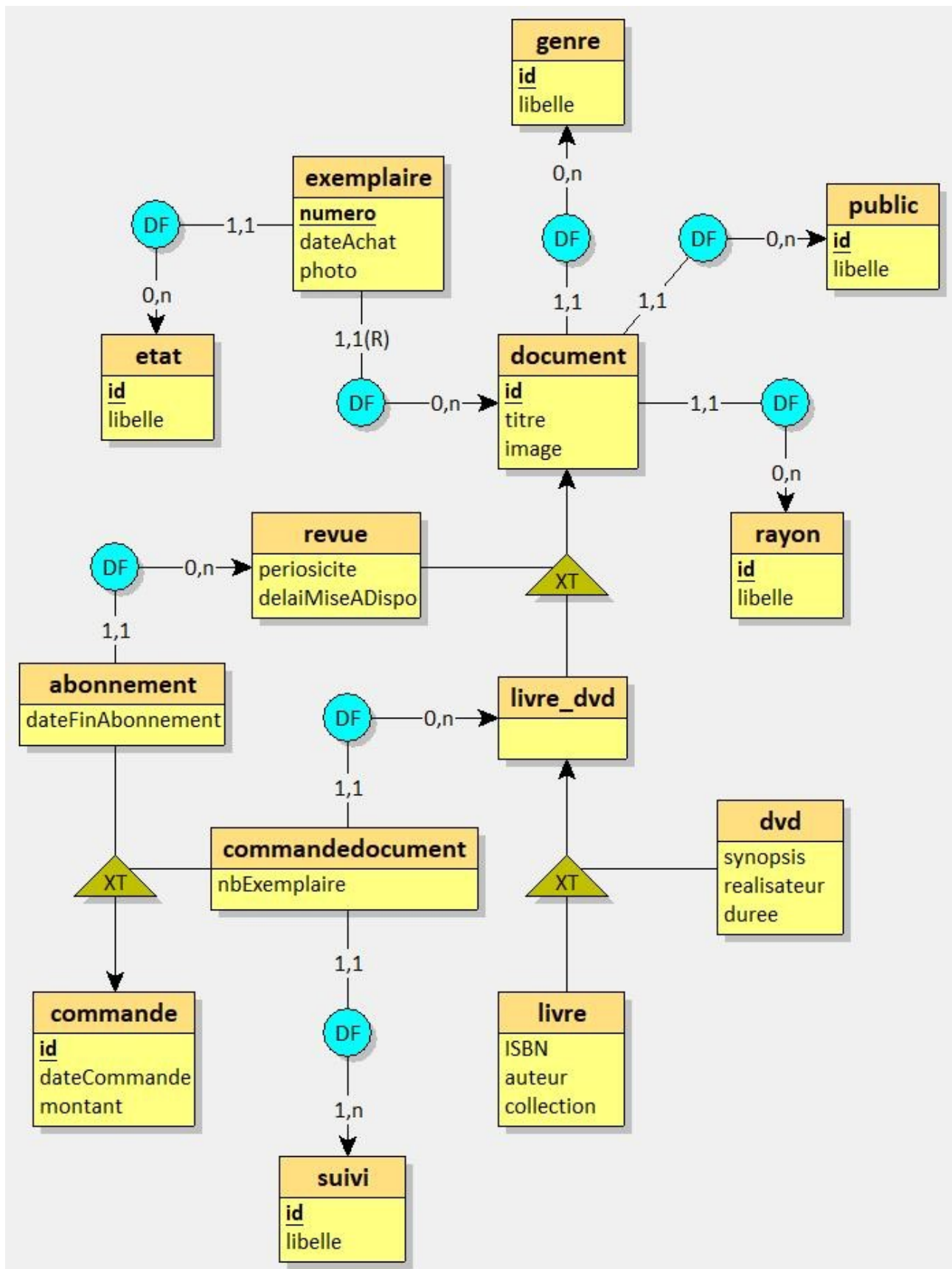
Temps estimé : 8h

Temps effectif : 19h

## Diagramme des cas de la tâche à réaliser



## Modélisation conceptuelle des données



# Prototype de la vue concernant les commandes

Gestion des documents de la médiathèque

Livres

Dvd

Revues

Parution des revues

Commandes livre

Commande DVD

Recherche

Saisir un numéro de livre :

	nbExemplaire	dateCommande	montant	suivi
	10	21/11/1999	12.98	en cours
	100	21/11/1999	9.99	livrée

Liste des commandes pour le livre  
00017

Nouvelle commande

Nombre d'exemplaires

Montant

Modifier le suivi

Numéro de commande

Etat du suivi

Supprimer la commande

Numéro de commande

Informations détaillées

Numéro de document :

Code ISBN :

Titre :

Auteur :

Collection :

Genre :

Public :

Rayon :

Champ en lecture seule, pré-rempli lors de la sélection d'une commande

## Implémentation du modèle

Le modèle n'étant pas totalement implémenté, l'ajout de certains objets est nécessaire pour gérer les commandes de documents.

```
public class Commande
{
    public string id { get; }
    public DateTime dateCommande { get; }
    public double montant { get; }
    public Commande(string id, DateTime dateCommande, double montant)
    {
        this.id = id;
        this.dateCommande = dateCommande;
        this.montant = montant;
    }
}
```

```

public class CommandeDocument : Commande
{
    public string idLivreDvd { get; }
    public int idSuivi { get; }
    public int nbExemplaire { get; }

    public CommandeDocument(string id, DateTime dateCommande, double montant,
string idLivreDvd, int idSuivi, int nbExemplaire) : base(id, dateCommande, montant)
    {
        this.idLivreDvd = idLivreDvd;
        this.idSuivi = idSuivi;
        this.nbExemplaire = nbExemplaire;
    }
}

```

Au niveau conceptuel, on remarque le lien entre commande et commandedocument ainsi que commandedocument et suivi.

Implémentation du Suivi :

```

public class Suivi
{
    public int id { get; }
    public string libelle { get; }

    public Suivi(int id, string libelle)
    {
        this.id = id;
        this.libelle = libelle;
    }
    public override string ToString()
    {
        return libelle;
    }
}

```

L'override ToString permet de faciliter l'affichage des suivi dans les contrôles Windows Forms.

## **Recherche et affichage des commandes de Livre ou Dvd**

Une *TextBox* permet, dans les deux onglets, de saisir le numéro d'un document, selon si on se trouve dans l'onglet de commande de livre ou de dvd, la recherche attendra un numéro de livre, ou de dvd. En d'autres termes, il est impossible de saisir un numéro de dvd dans l'onglet des livres et réciproquement. Dans les deux onglets, on associe le clic sur le bouton de recherche à la même méthode événementielle qui gère les deux cas possibles (recherche d'un livre, ou recherche d'un dvd).

```

private void button1_Click(object sender, EventArgs e)
{
    lesCommandes = null;
}

```

```

livreCourant = null;
LivreDvd livreDvd = null;
Button _sender = sender as Button;
if (_sender.Name == "btnRechercheDvd") { //Cas du dvd
    if (tbxNumDvd.Text.Length != 5) {
        MessageBox.Show("Erreur, l'identifiant saisi ne respecte pas le format (5 chiffres obligatoires).");
        return;
    }
    livreDvd = lesDvd.Find(x => x.Id.Equals(tbxNumDvd.Text)) as LivreDvd;
} else { //Cas du livre
    if (tbxNumLivre.Text.Length != 5) {
        MessageBox.Show("Erreur, l'identifiant saisi ne respecte pas le format (5 chiffres obligatoires).");
        return;
    }
    livreDvd = lesLivres.Find(x => x.Id.Equals(tbxNumLivre.Text)) as LivreDvd;
}
if (livreDvd != null) {
    AfficheLivresInfosCommandes(livreDvd);
    AfficherInfosCommandes(livreDvd);
} else {
    MessageBox.Show("Erreur, le document n'a pas été trouvé.");
}
}

```

En castant le paramètre *sender* en tant que *Button*, on distingue le cas dans lequel nous sommes. En effet, si le bouton qui déclenche l'appel de cette méthode porte le nom « btnRechercheDvd » on devra aller chercher le *Dvd* de numéro correspondant dans la liste *lesDvd* alors que dans le cas contraire, on ira chercher un *Livre* dans la liste *lesLivres*.

Les classes métiers *Livre* et *Dvd* héritent toutes les deux de *LivreDvd*, ainsi il est possible de transtyper *LivreDvd* en *Dvd* ou *Livre* pour savoir de quel type de document il s'agit. Ce transtypage sera utile pour gérer les tâches concernant les livres et les dvd sans écrire de méthodes redondantes.

A la fin de cette méthode, on appelle *AfficheLivresInfosCommandes* et *AfficherInfosCommandes* avec comme paramètre un *LivreDvd* correspondant au document actuellement recherché.

```

private void AfficheLivresInfosCommandes(LivreDvd livre)
{
    if(livre.GetType() == typeof(Livre)) {
        tbxAuteur.Text = ((Livre)livre).Auteur;
        tbxCollection.Text = ((Livre)livre).Collection;
        tbxNumISBN.Text = ((Livre)livre).Isbn;
        tbxNumDoc.Text = livre.Id;
        tbxGenre.Text = livre.Genre;
        tbxPublic.Text = livre.Public;
        tbxRayon.Text = livre.Rayon;
        tbxTitre.Text = livre.Titre;
    } else if(livre.GetType() == typeof(Dvd)) {
        tbxRealisateur.Text = ((Dvd)livre).Realisateur;
        tbxNumDvdInfos.Text = livre.Id;
        tbxTitreDvd.Text = livre.Titre;
        tbxGenreDvd.Text = livre.Genre;
        tbxPublicDvd.Text = livre.Public;
        tbxRayonDvd.Text = livre.Rayon;
        tbxDuree.Text = ((Dvd)livre).Duree.ToString();
    }
}

```

```
}  
}
```

Le but de la méthode ci-dessus *AfficherLivresInfosCommandes* est de remplir les *TextBox* avec les informations de document, qu'il s'agisse d'un *Dvd* ou d'un *Livre*, la méthode sera capable de gérer les deux cas. La différence majeure entre ces deux cas est le nom des contrôles utilisés ainsi que certaines propriétés présentes ou absentes (exemple : un livre possède un auteur alors qu'un dvd possède un réalisateur).

```
private void AfficherInfosCommandes(LivreDvd livre)  
{  
    DataGridView dgvCourante = null;  
    TextBox tbxNumCommandeSuiviCourant = null;  
    TextBox tbxNumCommandeSupprCourant = null;  
    if (livre.GetType() == typeof(Livre)) {  
        dgvCourante = dgvCommandes;  
        tbxNumCommandeSuiviCourant = tbxNumCommandeSuivi;  
        tbxNumCommandeSupprCourant = tbxNumCommandeSuppr;  
    } else {  
        dgvCourante = dgvCommandesDvd;  
        tbxNumCommandeSuiviCourant = tbxNumCommandeDvd;  
        tbxNumCommandeSupprCourant = tbxNumCommandeDvdSuppr;  
    }  
    if (livreCourant == null) {  
        this.livreCourant = livre;  
    }  
    dgvCourante.Columns.Clear();  
    dgvCourante.Rows.Clear();  
    if (lesCommandes == null) {  
        this.lesCommandes = this.controller.getCommandesLivre(livre.Id);  
    }  
    bdgCommandes.DataSource = this.lesCommandes;  
    dgvCourante.DataSource = bdgCommandes;  
    dgvCourante.Columns.Add("idSuivi_", "suivi");  
    for (int i = 0; i < this.lesCommandes.Count; i++) {  
        dgvCourante.Rows[i].Cells["idSuivi_"].Value =  
lesSuivis[this.lesCommandes[i].idSuivi - 1].libelle;  
    }  
    dgvCourante.Columns["idSuivi_"].Visible = false;  
    dgvCourante.Columns["id"].Visible = false;  
    dgvCourante.Columns["idLivreDvd"].Visible = false;  
    tbxNumCommandeSuiviCourant.Text = string.Empty;  
    tbxNumCommandeSupprCourant.Text = string.Empty;  
}
```

On remarque l'appel à une méthode du contrôleur *getCommandesLivre* qui se charge de récupérer toutes les commandes d'un document grâce à son *Id*. Cette méthode va donc interroger l'API via sa couche d'accès aux données pour récupérer les commandes depuis la base de données. Néanmoins, l'API ne possède pas encore de méthode permettant de faire de jointures, on gèrera ce cas spécifiquement avec une méthode dédiée à la récupération des commandes.

```
protected function traitementSelect(string $table, ?array $champs) : ?array{  
    switch($table){  
        case "livre" :  
            return $this->selectAllLivres();  
        case "dvd" :  
            return $this->selectAllDvd();  
        case "revue" :  

```



```

        return $this->selectAllRevues();
    case "exemplaire" :
        return $this->selectExemplairesRevue($champs);
    case "genre" :
    case "public" :
    case "rayon" :
    case "etat" :
        // select portant sur une table contenant juste id et libelle
        return $this->selectTableSimple($table);
    case "suivi" :
        // select portant sur une table contenant juste id et libelle
        return $this->selectTableSimple($table);
    case "commandes" : // Gestion du cas des commandes
        return $this->selectAllCommandes($champs);
    default:
        // cas général
        return $this->selectTuplesOneTable($table, $champs);
    }
}

```

```

private function selectAllCommandes(?array $champs) : ?array {
    if(empty($champs) || !array_key_exists('idLivreDvd', $champs)) {
        return null;
    }
    $champNecessaire['idLivreDvd'] = $champs['idLivreDvd'];
    $requete = "SELECT commande.id, dateCommande, montant, idLivreDvd, idSuivi,
nbExemplaire FROM commandedocument JOIN commande ON commande.id =
commandedocument.id WHERE idLivreDvd = :idLivreDvd ORDER BY dateCommande DESC";
    return $this->conn->queryBDD($requete, $champNecessaire);
}

```

Cette méthode fait la jointure entre *commande* et *commandedocument* sur les commandes d'identifiants identiques, puis retourne les lignes trouvées dans la base de données. Du côté de l'application cliente, on pourra alors construire les objets du modèle et les représenter facilement dans les contrôles Windows Forms.

FrmMediatekController.cs →

```

public List<CommandeDocument> getCommandesLivre(string idLivre)
{
    return access.getCommandesLivre(idLivre);
}

```

Access.cs →

```

public List<CommandeDocument> getCommandesLivre(string idLivre)
{
    String jsonIdLivre = convertToJson("idLivreDvd", idLivre);
}

```

```

        List<CommandeDocument> lesCommandes = TraitementRecup<CommandeDocument>(GET,
"commandes/" + jsonIdLivre, null);
        return lesCommandes;
    }

```

La méthode précédente se charge de contacter l'API via une requête HTTP GET dont l'URI est de la forme <http://api.com/commandes/{ « idLivreDvd » : « XXXXX » }>

Lors de la recherche de document dans les onglets, cette méthode est utilisée pour peupler le tableau de commande *lesCommandes*. Une fois récupérées, il suffit d'associer ces données à la *DataGridView* correspondante. Pour ce faire, on utilise un *BindingSource* dont la propriété *DataSource* est fixée à la liste des commandes récupérées, puis on donne ce *BindingSource* à la *DataGridView* pour qu'elle affiche les commandes.

Extrait de *AfficherInfosCommandes* de *FrmMediatek.cs* :

```

        bdgCommandes.DataSource = this.lesCommandes;
        dgvCourante.DataSource = bdgCommandes;

```

Une étape supplémentaire de mise en forme est nécessaire pour afficher le suivi qui nécessite une colonne dans la *DataGridView*.

```

dgvCourante.Columns.Add("idSuivi_", "suivi");
for (int i = 0; i < this.lesCommandes.Count; i++) {
    dgvCourante.Rows[i].Cells["idSuivi_"].Value =
        lesSuivis[this.lesCommandes[i].idSuivi - 1].libelle;
}

```

Le tableau de *Suivi lesSuivis* contient tous les libellés de suivi ordonnés par id croissant. Ainsi, le premier *Suivi* sera le suivi correspondant à l'id de suivi 1 et ainsi de suite, de cette manière on peut faire correspondre un id de suivi avec le libellé correspondant.

## Enregistrement de commandes

L'enregistrement d'une commande est déclenché par l'utilisateur lorsqu'il clic sur le bouton d'enregistrement après avoir rempli correctement les champs de saisies proposés (voir maquette pour plus de détails). Le clic des boutons d'enregistrement des deux onglets est associé à la méthode événementielle *btnEnregistrerCommande\_Click* dont voici le contenu :

```

private void btnEnregistrerCommande_Click(object sender, EventArgs e)
{
    Button _sender = sender as Button;
    TextBox tbxMontantCourant = null;
    TextBox tbxNumDocCourant = null;
    ComboBox cbxNbExemplaireCourant = null;
    if(_sender.Name == "btnEnregistrerCommandeDvd") {
        tbxMontantCourant = tbxMontantDvd;
        tbxNumDocCourant = tbxNumDvdInfos;
    }
}

```

```

        cbxNbExemplaireCourant = cbxNumExemplaireDvd;
    } else {
        tbxMontantCourant = tbxCommandeMontant;
        tbxNumDocCourant = tbxNumLivre;
        cbxNbExemplaireCourant = cbxCommandeNbExemplaires;
    }
    if (tbxMontantCourant.Text.Length > 0 && tbxNumDocCourant.Text.Length > 0
&& livreCourant != null && cbxNbExemplaireCourant.Text.Length > 0) {
        try {
            double montant = double.Parse(tbxMontantCourant.Text,
NumberStyles.Any, new CultureInfo("en-US"));
            string idLivre = tbxNumDocCourant.Text;
            int nbExemplaire = int.Parse(cbxNbExemplaireCourant.Text);
            EnregistrerNouvelleCommande(montant, idLivre, nbExemplaire);
        } catch (Exception ex) {
            Console.WriteLine(ex);
        }
    } else {
        MessageBox.Show("Les informations de commande saisies sont
incorrectes, veuillez vérifier la saisie.", "Erreur de saisie");
    }
}

```

Après s'être assuré de la saisie et du cas dans lequel la méthode est appelée (*Livre* ou *Dvd*), on délègue l'ajout effectif de la commande à la méthode *EnregistrerNouvelleCommande* qui transfère la demande d'ajout au contrôleur puis à la classe d'accès aux données, et se charge de ré-afficher les commandes, pour que l'ajout soit pris en compte dans la liste des commandes et la *DataGridView* courante.

```

private void EnregistrerNouvelleCommande(double montant, string idLivre, int
nbExemplaire)
{
    if (controller.enregistrerNouvelleCommande(montant, idLivre, nbExemplaire)) {
        reafficherCommandes();
    } else {
        MessageBox.Show("Une erreur est survenue lors de l'enregistrement de la
nouvelle commande.");
    }
}
private void reafficherCommandes()
{
    lesCommandes = null;
    LivreDvd leLivre = livreCourant;
    livreCourant = null;
    AfficherInfosCommandes(leLivre);
}

```

La méthode *enregistrerNouvelleCommande* de la classe d'accès aux données contacte l'API via une requête HTTP POST contenant les données à insérer dans la table des commandes.

```

public bool enregistrerNouvelleCommande(double montant, string idLivre, int
nbExemplaire)
{
    Dictionary<Object, Object> parametres = new Dictionary<Object, Object>();
    parametres.Add("dateCommande", DateTime.Now.ToString("yyyy-MM-dd"));
    parametres.Add("montant", montant);
    parametres.Add("idSuivi", 1);
    parametres.Add("idLivreDvd", idLivre);
    parametres.Add("nbExemplaire", nbExemplaire);
    string jsonArray = convertToJsonArray(parametres);
}

```

```

try {
    TraitementRecup<Commande>(POST, "insert_commande", "champs=" + jsonArray);
    return true;
} catch {
    return false;
}
}

```

Comme les données à insérer sont multiples, il est nécessaire de construire une nouvelle méthode de conversion JSON des tableaux :

```

private String convertToJsonArray(Dictionary<Object, Object> nomsValeurs)
{
    Dictionary<Object, Object> dictionary = new Dictionary<Object, Object>();
    foreach (var ligne in nomsValeurs)
    {
        dictionary.Add(ligne.Key, ligne.Value);
    }
    return JsonConvert.SerializeObject(dictionary);
}

```

L'insertion d'une nouvelle commande dans la base de données implique en fait deux insertion dans deux tables différentes, les méthodes déjà présentes sont difficilement réutilisables pour ce cas précis, on décidera finalement de gérer celui-ci dans une nouvelle méthode dédiée *insérerCommandeDocument* :

```

protected function traitementInsert(string $table, ?array $champs) : ?int{
    switch($table){
        case "insert_commande" : // Gestion personnalisée
            return $this->insérerCommandeDocument($champs);
        default:
            // cas général
            return $this->insertOneTupleOneTable($table, $champs);
    }
}

private function insérerCommandeDocument($champs) : ?int
{
    if(empty($champs)) {
        return null;
    }
    $requete = "INSERT INTO commande (dateCommande, montant)
VALUES(:dateCommande, :montant);";
    $requete .= " INSERT INTO commandedocument (id, idSuivi, idLivreDvd,
nbExemplaire) VALUES((SELECT MAX(id) FROM
commande), :idSuivi, :idLivreDvd, :nbExemplaire);";
    $champNecessaire['dateCommande'] = $champs['dateCommande'];
    $champNecessaire['montant'] = $champs['montant'];
    $champNecessaire['idSuivi'] = $champs['idSuivi'];
    $champNecessaire['idLivreDvd'] = $champs['idLivreDvd'];
    $champNecessaire['nbExemplaire'] = $champs['nbExemplaire'];
    return $this->conn->updateBDD($requete, $champNecessaire);
}

```

Lorsqu'on regarde le MLD (via phpmyadmin) de la table commande, on remarque que les id de commandes sont des VARCHAR(5), ce qui rend impossible l'utilisation de AUTO\_INCREMENT pour générer des identifiants, pour pallier à cela, on crée un trigger avant insertion sur la table commande, chargé de générer les id.

```
CREATE TRIGGER `generer_id_commande` BEFORE INSERT ON `commande`  
    FOR EACH ROW  
    BEGIN  
        DECLARE nextID INT;  
        SELECT COALESCE(MAX(CAST(id AS UNSIGNED)), 0) + 1 INTO nextID  
    FROM commande;  
        SET NEW.id = LPAD(nextID, 5, '0');  
    END
```

Ce déclencheur est construit en deux parties :

D'abord, on calcule le prochain identifiant en sélectionnant le plus grand id présent dans la table commande grâce à la fonction MAX et CAST pour caster le VARCHAR en UNSIGNED INT, si la table est vide, l'identifiant est mis à 0. Dans tous les cas, on ajoute 1 à l'identifiant calculé.

En suite, on assigne le nouvel id calculé après l'avoir formaté pour correspondre à un VARCHAR de 5 caractères grâce à la fonction LPAD qui ajoute des 0 à gauche de façon à ce que l'identifiant soit toujours composé de 5 caractères.

## **Modification du suivi d'une commande**

Avant de modifier le suivi d'une commande, l'utilisateur doit sélectionner une ligne dans la *DataGridView*. L'évènement *SelectionChanged* des *DataGridView* de chaque onglet de commande est associé à *dgvCommandes\_SelectionChanged*, on distingue en suite les deux cas en fonction du nom de la *DataGridView* ayant déclenché l'appel de cette méthode événementielle. Finalement, on écrit l'identifiant de la commande dans les champs en lecture seule des *GroupBox* responsables de la modification de suivi et de la suppression de commande.

Note : l'affichage du numéro de commande est là uniquement comme une aide visuelle pour l'utilisateur, la sélection de la commande se fait bien avec les propriétés de la *DataGridView* (en particulier *DataGridView.CurrentRow.Index*).

```
private void dgvCommandes_SelectionChanged(object sender, EventArgs e)  
{  
    DataGridView dgvCourante = sender as DataGridView;
```

```

TextBox tbxNumCommandeSuiviCourant;
TextBox tbxNumCommandeSupprCourant;
ComboBox cbxSuiviCourant;
if (dgvCourante.Name == "dgvCommandesDvd") {
    tbxNumCommandeSuiviCourant = tbxNumCommandeDvd;
    tbxNumCommandeSupprCourant = tbxNumCommandeDvdSuppr;
    cbxSuiviCourant = cbxSuiviDvd;
} else {
    tbxNumCommandeSuiviCourant = tbxNumCommandeSuivi;
    tbxNumCommandeSupprCourant = tbxNumCommandeSuppr;
    cbxSuiviCourant = cbxSuivi;
}
if (dgvCourante.Focused) { //Permet de s'assurer qu'il s'agit bien de la sélection
d'une ligne par l'utilisateur et non du déclenchement automatique de cet évènement
après le binding des commandes.
    if (dgvCourante.CurrentCell != null) {
        int index = dgvCourante.CurrentCell.RowIndex;
        if (index != -1 && lesCommandes != null) {
            tbxNumCommandeSuiviCourant.Text = lesCommandes[index].id;
            tbxNumCommandeSupprCourant.Text = lesCommandes[index].id;
            cbxSuiviCourant.SelectedIndex = lesCommandes[index].idSuivi - 1;
        }
    }
}
}
}

```

Lors d'un clic sur le bouton de modification de suivi d'une commande, la méthode *button2\_Click* est appelée.

```

private void button2_Click(object sender, EventArgs e)
{
    Button _sender = sender as Button;
    DataGridView dgvCourante = null;
    ComboBox cbxSuiviCourant = null;
    if(_sender.Name == "btnModifierSuiviDvd") { //Cas des dvd
        dgvCourante = dgvCommandesDvd;
        cbxSuiviCourant = cbxSuiviDvd;
    } else { //Cas des livres
        dgvCourante = dgvCommandes;
        cbxSuiviCourant = cbxSuivi;
    }
    if (lesCommandes != null && lesCommandes.Count != 0 &&
dgvCourante.CurrentCell != null) {
        int index = dgvCourante.CurrentCell.RowIndex;
        Suivi suiviChoisi = cbxSuiviCourant.SelectedItem as Suivi;
        CommandeDocument laCommande = lesCommandes[index];
        bool error = false;
        if (laCommande != null) {
            if (suiviChoisi.id != laCommande.idSuivi) {
                // Contrôle des règles de gestion concernant l'état des suivi
                if (laCommande.idSuivi == 2 || laCommande.idSuivi == 3 &&
(suiviChoisi.id == 1 || suiviChoisi.id == 4)) {
                    error = true;
                } else if (suiviChoisi.id == 2 && laCommande.idSuivi != 3) {
                    error = true;
                }
            }
            if (!error) {
                if (controller.modifierSuiviCommande(laCommande.id,
suiviChoisi.id)) {
                    MessageBox.Show("L'état de la commande a bien été
modifié.");
                    reafficherCommandes();
                }
            }
        }
    }
}

```

```

        }
    } else {
        MessageBox.Show("L'état actuel de la commande ne permet
pas son changement à l'état choisi.", "Erreur de gestion");
    }
    } else {
        MessageBox.Show("Le suivi sélectionné est identique au suivi
actuel de la commande.", "Erreur de saisie");
    }
}
} else {
    MessageBox.Show("Vous devez d'abord sélectionner une commande pour
modifier son suivi.", "Erreur");
}
}
}

```

Après avoir discriminé l'onglet courant, on doit récupérer la commande actuellement sélectionnée par l'utilisateur. Pour ce faire, on utilise la propriété *RowIndex* sur la *DataGridView.CurrentRow*.

```

int index = dgvCourante.CurrentRow.RowIndex;
...
CommandeDocument laCommande = lesCommandes[index];

```

Les indices de la *DataGridView* et du tableau *lesCommandes* correspondent, il n'y a donc pas de risque de mélange des commandes, même lorsqu'on trie l'affichage sur une colonne, c'est en réalité le liste *lesCommandes* qui est réordonnée puis l'affichage de la *DataGridView* est mis à jour :

```

private void dgvCommandes_ColumnHeaderMouseClick(object sender,
DataGridViewCellEventArgs e)
{
    DataGridView _sender = sender as DataGridView;
    string titreColonne = _sender.Columns[e.ColumnIndex].HeaderText;
    List<CommandeDocument> sortedList = new List<CommandeDocument>();
    bool success = false;
    switch (titreColonne) {
        case "nbExemplaire":
            sortedList = lesCommandes.OrderBy(o =>
o.nbExemplaire).Reverse().ToList();
            success = true;
            break;
        case "dateCommande":
            sortedList = lesCommandes.OrderBy(o =>
o.dateCommande).Reverse().ToList();
            success = true;
            break;
        case "montant":
            sortedList = lesCommandes.OrderBy(o =>
o.montant).Reverse().ToList();
            success = true;
            break;
        case "suivi":
            sortedList = lesCommandes.OrderBy(o => o.idSuivi).ToList();
            success = true;
            break;
        default: break;
    }
    if (success) {
        lesCommandes = sortedList;
    }
}

```

```

        AfficherInfosCommandes(livreCourant);
    }
}

```

Avant de valider la modification du suivi d'une commande, il convient de vérifier que le suivi choisi respecte les règles de gestions énoncées dans le cahier des charges :

« 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). »

Ce qui se traduit par les conditions suivantes :

```

if (laCommande.idSuivi == 2 || laCommande.idSuivi == 3 && (suiviChoisi.id == 1 ||
suiviChoisi.id == 4))
{
    error = true;
}
else if (suiviChoisi.id == 2 && laCommande.idSuivi != 3)
{
    error = true;
}

```

Si la variable *error* vaut *true*, la modification n'est pas effectuée, et un message d'erreur est affiché à l'utilisateur. Dans le cas contraire, on autorise la modification en appelant la méthode *modifierSuiviCommande* du contrôleur.

```

if (!error) {
    if (controller.modifierSuiviCommande(laCommande.id, suiviChoisi.id)) {
        MessageBox.Show("L'état de la commande a bien été modifié.");
        reafficherCommandes();
    }
}

```

Le contrôleur transfère l'appel à la couche d'accès qui se charge d'interroger l'API via une requête HTTP PUT (pour mettre à jour l'enregistrement dans la table) :

```

public bool modifierSuiviCommande(string idCommande, int idSuivi)
{
    string jsonIdSuivi = convertToJson("idSuivi", idSuivi);
    Console.WriteLine(jsonIdSuivi);
    try {
        TraitementRecup<CommandeDocument>(PUT, "commandedocument/" + idCommande,
"champs=" + jsonIdSuivi);
        return true;
    } catch (Exception ex) {
        return false;
    }
}

```

Ce cas est correctement géré dans l'API, il n'y a donc pas besoin de créer une méthode spécifique pour cette action.



## Suppression de commande

La méthode qui gère l'événement de clic sur le bouton de suppression est nommée *button3\_Click*, de la même façon que pour modifier le suivi d'une commande, on récupère la commande sélectionnée par l'utilisateur dans la *DataGridView*, si cette commande n'est pas livrée, on appelle la méthode du contrôleur chargée de la suppression.

FrmMediatekController.cs →

```
public bool supprimerCommande(string idCommande)
{
    return access.supprimerCommande(idCommande);
}
```

Access.cs →

```
public bool supprimerCommande(string idCommande)
{
    Object jsonId = convertToJson("id", idCommande);
    try
    {
        Console.WriteLine(jsonId);
        TraitementRecup<CommandeDocument>(DELETE, "commandedocument/" + jsonId,
null);
        return true;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex);
        return false;
    }
}
```

Cette méthode envoie une requête HTTP DELETE à l'API, qui possède des méthodes déjà écrites pour la suppression de données dans une table. Les paramètres de la requête sont la table : « commandedocument » ainsi que l'id de la commande au format JSON.

Pour automatiser la suppression de la commande dans la classe fille, on utilise le trigger suivant après suppression d'un enregistrement dans la table commandedocument :

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

De cette façon, on préserve la cohérence des données dans les deux tables.

## Création automatisée des exemplaires de documents

Lorsqu'une commande passe à l'état « Livrée », le système doit générer les exemplaires correspondants en les numérotant séquentiellement par rapport au document concerné.

Pour cela, on utilise un déclencheur après UPDATE sur la table commandedocument, si l'état de suivi passe à « livré ».

```
CREATE TRIGGER `generer_exemplaires` AFTER UPDATE ON `commandedocument`  
FOR EACH ROW BEGIN  
    DECLARE numeroExemplaire INT;  
    DECLARE dateAchatCommande DATE;  
    DECLARE compteur INT DEFAULT 0;  
    IF (NEW.idSuivi = 3) THEN  
        SELECT dateCommande INTO dateAchatCommande FROM commande WHERE  
id=OLD.id;  
        SELECT max(numero) + 1 INTO numeroExemplaire FROM exemplaire WHERE  
id=OLD.idLivreDvd;  
        IF (numeroExemplaire IS NULL) THEN  
            SET numeroExemplaire = 1;  
        END IF;  
        WHILE (compteur < OLD.nbExemplaire) DO  
            INSERT INTO exemplaire VALUES (OLD.idLivreDvd, numeroExemplaire,  
dateAchatCommande, "", "00001");  
            SET compteur = compteur + 1;  
            SET numeroExemplaire = numeroExemplaire + 1;  
        END WHILE;  
    END IF;  
END
```

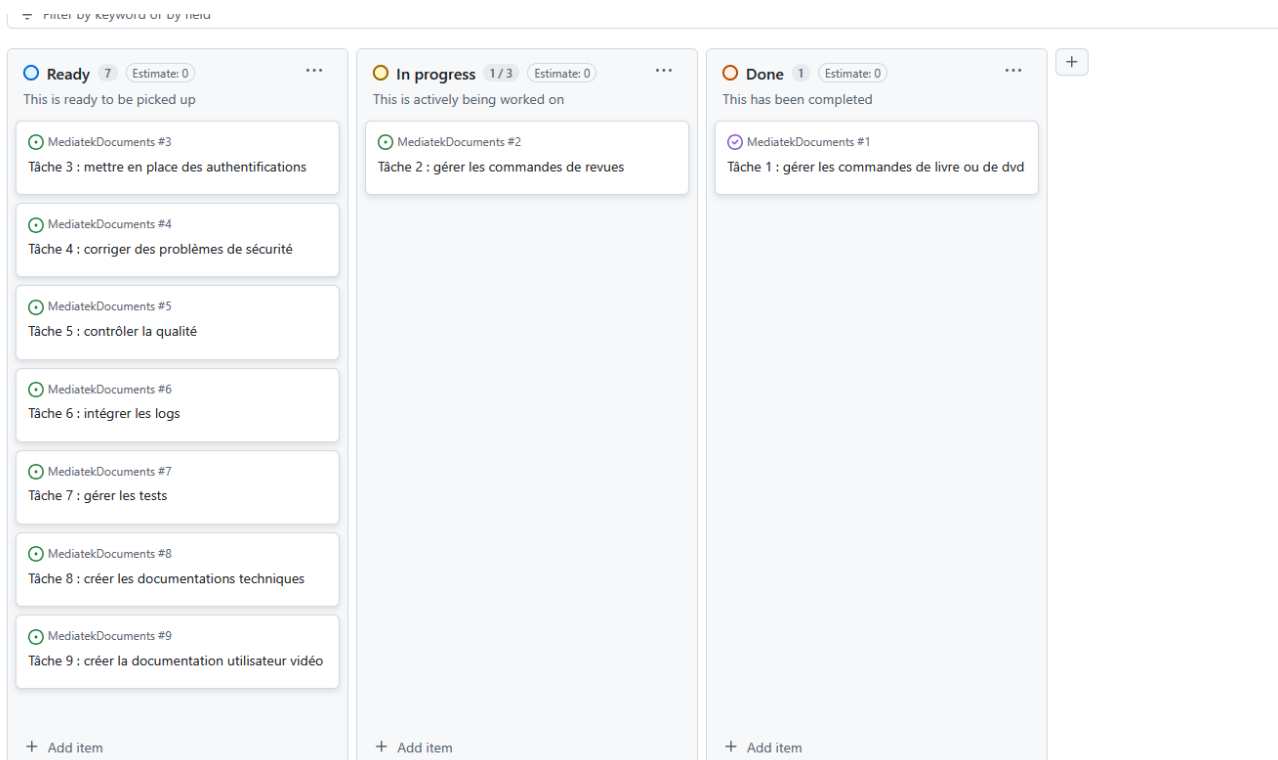
## Tâche n°2 : gérer les abonnements

Une commande de revue revient à réaliser un abonnement ou un renouvellement d'abonnement (les deux sont équivalents). L'application doit permettre de voir la liste des abonnements de revues, une commande ne peut être supprimée que si aucun exemplaire ne lui est rattaché. Au démarrage de l'application, une fenêtre doit s'afficher avec la liste des abonnements qui expirent dans moins de 30 jours.

Durée estimée : 4h

Durée effective : 5h58

## Aperçu du kanban de la tâche en cours



## Maquette de l'interface d'abonnement

**Gestion des documents de la médiathèque**

Livres

Dvd

Revue

Parution des revues

Commandes livre

Commande DVD

**Abonnements**

Recherche

Saisir un numéro de revue :

	dateCommande	montant	date fin abonnement
	21/11/1999	10	21/11/2025
	21/11/1999	5.99	21/12/2025

Informations détaillées

Numéro de document :

Titre :

Auteur :

Genre :

Public :

Rayon :

Périodicité :

Délai mise à disposition :

Nouvelle commande / renouvellement

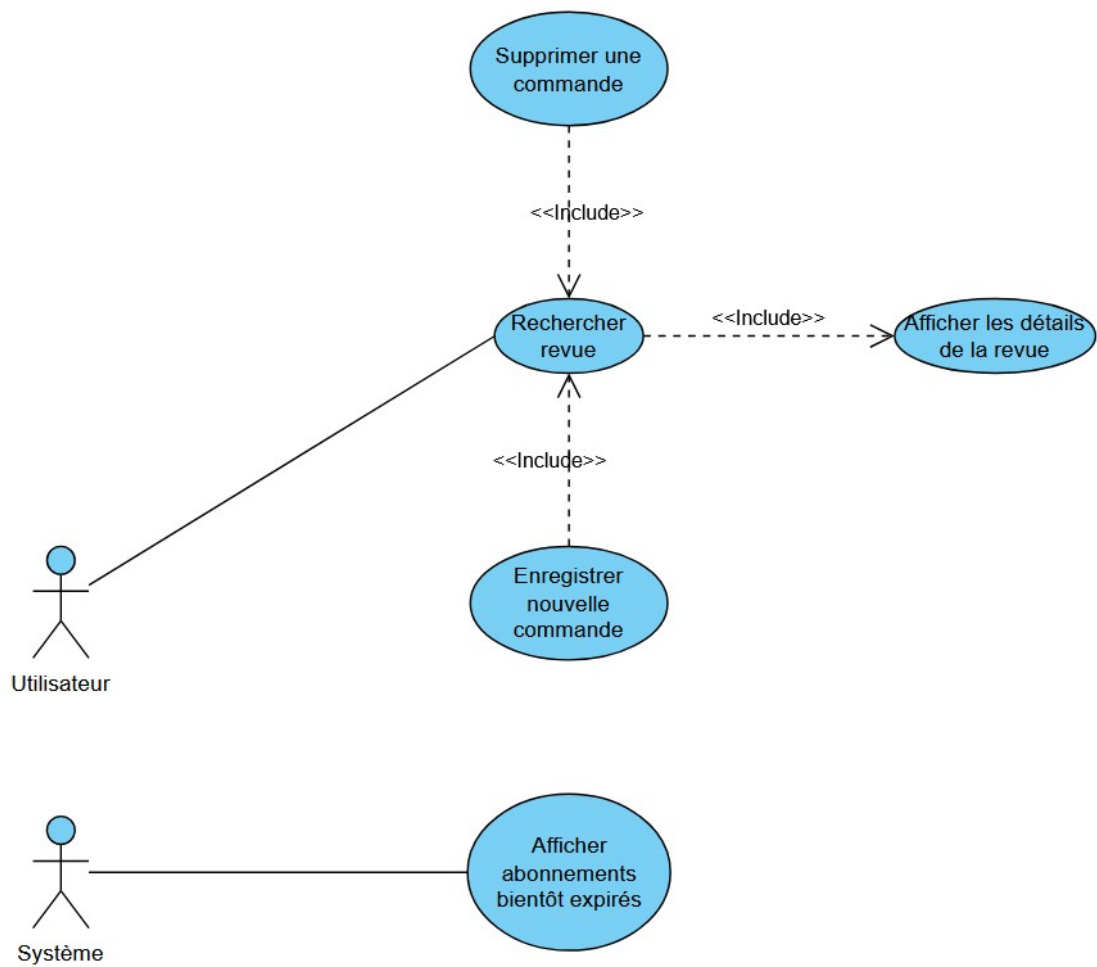
Montant :

Date de fin d'abonnement :

Supprimer un abonnement

Numéro d'abonnement :

## Diagramme des cas d'utilisations de la tâche



## Ajout de l'objet Abonnement dans le modèle

Pour pouvoir gérer les abonnements, il est nécessaire d'ajouter une classe objet dans le modèle. On remarque sur le MCD que toute commande est en réalité soit une commande de document, soit un abonnement. Pour représenter ce lien, la classe abonnement héritera de commande, ainsi lorsqu'on manipulera l'objet abonnement, on pourra également récupérer les informations de la table commande qui lui sont associées.

```

public class Abonnement : Commande
{
    public DateTime dateFinAbonnement { get; set; }
    public string idRevue { get; set; }
    public Abonnement(string id, DateTime dateCommande, double montant, DateTime dateFinAbonnement, string idRevue) : base(id, dateCommande, montant)
    {
        this.dateFinAbonnement = dateFinAbonnement;
        this.idRevue = idRevue;
    }
}
  
```

Les seules propriétés supplémentaires d'un abonnement sont la date de fin de l'abonnement de type *DateTime*, et l'identifiant de revue de type *string*.

## **Recherche d'une revue et des abonnements associés**

La recherche de revue s'effectue, comme pour les livres et dvd, à l'aide de son identifiant à saisir dans la *TextBox* de recherche. Une fois saisi, on recherche la revue correspondante dans la liste de toutes les revues récupérées depuis la base de données.

La récupération de la liste des revues se fait à l'entrée sur l'onglet abonnement via la méthode événementielle suivante (FrmMediatek.cs) :

```
private void tabAbonnementRevue_Enter(object sender, EventArgs e)
{
    lesRevues = controller.GetAllRevues();
}
```

Où *lesRevues* est une *List<Revue>* accessible uniquement dans la classe d'affichage.

On associe le clic sur le bouton de recherche à la méthode suivante :

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

Après s'être assuré de la saisie en vérifiant le texte présent dans les *TextBox* et stocké la revue dans une variable de portée de classe *revueCourante*

```
revueCourante = lesRevues.Find(x => x.Id.Equals(tbxRechercheRevue.Text));
```

on affiche les informations de la revue ainsi que les abonnements qui lui sont associés à l'aide de deux méthodes :

```
afficherInfosRevues();
afficherInfosCommandesRevue();
```

La première se charge simplement de mettre à jour la propriété *Text* des *TextBox* du *GroupBox* contenant les informations détaillées de la revue :

```
private void afficherInfosRevues()
{
```

```

    tbxInfoNumRevue.Text = revueCourante.Id;
    tbxInfoTitreRevue.Text = revueCourante.Titre;
    tbxInfoRayonRevue.Text = revueCourante.Rayon;
    tbxInfoGenreRevue.Text = revueCourante.Genre;
    tbxInfoPublicRevue.Text = revueCourante.Public;
    tbxInfoPeriodiciteRevue.Text = revueCourante.Periodicite;
    tbxInfoMiseADispo.Text = revueCourante.DelaiMiseADispo.ToString();
}

```

La deuxième affiche les abonnements rattachés à la revue courante :

```

private void afficherInfosCommandesRevue(bool sorted=false)
{
    if(revueCourante != null) {
        dgvCommandesRevue.Columns.Clear();
        dgvCommandesRevue.Rows.Clear();
        if(!sorted) {
            lesAbonnements = controller.getAbonnementsRevue(revueCourante.Id);
        }
        bdgAbonnement.DataSource = lesAbonnements;
        dgvCommandesRevue.DataSource = bdgAbonnement;
        dgvCommandesRevue.Columns["idRevue"].Visible = false;
        dgvCommandesRevue.Columns["id"].Visible = false;
    }
}

```

Lorsque le paramètre *sorted* est fixé à true, la méthode ne récupère pas à nouveau les commandes depuis l'API, elle est utilisée de cette façon pour ré-afficher la liste des abonnements après tri sur une colonne de la DataGridView, lorsque la liste *lesAbonnements* a été triée.

Dans les autres cas, on va chercher les informations d'abonnements dans la base de données par l'intermédiaire de l'API avec un méthode de la couche d'accès aux données appelée par le contrôleur de la fenêtre.

```

public List<Abonnement> getAbonnementsRevue(string idRevue)
{
    String jsonIdRevue = convertToJson("idRevue", idRevue);
    try {
        List<Abonnement> lesAbonnements = TraitementRecup<Abonnement>(GET,
"abonnements/" + jsonIdRevue, null);
        return lesAbonnements;
    } catch(Exception e) {
        return null;
    }
}

```

Une fois les abonnements de la revue récupérées, la DataGridView est mise à jours avec la nouvelle source de données :

Méthode `afficherInfosCommandesRevue`

```

    bdgAbonnement.DataSource = lesAbonnements;
    dgvCommandesRevue.DataSource = bdgAbonnement;

```

## Enregistrement d'un nouvel abonnement

Du point de vue de l'API, l'enregistrement d'une nouvelle commande nécessite l'utilisation de deux requêtes consécutives.

D'abord il faut insérer une ligne dans la table commande, puis ensuite insérer une ligne dans la table abonnement (qui dépend de commande sur la colonne id).

Ce qui se traduit par le code SQL suivant :

```
INSERT INTO
    commande (dateCommande, montant)
VALUES
    (:dateCommande, :montant);
INSERT INTO
    abonnement (id, dateFinAbonnement, idRevue)
VALUES
    ((SELECT MAX(id) FROM commande), :dateFinAbonnement, :idRevue);
```

Rappel : la table commande possède un trigger avant insertion qui calcul le prochain identifiant de commande.

On a ajouté un cas spécifique dans *traitementInsert* pour effectuer cette tâche :

```
case "insert_abonnement" :
    return $this->insererAbonnementRevue($champs);
[...]

private function insererAbonnementRevue($champs) : ?int
{
    if(empty($champs)) {
        return null;
    }
    $requete = "INSERT INTO commande (dateCommande, montant)
VALUES(:dateCommande, :montant);";
    $requete .= " INSERT INTO abonnement (id, dateFinAbonnement, idRevue)
VALUES((SELECT MAX(id) FROM commande), :dateFinAbonnement, :idRevue);";
    $champNecessaire['dateCommande'] = $champs['dateCommande'];
    $champNecessaire['montant'] = $champs['montant'];
    $champNecessaire['dateFinAbonnement'] = $champs['dateFinAbonnement'];
    $champNecessaire['idRevue'] = $champs['idRevue'];
    return $this->conn->updateBDD($requete, $champNecessaire);
}
```



Du point de vue de l'application cliente, comme pour l'ajout de commande de livre/dvd, on doit vérifier que la saisie est correcte : une revue est sélectionnée et le formulaire d'enregistrement est complété. Pour cela, on associe la méthode événementielle *btnEnregistrerAbonnement\_Click* au clic sur le bouton d'enregistrement d'abonnement.

```
private void btnEnregistrerAbonnement_Click(object sender, EventArgs e)
{
    if(revueCourante != null) {
        if(tbxMontantAbonnement.Text.Length > 0 && dtpDateFinAbonnement.Checked) {
            if(dtpDateFinAbonnement.Value > DateTime.Now) {
                double montant = double.Parse(tbxMontantAbonnement.Text,
NumberStyles.Any, new CultureInfo("en-US"));
                string idRevue = revueCourante.Id;
                DateTime dateFinAbonnement = dtpDateFinAbonnement.Value;
                if (controller.enregistrerAbonnement(montant, idRevue,
dateFinAbonnement)) {
                    MessageBox.Show("L'abonnement a bien été enregistré.");
                    btnRechercheRevue.PerformClick();
                } else {
                    MessageBox.Show("Une erreur interne est survenue lors de
l'enregistrement de l'abonnement.", "Erreur interne");
                }
            } else {
                MessageBox.Show("Erreur, la date choisie doit être postérieure à la
date actuelle.", "Erreur de saisie");
            }
        } else {
            MessageBox.Show("Erreur de saisie, vous devez saisir tous les champs du
formulaire d'abonnement.");
        }
    } else {
        MessageBox.Show("Erreur, vous devez d'abord rechercher une revue avant de
pouvoir enregistrer un abonnement.");
    }
}
```

L'ajout réel de l'abonnement se traduit par ce passage :

```
...
double montant = double.Parse(tbxMontantAbonnement.Text, NumberStyles.Any, new
CultureInfo("en-US"));
string idRevue = revueCourante.Id;
DateTime dateFinAbonnement = dtpDateFinAbonnement.Value;
...controller.enregistrerAbonnement(montant, idRevue, dateFinAbonnement)...
```

On récupère et cast si besoin les informations saisies, puis on demande au contrôleur d'envoyer ces informations à l'API, qui se chargera d'insérer les données dans les tables correspondantes.

*controller.enregistrerAbonnement(...)* transfère les paramètres à la méthode *enregistrerAbonnement* de Access.cs dont voici le corps :

```
public bool enregistrerAbonnement(double montant, string idRevue, DateTime
dateFinAbonnement)
{
    Dictionary<Object, Object> parametres = new Dictionary<Object, Object>();
    parametres.Add("dateCommande", DateTime.Now.ToString("yyyy-MM-dd"));
```

```

parametres.Add("montant", montant);
parametres.Add("dateFinAbonnement", dateFinAbonnement.ToString("yyyy-MM-dd"));
parametres.Add("idRevue", idRevue);
string jsonArray = convertToJsonArray(parametres);
try {
    TraitementRecup<Abonnement>(POST, "insert_abonnement", "champs=" + jsonArray);
    return true;
} catch (Exception ex) {
    Console.WriteLine(ex.ToString());
    return false;
}
}

```

Cette méthode envoie une requête HTTP POST à l'API avec comme paramètres les données de l'abonnement à insérer dans les tables commande et abonnement au format JSON.

## Suppression d'abonnement

La suppression d'un abonnement n'est possible que si aucun exemplaire n'est rattaché à l'abonnement, c'est à dire que la date de parution de l'exemplaire ne doit pas être comprise entre la date de commande et la date de fin de l'abonnement.

Pour traduire cette règle de gestion, on définit la méthode *ParutionDansAbonnement* :

```

private bool ParutionDansAbonnement(DateTime dateCommande, DateTime dateFinAbonnement,
DateTime dateParution)
{
    return dateParution >= dateCommande && dateParution <= dateFinAbonnement;
}

```

Donc la suppression n'est possible que si la méthode précédente retourne faux.

La méthode événementielle qui gère cette suppression se sert de cette méthode pour autoriser ou non la suppression :

```

private void btnSupprAbonnement_Click(object sender, EventArgs e)
{
    if (dgvCommandesRevue.CurrentCell != null) {
        int index = dgvCommandesRevue.CurrentCell.RowIndex;
        Abonnement abonnementCourant = lesAbonnements[index];
        List<Exemplaire> exemplairesRevue =
controller.GetExemplairesRevue(abonnementCourant.idRevue);
        bool supprAutorise = true;
        if (exemplairesRevue != null) {
            foreach (Exemplaire exemplaire in exemplairesRevue) {
                if (ParutionDansAbonnement(abonnementCourant.dateCommande,
abonnementCourant.dateFinAbonnement, exemplaire.DateAchat)) {
                    supprAutorise = false;
                    break;
                }
            }
        }
        if(supprAutorise) {
            if (controller.supprimerAbonnement(abonnementCourant.id)) {

```

```

        MessageBox.Show("L'abonnement a bien été supprimé.");
        btnRechercheRevue.PerformClick();
    } else {
        MessageBox.Show("Erreur interne lors de la suppression de
l'abonnement.", "Erreur interne");
    }
    } else {
        MessageBox.Show("Erreur, l'abonnement n'a pas pu être supprimé car un
ou plusieurs exemplaires lui sont rattaché(s).", "Erreur de gestion");
    }
    }
}
}

```

On distingue deux passages importants dans la méthode précédente, d'abord la récupération des exemplaires associés à la revue :

```

...
int index = dgvCommandesRevue.CurrentCell.RowIndex;
//Abonnement sélectionné dans la DataGridView
Abonnement abonnementCourant = lesAbonnements[index];
//Liste des exemplaires concernant la revue dont l'abonnement est sélectionné
List<Exemplaire> exemplairesRevue =
controller.GetExemplairesRevue(abonnementCourant.idRevue);
...

```

Et d'autre part, la vérification des dates de parution des exemplaires par rapport à la date de l'abonnement :

```

bool supprAutorise = true;
if (exemplairesRevue != null)
{
    foreach (Exemplaire exemplaire in exemplairesRevue)
    {
        if (ParutionDansAbonnement(abonnementCourant.dateCommande,
abonnementCourant.dateFinAbonnement, exemplaire.DateAchat))
        {
            supprAutorise = false;
            break;
        }
    }
    if(supprAutorise)
    {
        ...controller.supprimerAbonnement(abonnementCourant.id) ; ...
    }
}
...

```

On parcourt la liste des exemplaires, et pour chacun d'entre eux, on vérifie que la date de parution n'est pas incluse entre la date de commande et la date de fin de l'abonnement. Si ne serait-ce qu'un seul exemplaire est présent dans cette intervalle de temps, on termine la boucle et on interdit la suppression. Dans le cas contraire, on permet la suppression en demandant au contrôleur de faire supprimer l'abonnement.

La méthode de la couche d'accès aux données *supprimerAbonnement* se charge d'envoyer une requête HTTP de type DELETE à l'API avec l'identifiant de l'abonnement en paramètre.

Pour automatiser la suppression dans la table commande (dont dépend l'abonnement), on utilise un trigger après suppression sur la table abonnement, ainsi, pour chaque abonnement supprimé, la commande correspondante le sera également.

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

## **Rappel des abonnements qui expirent dans moins de 30 jours.**

Pour déterminer une liste des abonnements qui expirent dans moins de 30 jours, il est nécessaire de pouvoir comparer des dates entre elles à l'aide des opérateurs <, >, ≥, ≤ ou du mot clef BETWEEN, ce que ne permet actuellement pas l'API. On gèrera donc ce nouveau cas spécifiquement dans la méthode *traitementSelect*, via une méthode supplémentaire.

```
case "abonnements_30jours":  
    return $this->getAbonnementsExpirationProche($champs);
```

```
private function getAbonnementsExpirationProche($champs) : ?array  
{  
    if(empty($champs)) {  
        return null;  
    }  
    $requete = "SELECT commande.id, commande.dateCommande, commande.montant,  
abonnement.dateFinAbonnement, abonnement.idRevue FROM commande JOIN abonnement ON  
commande.id = abonnement.id WHERE dateFinAbonnement <= :maxDate AND  
dateFinAbonnement >= NOW() ORDER BY dateFinAbonnement ASC";  
    $champNecessaire['maxDate'] = $champs['maxDate'];  
    return $this->conn->queryBDD($requete, $champNecessaire);  
}
```

La liste d'objets abonnements se forme grâce à la jointure des colonnes entre commande et abonnement pour un même id de commande.

Du côté de l'application cliente, il ne reste plus qu'à récupérer les lignes retournées par cette requête à l'ouverture du programme, puis, de les afficher dans une DataGridView si la liste n'est pas vide.

Voici l'événement déclenché à l'ouverture de l'application :

```
private void FrmMediatek_Shown(object sender, EventArgs e)
{
    VerifierAbonnementsBientotExpires();
}
private void VerifierAbonnementsBientotExpires()
{
    abonnementsExpirationProche = controller.getAbonnementsExpirationProche();
    lesRevue = controller.GetAllRevue();
    if (abonnementsExpirationProche.Count > 0) {
        using (Form form = new Form())
        {
            form.SetBounds(0, 0, 512, 256);
            form.StartPosition = FormStartPosition.CenterParent;
            form.FormBorderStyle = FormBorderStyle.FixedSingle;
            form.MaximizeBox = false;
            DataGridView dgvAboExpires = new DataGridView();
            form.Text = "Abonnements expirant dans moins de 30 jours.";
            dgvAboExpires.Width = 512;
            dgvAboExpires.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
            dgvAboExpires.DataSource = abonnementsExpirationProche;
            dgvAboExpires.Columns.Add("titreRevue", "Titre");
            dgvAboExpires.DataBindingComplete += dgvAboExpires_DataBindingComplete;
            form.Controls.Add(dgvAboExpires);
            form.ShowDialog(this);
        }
    }
}
```

On construit « programmatiquement » la fenêtre (*Form*) et les *Controls* responsables d'afficher les abonnements qui expirent prochainement.

Néanmoins, l'affichage ne pouvant se contenter d'afficher les abonnements tels quels, il faut modifier le format d'affichage pour correspondre aux contraintes énoncées dans le cahier des charges, à savoir, afficher uniquement le titre de la revue et la date d'expiration de l'abonnement concerné. Pour ce faire, on utilise l'événement qui se déclenche après que l'association des données avec la DataGridView soit terminé :

```
private void dgvAboExpires_DataBindingComplete(object sender, EventArgs e)
{
    DataGridView _sender = sender as DataGridView;
    _sender.Columns["id"].Visible = false;
    _sender.Columns["dateCommande"].Visible = false;
    _sender.Columns["montant"].Visible = false;
    _sender.Columns["idRevue"].Visible = false;
    _sender.Columns["dateFinAbonnement"].HeaderText = "Date d'expiration";
    _sender.Refresh();
    for (int i = 0; i < _sender.RowCount; i++)
    {

```

```

        _sender.Rows[i].Cells["titreRevue"].Value = lesRevue.Find(x =>
x.Id.Equals(abonnementsExpirationProche[i].idRevue)).Titre;
    }
}

```

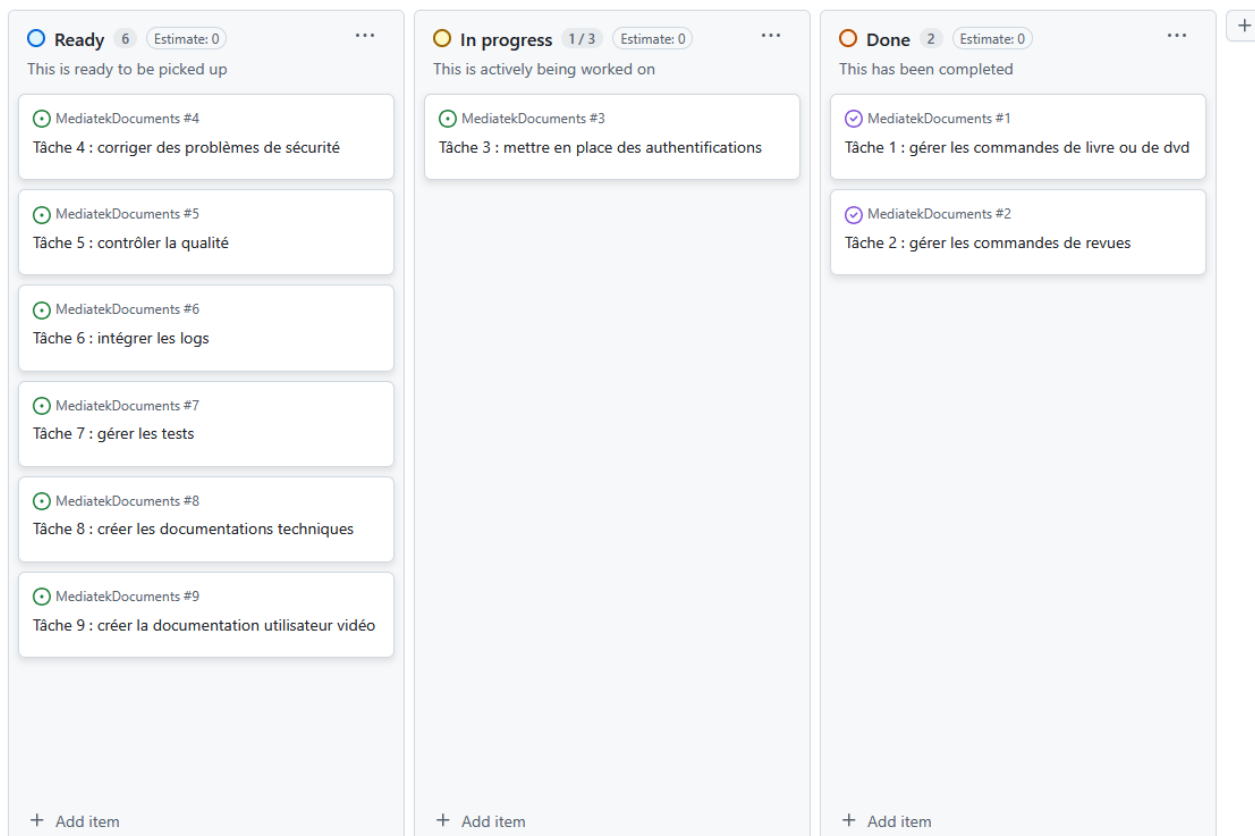
## Tâche n°3 : mise en place des authentifications.

L'administrateur et le personnel du service administratif ont accès à toutes les fonctionnalités, les employés du services "prêts" n'ont accès qu'aux fonctionnalités de consultation des documents, les employés du service "culture" (organisation des événements) ne sont pas censés manipuler cette application.

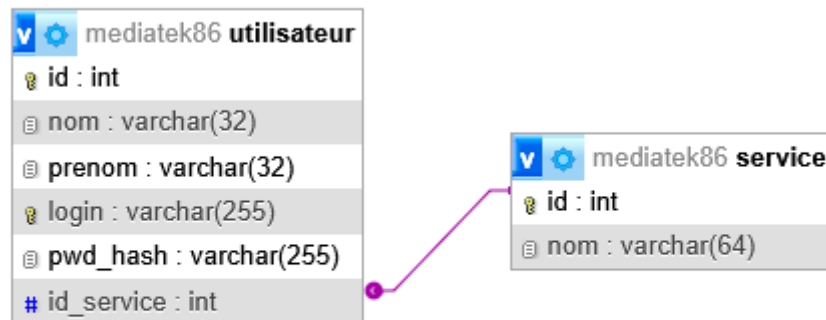
Durée estimée : 4h

Durée effective : 3h32

### Aperçu du kanban de la tâche en cours.



## Modèle logique de données concernant l'ajout d'authentification.



## Ajout d'entité dans le modèle

Pour représenter les utilisateurs et le service auquel ils sont rattachés, on crée une nouvelle classe objet Utilisateur.

```
public class Utilisateur
{
    public int id { get; set; }
    public string nom { get; set; }
    public string prenom { get; set; }
    public int id_service { get; set; }
    public string pwd_hash { get; set; }
    public Utilisateur(int id, string nom, string prenom, int id_service, string
pwd_hash)
    {
        this.id = id;
        this.nom = nom;
        this.prenom = prenom;
        this.id_service = id_service;
        this.pwd_hash = pwd_hash;
    }
}
```

## Classe auxiliaire de sécurisation des mots de passes utilisateurs (CryptoTools.cs)

La gestion des mots de passes utilisateurs nécessite l'utilisation de procédés cryptographiques sûrs, et sans avoir recours à des bibliothèques ou services tiers. C'est pour cette raison qu'on a choisi d'utiliser des méthodes du namespace *System.Security.Cryptography*.

Cet espace de nom possède des méthodes permettant d'utiliser une technique de hashage appelée « Password-Based Key Derivation Function 2 ». PBKDF2 fait appel à des algorithmes de hashage connus et éprouvés tels que SHA-(HMAC)-256/512, mais l'intérêt majeur est l'aspect de dérivation de clef.

En effet, le processus de hashage est répété un nombre déterminable de fois, ce qui rend l'attaque par force brute beaucoup plus longue. L'intérêt étant de trouver un nombre d'itération rendant le temps de brute force basé sur des GPU ou autre types de puces modernes non acceptable pour un attaquant, tout en étant capable de générer un hash en temps raisonnable pour les besoins de l'application.

Cette classe possède des constantes pour paramétrer le hashage, à noter qu'il serait préférable de placer ces valeurs dans des fichiers de configurations pour qu'il n'y ait pas à redéployer le programme en cas de changement de l'une d'elle (la taille du salt peut avoir à évoluer avec le temps, le nombre d'itérations également).

On choisit de définir la taille du *salt* sur 64 bytes :

```
private const int SALT_SIZE = 64;
```

On définit la taille du *hash* sur 512 bits = 64 bytes car on utilise l'algorithme SHA-512 :

```
private const int HASH_SIZE = 64;
```

On définit le nombre d'itérations de dérivation de clef sur 210 000 conformément à une recommandation de l'OWASP sur PBKDF2 avec SHA-HMAC-512 :

[https://cheatsheetseries.owasp.org/cheatsheets/Password\\_Storage\\_Cheat\\_Sheet.html#pbkdf2](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#pbkdf2)

```
private const int ITERATIONS = 210000;
```

Deux méthodes ont été ajoutées dans cette classe, l'une pour hasher une chaîne en claire passée en paramètre (un mot de passe), et l'autre pour vérifier qu'un hash de référence (celui contenu en base de données) correspond bien à la chaîne d'entrée.

```
public static string HashPassword(string plaintext)
{
    if(plaintext.Length <= 0) {
        return string.Empty;
    }
}
```



```

    byte[] salt = new byte[SALT_SIZE];
    var generator = RandomNumberGenerator.Create();
    generator.GetBytes(salt);
    Rfc2898DeriveBytes hasher = new Rfc2898DeriveBytes(plaintext, salt, ITERATIONS,
ALGO);
    byte[] hash = hasher.GetBytes(HASH_SIZE);
    byte[] hashBytes = new byte[SALT_SIZE + HASH_SIZE];
    Array.Copy(salt, 0, hashBytes, 0, SALT_SIZE);
    Array.Copy(hash, 0, hashBytes, SALT_SIZE, HASH_SIZE);
    return Convert.ToBase64String(hashBytes);
}

```

Le processus de génération de *hash* se fait en plusieurs étapes :

- D'abord, on doit générer un salt de façon aléatoire en utilisant une méthode cryptographique, puis on stocke ce *salt* dans le tableau de bytes *salt*.

```

var generator = RandomNumberGenerator.Create();
generator.GetBytes(salt);

```

- Ensuite, on peut passer à la génération du *hash* avec la classe *Rfc2898DeriveBytes* en lui passant les paramètres de configuration du hashage, c'est à dire, la chaîne d'entrée, le *salt*, le nombre d'itérations et l'algorithme de chiffrement utilisé. On stocke ensuite le *hash* résultat dans le tableau de bytes *hash*.

```

Rfc2898DeriveBytes hasher = new Rfc2898DeriveBytes(plaintext, salt, ITERATIONS, ALGO);
byte[] hash = hasher.GetBytes(HASH_SIZE);

```

- Enfin, il ne reste plus qu'à fusionner les deux tableau de bytes *salt* et *hash* pour produire la somme de contrôle finale puis à retourner la chaîne de représentation du *hash salé* en base64 :

```

byte[] hashBytes = new byte[SALT_SIZE + HASH_SIZE];
Array.Copy(salt, 0, hashBytes, 0, SALT_SIZE);
Array.Copy(hash, 0, hashBytes, SALT_SIZE, HASH_SIZE);
return Convert.ToBase64String(hashBytes);

```

L'autre méthode sert à déterminer l'identité entre un hash de référence et une chaîne en claire, autrement dit, elle vérifie qu'un mot de passe en clair, une fois passé dans la fonction de hashage avec les mêmes paramètres (salt) que le hash de référence, retourne un hash strictement identique :

```

public static bool VerifyPassword(string plaintext, string hashReference)
{
    if(plaintext.Length <= 0 || hashReference.Length <= 0) {
        return false;
    }
    byte[] hashBytes = Convert.FromBase64String(hashReference);
    byte[] salt = new byte[SALT_SIZE];

```

```

        Array.Copy(hashBytes, 0, salt, 0, SALT_SIZE);
        var pbkdf2 = new Rfc2898DeriveBytes(plaintext, salt, ITERATIONS, ALGO);
        byte[] hash = pbkdf2.GetBytes(HASH_SIZE);
        for(int i = 0; i < HASH_SIZE; i++) {
            if (hash[i] != hashBytes[SALT_SIZE + i]) {
                return false;
            }
        }
        return true;
    }
}

```

Cette méthode se découpe en trois grandes parties :

D'abord, on récupère le *hash* de référence qu'on stocke dans le tableau de bytes *hashBytes*, puis on sépare le *salt* du *hash*, qu'on copie dans un tableau *salt* :

```

byte[] hashBytes = Convert.FromBase64String(hashReference);
byte[] salt = new byte[SALT_SIZE];
Array.Copy(hashBytes, 0, salt, 0, SALT_SIZE);

```

Ensuite, on procède à la génération du *hash* de la chaîne d'entrée en clair avec les mêmes paramètres de hashage utilisés par le *hash* de référence (*salt*, nombre d'itérations de dérivation de clef, algorithme de chiffrement). On stocke le *hash* généré dans le tableau de bytes *hash*.

```

var pbkdf2 = new Rfc2898DeriveBytes(plaintext, salt, ITERATIONS, ALGO);
byte[] hash = pbkdf2.GetBytes(HASH_SIZE);

```

Enfin, il ne reste plus qu'à comparer le *hash* de référence et le *hash* précédemment généré. Comme on manipule des *array* de bytes, on vérifiera chaque élément du tableau un par un dans une boucle *for*, à noter qu'on ne compare que les deux *hash* et non pas le *hash* combiné au *salt*. C'est pour cette raison qu'il y a un décalage d'indice sur le tableau *hashBytes* dans la boucle (car ce tableau combine le *salt* et le *hash*) :

```

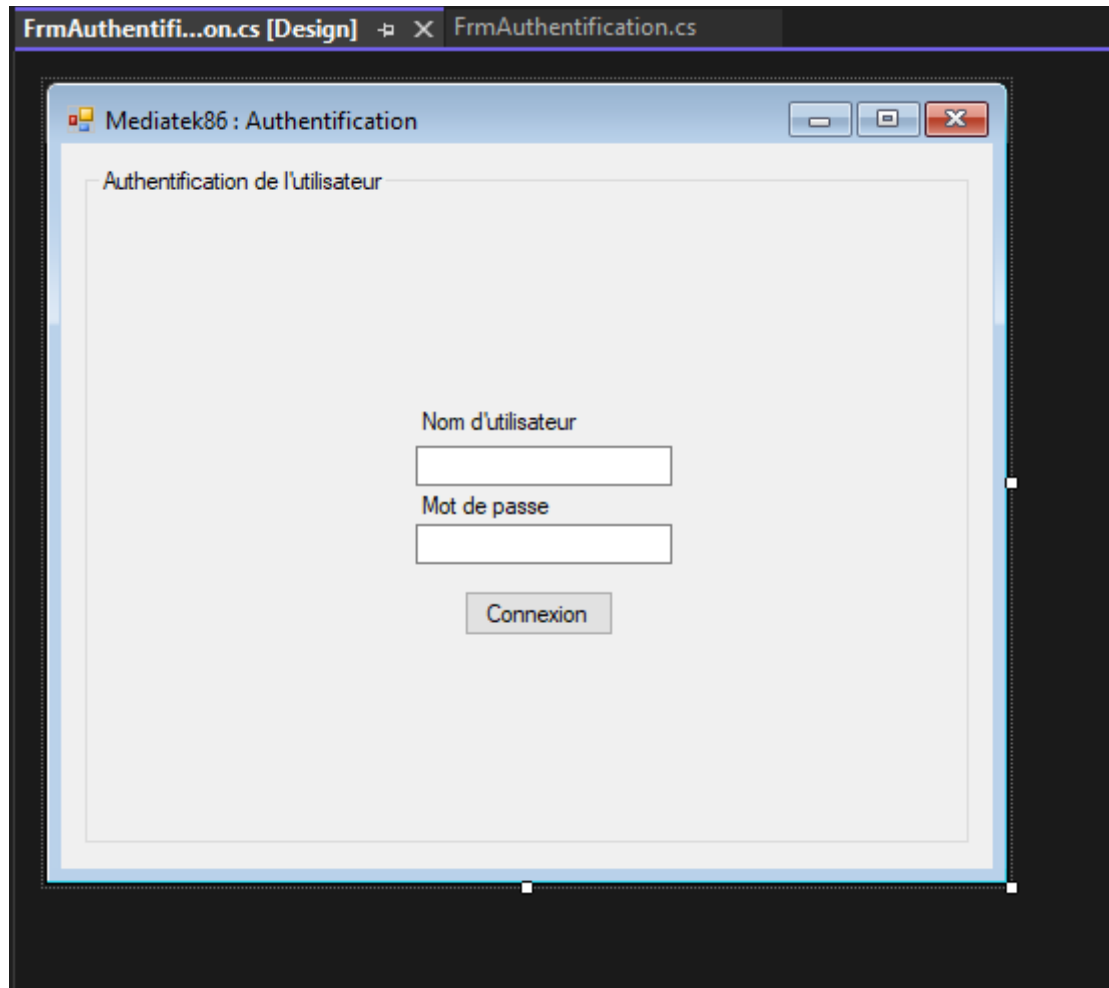
for(int i = 0; i < HASH_SIZE; i++) {
    if (hash[i] != hashBytes[SALT_SIZE + i]) {
        return false;
    }
}
return true;

```

La classe *CryptoTools* peut maintenant être utilisée concrètement dans notre application C#.

## Création d'une nouvelle vue pour la fenêtre d'authentification.

Au démarrage de l'application, on doit voir une fenêtre d'authentification demandant à l'utilisateur de saisir un nom d'utilisateur et un mot de passe afin de pouvoir l'identifier et lui attribuer les habilitations nécessaires à son service de rattachement.



Pour lancer cette fenêtre au lieu de la fenêtre principale au démarrage, il suffit de modifier le point d'entrée du programme (Program.cs) en remplaçant la classe d'affichage appelée par la classe d'affichage de la fenêtre d'authentification :

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new FrmMediatek());
    Application.Run(new FrmAuthentification());
}
```

On gère l'événement de clic sur le bouton de connexion avec la méthode suivante :

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

La connexion implique de retrouver le *hash* du mot de passe de l'utilisateur concerné dans la base, pour ce faire, on a besoin d'une nouvelle méthode dans le *DAL* permettant d'accéder à cette information.

```
public Utilisateur getUserInfos(string login)
{
    if(login.Length <= 0 || login == null) {
        return null;
    }
    List<Utilisateur> utilisateur = null;
    try {
        string jsonLogin = convertToJson("login", login);
        utilisateur = TraitementRecup<Utilisateur>(GET, "utilisateur/" + jsonLogin,
null);
    } catch (Exception ex) {
        Console.WriteLine(ex.Message);
        return null;
    }
    if(utilisateur.Count > 0) {
        return utilisateur[0];
    } else {
        return null;
    }
}
```

La méthode ci-dessus contacte l'API via une requête HTTP GET en lui demandant de récupérer la ligne de la table utilisateur dont le champ login est fixé à la valeur saisie par l'utilisateur lors de sa tentative de connexion. Au retour de l'API, si les données ont été trouvées, on génère l'objet utilisateur puis on le retourne. La classe objet Utilisateur possède une propriété *pwd\_hash*, ce qui nous permet de l'utiliser comme *hash* de référence dans notre méthode *CryptoTools.VerifyPassword*.

```
private void btnConnexion_Click(object sender, EventArgs e)
{
    [...]
    Utilisateur utilisateur = controller.getUserInfos(tbxLogin.Text);
    [...]
    if(CryptoTools.VerifyPassword(tbxPwd.Text, utilisateur.pwd_hash)) {
        MessageBox.Show("Authentification réussie");
        FrmMediatek frm = new FrmMediatek(utilisateur);
        this.Hide();
        frm.ShowDialog();
        this.Close();
    }
    [...]
}
```

*CryptoTools.VerifyPassword(tbxPwd.Text, utilisateur.pwd\_hash)* retourne *true* si le mot de passe saisi en clair correspond bien au *hash* de référence récupéré depuis la base de données, et *false* sinon.

Si la vérification est positive, on charge la vue principale de l'application :

```
FrmMediatek frm = new FrmMediatek(utilisateur);
frm.ShowDialog();
```

Note : on a ajouté un paramètre de type Utilisateur au constructeur de la classe *FrmMediatek*, c'est en fait l'utilisateur actuellement authentifié, de cette façon on garde une référence vers son service de rattachement, ce qui est utile pour savoir quels onglets afficher ou non. Par ailleurs, cela reste utile pour le futur de l'application, dans le cas où on souhaiterait ajouter un onglet accessible aux administrateurs permettant de gérer les comptes utilisateurs et leurs privilèges.

## **Restriction des Controls visibles à l'utilisateur en fonction de son service**

Dans *FrmMediatek.cs* (la classe d'affichage de la vue principale), on définit une nouvelle propriété privée représentant l'utilisateur actuellement authentifié.

```
private Utilisateur utilisateur = null;
```

C'est le constructeur de la classe qui se charge de lui affecter la bonne valeur via un passage en paramètre depuis la fenêtre d'authentification :

```
internal FrmMediatek(Utilisateur utilisateur)
{
    InitializeComponent();
    this.utilisateur = utilisateur;
    this.controller = new FrmMediatekController();
    disableControlsForUser();
    [...]
}
```

Donc à cette étape du programme, on est en mesure de connaître le service de l'utilisateur. A la fin du constructeur, on appelle une méthode chargée de cacher les *Controls* non accessibles à l'utilisateur, en fonction de son id de service.

```
public void disableControlsForUser()
{
    switch(utilisateur.id_service) {
        case 1: //Service administratif
            break;
        case 2: //Service prêts
            tabOngletsApplication.TabPages.Remove(tabReceptionRevue);
            tabOngletsApplication.TabPages.Remove(tabCommandeLivre);
            tabOngletsApplication.TabPages.Remove(tabCommandeDvd);
            tabOngletsApplication.TabPages.Remove(tabAbonnementRevue);
            break;
        case 3: //Service culture
            MessageBox.Show("Le personnel du service culture n'est pas habilité à accéder à l'application.", "Privilèges insuffisants.");
            Application.Exit();
            break;
        case 4: //Service administrateurs
            break;
        default: break;
    }
}
```

Finalement, il ne reste plus qu'à conditionner l'affichage du pop-up de rappel des abonnements qui expirent bientôt.

La méthode responsable de l'affichage de cette fenêtre est *VerifierAbonnementsBientotExpires*, elle est appelée depuis la méthode événementielle *FrmMediatek\_Shown* qui se déclenche automatiquement lorsque la fenêtre principale est chargée. Il ne reste plus qu'à ajouter la condition suivante avant l'appel de cette méthode :

```
private void FrmMediatek_Shown(object sender, EventArgs e)
{
    if(utilisateur!= null && utilisateur.id_service == 1 || utilisateur.id_service ==
4) {
        VerifierAbonnementsBientotExpires();
    }
}
```

De cette façon, le rappel ne sera affiché qu'aux utilisateurs des services concernés.

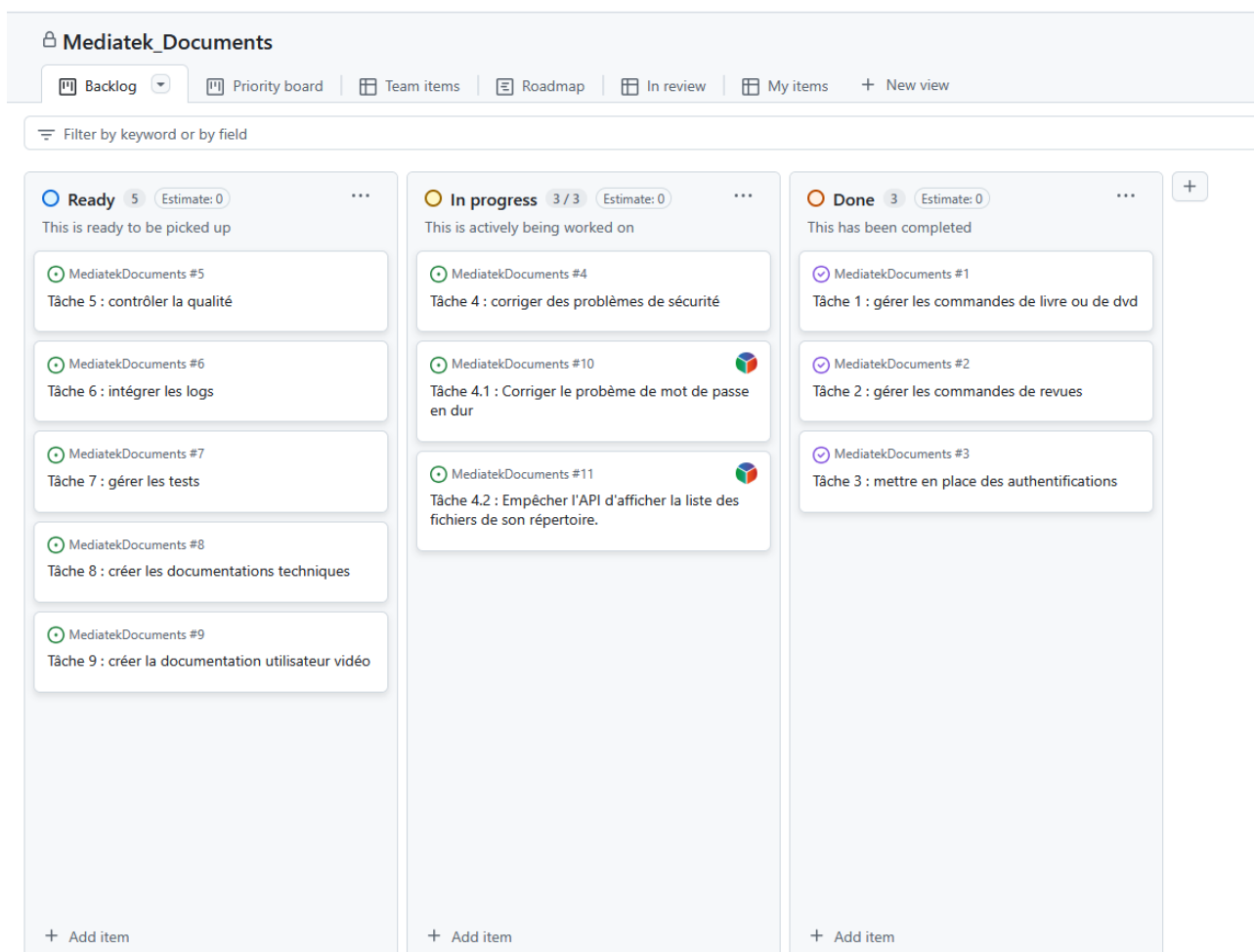
## **Tâche n°4 : corriger des problèmes de sécurité.**

Deux problèmes de sécurité ont été identifiés dans l'application C# et l'API. Le premier problème concerne l'écriture des identifiants de l'API en dur dans le code de la class Access.cs. Le deuxième concerne l'accès à l'API depuis un navigateur, en effet, il est possible d'afficher la liste des fichiers du répertoire racine du site, il faut empêcher cet accès.

Durée estimée : 2h

Durée effective : 1h16

### **Aperçu du kanban de la tâche en cours**



## Issues et Pull Requests liés aux problèmes identifiés.

Trois issues liées aux problèmes identifiés ont été créées.

- Tâche 4 : corriger des problèmes de sécurité
  - Tâche 4.1 : Corriger le problème de mot de passe en dur
  - Tâche 4.2 : Empêcher l'API d'afficher la liste des fichiers de son répertoire

L'issue « Tâche 4 » est en quelque sorte une issue bilan servant à indiquer que les deux sous tâches 4.1 et 4.2 (représentées par des issues) ont été réalisées.

<input type="checkbox"/>	<input checked="" type="checkbox"/>	Tâche 4.2 : Empêcher l'API d'afficher la liste des fichiers de son répertoire.	
#11 - by DevBlocks42 was closed 12 minutes ago			
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Tâche 4.1 : Corriger le problème de mot de passe en dur	
#10 - by DevBlocks42 was closed 12 minutes ago			
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Tâche 4 : corriger des problèmes de sécurité	
#4 - by DevBlocks42 was closed 11 minutes ago			

## Pull request du premier correctif :

The screenshot shows a GitHub pull request for a repository. At the top, the status bar indicates 0 conversations, 3 commits, 0 checks, and 6 files changed, with a net change of +13 -6 lines. A comment from DevBlocks42 states: "Les identifiants de connexion ne sont plus en clair dans le code mais dans le fichier de configuration App.config". Below the comment, a commit history shows three commits: "Tâche 4.1 : Corriger le problème de mot de passe en dur" (342c0f2), "gitignore" (de7a965), and "Update App.config" (9941f1f, marked as Verified). The pull request is self-assigned to DevBlocks42. On the right, the 'Reviewers' section shows 'No reviews' and a link to learn about draft PRs. The 'Assignees' section shows DevBlocks42. The 'Labels' section is empty. The 'Projects' section is empty. The 'Milestone' section is empty. The 'Development' section shows 'Successfully merging this pull request may close these issues.' and 'None yet'. The 'Notifications' section shows an 'Unsubscribe' button and a message: 'You're receiving notifications because you're watching this repository.' The '1 participant' section shows DevBlocks42. At the bottom, a 'Lock conversation' button is visible. A large green box at the bottom of the pull request indicates 'No conflicts with base branch' and 'Merging can be performed automatically.' It contains a 'Commit message' field with the text 'Merge pull request #12 from DevBlocks42/correction\_access' and an 'Extended description' field with the text 'Correction access'. At the bottom of this box are 'Confirm merge' and 'Cancel' buttons. A note at the bottom of the box states: 'This commit will be authored by 136115859+DevBlocks42@users.noreply.github.com.'

## Pull request du deuxième correctif :

### Tâche 4.2 : Empêcher l'API d'afficher la liste des fichiers de son ré... #1

Edit <> Code

The screenshot shows a GitHub pull request for a repository. At the top, the status bar indicates 0 conversations, 1 commit, 0 checks, and 1 file changed, with a net change of +2 -0 lines. A comment from DevBlocks42 states: "...pertoire. Il n'est désormais plus possible d'accéder à la liste des fichiers du répertoire courant de l'API depuis un navigateur." Below the comment, a commit history shows one commit: "Tâche 4.2 : Empêcher l'API d'afficher la liste des fichiers de son ré..." (1f594a8). The pull request is self-assigned to DevBlocks42. On the right, the 'Reviewers' section shows 'No reviews' and a link to learn about draft PRs. The 'Assignees' section shows DevBlocks42. The 'Labels' section is empty. The 'Projects' section is empty. The 'Milestone' section is empty. The 'Development' section shows 'Successfully merging this pull request may close these issues.' and 'None yet'. The 'Notifications' section shows an 'Unsubscribe' button and a message: 'You're receiving notifications because you're watching this repository.' The '1 participant' section shows DevBlocks42. At the bottom, a 'Lock conversation' button is visible. A large green box at the bottom of the pull request indicates 'No conflicts with base branch' and 'Merging can be performed automatically.' It contains a 'Merge pull request' button and a link to 'View command line instructions'. A note at the bottom of the box states: 'You can also merge this with the command line. View command line instructions.'



## Mise en place des correctifs

Pour la première correction, il suffit de placer les identifiants de connexion à l'API dans le fichier de configuration App.config en racine du projet. Par mesure de sécurité, on veillera à ne pas inclure ce fichier dans le dépôt git en ajoutant une référence vers ce fichier dans le .gitignore :

MediaTekDocuments/App.config

La chaîne de connexion s'ajoute à l'intérieur de la balise « connectionStrings » comme suit :

```
<connectionStrings>
  <clear />
  <add name="access"
    connectionString="nom_utilisateur:mot_de_passe"/>
</connectionStrings>
```

Il ne reste plus qu'à charger le contenu de cette chaîne de connexion nommée « access » dans le constructeur de la classe Access.cs :

```
authenticationString =
ConfigurationManager.ConnectionStrings["access"].ConnectionString;
api = ApiRest.GetInstance(uriApi, authenticationString);
```

En ce qui concerne la deuxième correction, il s'agit d'ajouter une condition et une redirection dans le fichier .htaccess de l'API.

La condition doit identifier les requêtes n'ayant pas de paramètres :

```
RewriteCond %{QUERY_STRING} ^champs=$
```

La redirection doit simplement rediriger vers une erreur HTTP 400 (bad request) :

```
RewriteRule ^ - [R=400,L]
```

# Tâche 5 : contrôler la qualité de code.

Cette tâche nous demande de contrôler le respect des règles de bonne pratique énoncées par SonarQube. Pour ce faire, on lancera une analyse complète de la solution puis on modifiera les extraits de code qui ne respectent pas ces règles.

Durée estimée : 1h

Durée effective : 1h36

## Aperçu du kanban de la tâche en cours.

The screenshot shows a Jira Kanban board for the project 'Mediatek\_Documents'. The board is organized into three columns: 'Ready', 'In progress', and 'Done'. Each column has a header with a status icon, a count of items, and an 'Estimate' field. Below the headers, tasks are listed as cards, each with a status icon, a task ID, and a description. The 'Ready' column contains four tasks (IDs #6, #7, #8, #9). The 'In progress' column contains one task (ID #5). The 'Done' column contains five tasks (IDs #1, #2, #3, #10, #11). Each task card has a plus icon at the bottom right for adding more details. The board also features a filter bar at the top and a 'New view' button.

**Mediatek\_Documents**

Backlog | Priority board | Team items | Roadmap | In review | My items | + New view

Filter by keyword or by field

**Ready** 4 Estimate: 0  
This is ready to be picked up

- MediatekDocuments #6  
Tâche 6 : intégrer les logs
- MediatekDocuments #7  
Tâche 7 : gérer les tests
- MediatekDocuments #8  
Tâche 8 : créer les documentations techniques
- MediatekDocuments #9  
Tâche 9 : créer la documentation utilisateur vidéo

+ Add item

**In progress** 1/3 Estimate: 0  
This is actively being worked on

- MediatekDocuments #5  
Tâche 5 : contrôler la qualité

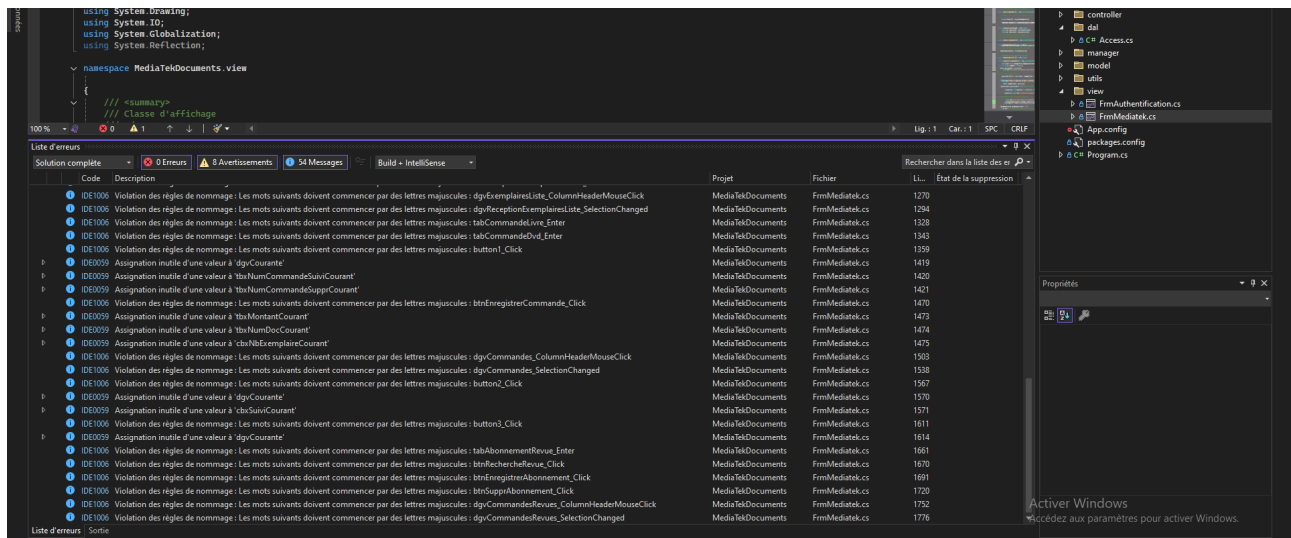
+ Add item

**Done** 6 Estimate: 0  
This has been completed

- MediatekDocuments #1  
Tâche 1 : gérer les commandes de livre ou de dvd
- MediatekDocuments #2  
Tâche 2 : gérer les commandes de revues
- MediatekDocuments #3  
Tâche 3 : mettre en place des authentifications
- MediatekDocuments #10  
Tâche 4.1 : Corriger le problème de mot de passe en dur
- MediatekDocuments #11  
Tâche 4.2 : Empêcher l'API d'afficher la liste des fichiers de son répertoire.
- MediatekDocuments #4  
Tâche 4 : corriger des problèmes de sécurité

+ Add item

# Règles mise en évidence par SonarQube.



Parmi les règles non respectées dans le code, on remarque surtout le non respect des conventions de nommage, notamment dans les propriétés des classes du modèle, en effet, une propriété doit commencer par une majuscule selon cette règle.

Lorsqu'on modifie le nom d'une propriété, il suffit d'utiliser la fonctionnalité de renommage de Visual Studio, qui se charge de modifier toutes les occurrences de cette propriété. Il faut néanmoins faire attention à ne pas renommer cette propriété lors de l'ajout des paramètres de requêtes vers l'API car les noms de colonnes de la base de données ne commencent pas par une majuscule.

D'autre part, SonarQube nous donne des indications sur les conditions imbriquées qui peuvent être simplifiées. Parfois, on imbrique des conditions entre elles car cela est plus évident à comprendre lorsqu'on construit le programme, néanmoins pour du code de production, il est préférable de fusionner les conditions qui peuvent l'être.

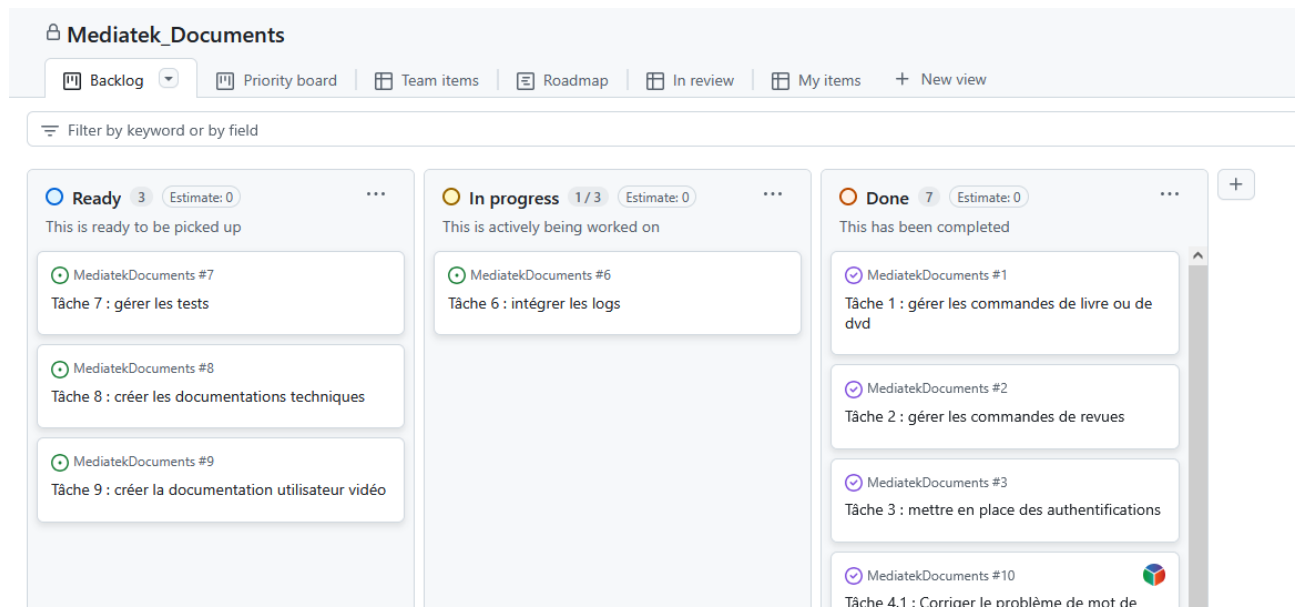
## Tâche n°6 : intégrer les logs.

Dans la classe Access, ajouter le code de configuration des logs et des logs au niveau de chaque affichage console (à enregistrer dans un fichier de logs).

Durée estimée : 1h

Durée effective : 40 min

## Kanban de la tâche en cours



## Classe auxiliaire pour la gestion des logs.

Pour gérer l'écriture des journaux, on a ajouté une nouvelle classe static (LoggingUtils.cs) utilisable partout dans le code. Cette classe possède une propriété représentant le chemin d'accès au fichier de journalisation récupéré depuis le fichier App.config ainsi qu'une méthode static *LogStringToFile* qui prend une chaîne de caractère en paramètre et l'écrit dans le fichier de logs en l'accompagnant de la date et l'heure de l'action.

```
private static string logFilePath =
ConfigurationManager.ConnectionStrings["log_file_location"].ConnectionString;

public static void LogStringToFile(string str)
{
    try
    {
        DateTime now = DateTime.Now;
        StreamWriter writer = new StreamWriter(LoggingUtils.logFilePath, true);
        writer.Write("[ " + now.ToString("dd/MM/yyyy HH:mm:ss") + " ] " + str +
"\r\n");
        writer.Flush();
        writer.Close();
    } catch (IOException ex) {
        Console.WriteLine(ex.Message);
    }
}
```

La classe `LoggingUtils.cs` peut désormais être utilisée de manière concrète dans notre application C#.

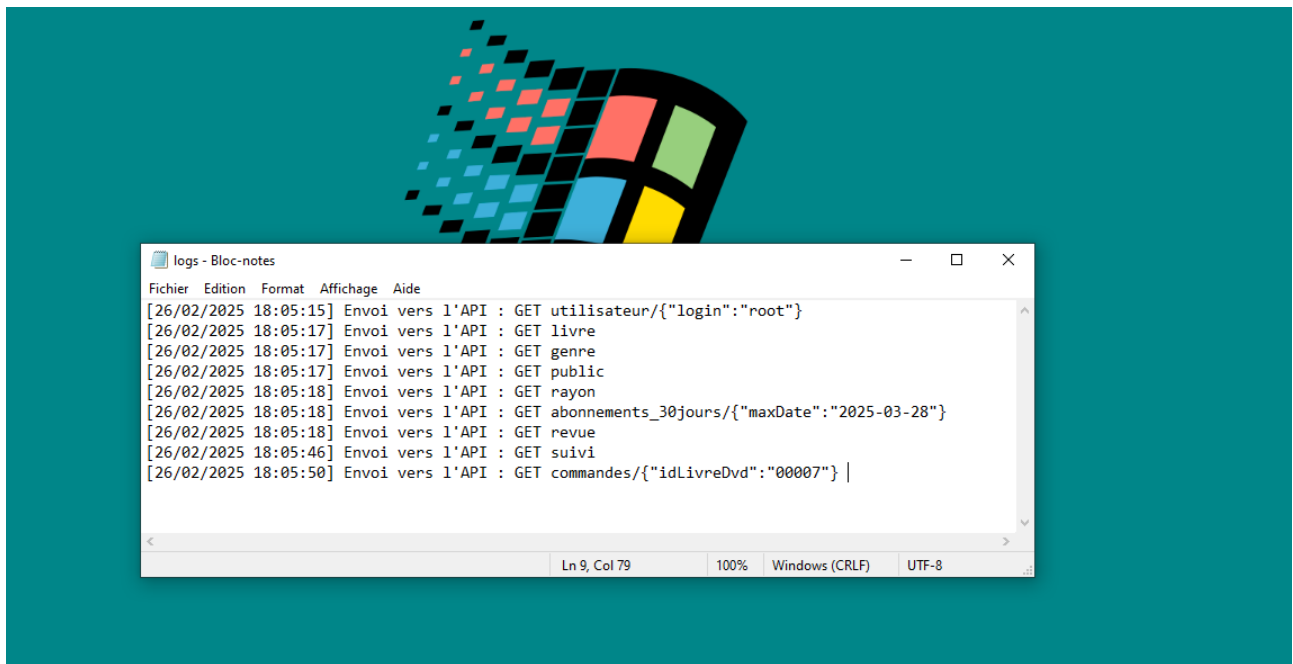
## Journalisation des affichages console de la classe Access

Pour journaliser les affichages consoles, il suffit d'appeler la méthode précédemment créée avec comme paramètre la chaîne à afficher.

Exemple pour la méthode *TraitementRecup* :

```
[...]
LoggingUtils.LogStringToFile("Envoi vers l'API : " + methode + " " + message + " " +
parametres);
[...]
LoggingUtils.LogStringToFile("code erreur = " + code + " message = " +
(String)retour["message"]);
[...]
LoggingUtils.LogStringToFile("Erreur lors de l'accès à l'API : " + e.Message);
```

Voici un aperçu du fichier de journalisation après ouverture de l'application :



# Tâche n°7 : gérer les tests.

Écrire les tests unitaires sur les classes du package Model. Dans l'application C#, écrire les tests fonctionnels sur les recherches dans l'onglet des livres. Construire une collection de tests dans Postman pour contrôler les fonctionnalités de l'API d'accès à la base de données.

Durée estimée : 5h

Durée effective : 6h53

## Kanban de la tâche en cours.

Mediatek\_Documents

Backlog

Priority board

Team items

Roadmap

In review

My items

New view

Filter by keyword or by field

Ready2Estimate: 0

This is ready to be picked up

MediatekDocuments #8

Tâche 8 : créer les documentations techniques

MediatekDocuments #9

Tâche 9 : créer la documentation utilisateur vidéo

+ Add item

In progress1 / 3Estimate: 0

This is actively being worked on

MediatekDocuments #7

Tâche 7 : gérer les tests

+ Add item

Done8Estimate: 0

This has been completed

MediatekDocuments #1

Tâche 1 : gérer les commandes de livre ou de dvd

MediatekDocuments #2

Tâche 2 : gérer les commandes de revues

MediatekDocuments #3

Tâche 3 : mettre en place des authentifications

MediatekDocuments #10

Tâche 4.1 : Corriger le problème de mot de passe en dur

MediatekDocuments #11

Tâche 4.2 : Empêcher l'API d'afficher la liste des fichiers de son répertoire.

MediatekDocuments #4

Tâche 4 : corriger des problèmes de sécurité

MediatekDocuments #5

Tâche 5 : contrôler la qualité

+ Add item

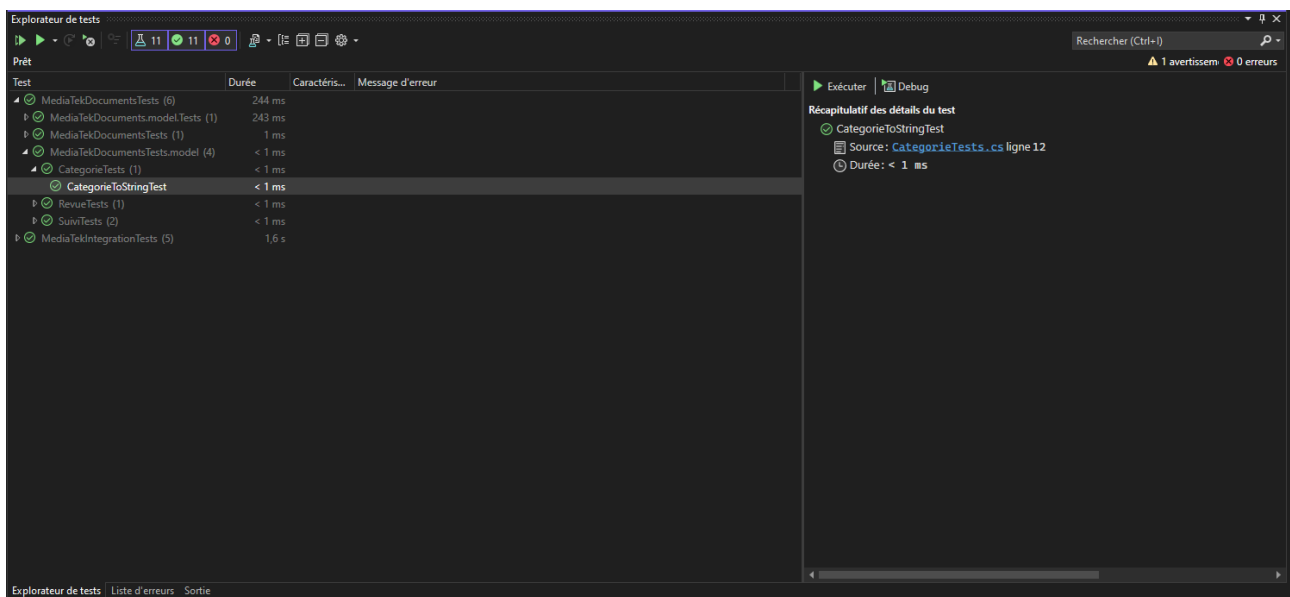
## Tests unitaires sur le modèle

Les tests unitaires sur le modèle permettent de vérifier que la logique métier correspond bien aux attentes et que les méthodes retournent des valeurs cohérentes.

Voici l'exemple du test unitaire sur la méthode ToString de la classe Categorie :

```
[TestMethod]
public void CategorieToStringTest()
{
    Genre genre = new Genre("00001", "Horreur");
    Assert.AreEqual("Horreur", genre.ToString());
}
```

Lorsqu'on lance ce test via l'explorateur de test Visual Studio, le programme instancie un nouveau Genre d'id « 00001 » et de libellé « Horreur », ensuite on vérifie que la valeur retournée par la méthode ToString sur le genre précédemment créé est bien fixée à « Horreur ». Si c'est le cas le test réussit, sinon il échoue.



## Tests d'intégrations sur les fonctionnalités de recherche.

Pour gérer les tests d'intégrations, on a utilisé l'extension Reqnroll (anciennement SpecFlow). Cette extension permet, à partir d'un scénario de test écrit en langage naturel (feature file), de générer le patron des méthodes de test dans un fichier C#. Il ne reste plus qu'à invoquer les Controls d'UI et d'interagir avec eux comme si on était en train d'utiliser l'application en direct.

Note : pour des raisons de simplicité, les tests ont été gérés en écrivant les identifiants de connexion en dur dans le code (et toutes les informations requises par l'application qui ont été migrées dans le fichier de config), car il est difficile de récupérer le fichier de configuration d'un autre projet.

Voici un exemple de test portant sur la recherche d'un livre par son numéro de document.

Feature file (description du scénario en langage naturel) :

**Feature:** RechercheLivreNumero

*A short summary of the feature*

@tag1

**Scenario:** recherche\_livre\_num

**Given** Je saisis l'identifiant "00017" dans la TextBox de recherche.

**When** Je clique sur le bouton de recherche.

**Then** Le résultat s'affiche dans la DataGridView

Le squelette de test généré en C# :

```
[Binding]
public class RechercheLivreNumeroStepDefinitions
{
    [Given("Je saisis l'identifiant {string} dans la TextBox de recherche.")]
    public void GivenJeSaisisLidentifiantDansLaTextBoxDeRecherche_(string p0)
    {}
    [When("Je clique sur le bouton de recherche.")]
    public void WhenJeCliqueSurLeBoutonDeRecherche_()
    {}
    [Then("Le résultat s'affiche dans la DataGridView")]
    public void ThenLeResultatSafficheDansLaDataGridView()
    {}
}
```

Il ne reste plus qu'à écrire le déroulement du scénario. Tout d'abord, on instancie la fenêtre principale du programme FrmMediatek, celle-ci a besoin d'une instance d'utilisateur en paramètre du constructeur.

```
private readonly FrmMediatek frm = new FrmMediatek(new Utilisateur(1, "Doe", "John",
4, "blabla"));
```

Note : l'utilisateur n'existe pas en base de données, on a généré cet objet de façon arbitraire. Pour autant, on a accès à l'intégralité de l'application ce qui veut dire que le système d'authentification n'est pas correctement sécurisé. En effet, si l'application démarre bien sur la fenêtre d'authentification en temps normal, il est toujours possible de contourner l'authentification en démarrant directement sur la fenêtre principale à l'aide d'un utilisateur fictif. Il faudrait revérifier



l'authentification à chaque chargement de fenêtre pour régler le problème, ainsi un utilisateur fictif ne pourrait plus être utilisé puisqu'il y aurait une vérification de son existence en base de données.

Une fois la fenêtre instanciée, on se place dans la première méthode de test. Le but est de récupérer le *Control TextBox* devant recevoir le numéro de document. Ensuite, on modifie sa propriété *Text* pour qu'elle contienne l'identifiant de document « 00017 ». A la fin de la méthode, on vérifie que l'identifiant saisi dans le *Control* correspond bien à celui en paramètre de la méthode.

```
public void GivenJeSaisisLidentifiantDansLaTextBoxDeRecherche_(string p0)
{
    frm.Show();
    TextBox tbxNumDoc =
    (TextBox)frm.Controls["tabOngletsApplication"].Controls["tabLivres"].Controls["grpLivresRecherche"].Controls["txbLivresNumRecherche"];
    tbxNumDoc.Text = p0;
    Assert.AreEqual(p0, tbxNumDoc.Text);
}
```

Si jamais l'identifiant contenu dans *tbxNumDoc.Text* ne correspond pas au paramètre, le test échoue, sinon il se poursuit.

La deuxième méthode se charge simplement d'émuler un clic sur le bouton de recherche :

```
public void WhenJeCliqueSurLeBoutonDeRecherche_()
{
    Button btnRecherche =
    (Button)frm.Controls["tabOngletsApplication"].Controls["tabLivres"].Controls["grpLivresRecherche"].Controls["btnLivresNumRecherche"];
    btnRecherche.PerformClick();
}
```

Puis, arrive la dernière méthode dans laquelle on s'assure que le résultat affiché correspond au résultat attendu. Dans ce cas de test, le titre du seul livre affiché à la fin doit être « *Catastrophes au Brésil* ». Si ce n'est pas le cas, le test échoue, sinon il réussit.

```
public void ThenLeResultatSafficheDansLaDataGridView()
{
    DataGridView dgvLivres =
    (DataGridView)frm.Controls["tabOngletsApplication"].Controls["tabLivres"].Controls["grpLivresRecherche"].Controls["dgvLivresListe"];
    Assert.AreEqual("Catastrophes au Brésil", dgvLivres.Rows[0].Cells["Titre"].Value);
}
```

## Collection de tests sur l'API via Postman.

Pour tester le bon fonctionnement de l'API, on a utilisé le logiciel Postman afin de construire et envoyer nos requêtes HTTP.

On a testé les différentes méthodes du protocole HTTP :

- GET : pour récupérer des données
- POST : pour insérer des données
- PUT : pour mettre à jour des données
- DELETE : pour supprimer des données

Voici un exemple de test portant sur la méthode POST :

URL : `http://localhost/rest_mediatekdocuments/insert_commande`

METHOD : POST

champs : `{"dateCommande":"2025-01-01", "montant":12.99, "idSuivi":1, "idLivreDvd":"00017", "nbExemplaire":2}`

La fonction de test javascript :

```
pm.test("Status code is 200", function () {  
    pm.response.to.have.status(200);  
});
```

Cette fonction vérifie juste que le code de réponse de la requête est 200 (Succès). Dans le cas où l'insertion aurait échoué, nous n'aurions pas obtenu ce même code.

# Plan de test intégral.

Contexte : MediaTek86

Application : MediatekDocuments (application de bureau C#) exploitant rest\_mediatekdocuments (API REST en PHP)

## Plan de tests

### Tests unitaires sur les classes du package model

But du test	Action de contrôle	Résultat attendu	Bilan
Contrôler la méthode ToString() de la classe Categorie pour voir si elle retourne le libelle.	Test unitaire lancé après avoir créé un objet de type Genre avec libelle contenant : "Horreur"	"Horreur"	OK
Contrôler le contenu des propriétés de la classe Abonnement pour vérifier que les propriétés correspondent aux paramètres de l'instance de l'objet	Test unitaire lancé après avoir créé un objet de type Abonnement.	Les propriétés de l'objet contiennent bien les valeurs passées en paramètre.	OK
Contrôler la méthode ToString() de la classe Suivi pour voir si elle retourne le bon libelle.	Test unitaire lancé après avoir créé une liste de suivi et une liste de libelle ordonnés pour correspondre aux indices des Suivi dans la liste des suivi (et dans la base de données).	Chaque libelle correspond bien au libellé attendu dans la liste des libellés.	OK
Contrôler le contenu des propriétés de la classe Utilisateur pour voir s'ils correspondent bien aux valeurs avec lesquelles a été créé l'objet.	Test unitaire lancé après avoir créé un objet Utilisateur avec comme Id_service : 1, Prénom : John ; Nom : Doe ; Pwd_hash : sha512PBKDF2Base64Repr==	Chaque propriété contient la valeur attendue.	OK
Contrôler le contenu des propriétés de la classe Revue pour voir s'ils correspondent bien aux valeurs avec lesquelles a été créé l'objet.	Test unitaire lancé après avoir créé un objet Revue.	Chaque propriété contient la valeur attendue.	OK

### Tests fonctionnels avec Reqnroll sur les recherches (onglet livre)

But du test	Action de contrôle	Résultat attendu	Bilan
Contrôler la fonctionnalité de recherche de livre par numéro de document.	On ouvre une instance de la fenêtre principale puis on récupère la TextBox qui doit contenir le numéro de livre et on fixe sa propriété Text à « 00017 ».	On doit apercevoir un seul résultat dans la DataGridView : le livre de titre « Catastrophe au Brésil »	OK
Contrôler le fonctionnement de la recherche de livre par titre ou extrait de titre.	On instancie la fenêtre principale puis on récupère le TextBox qui doit contenir le titre. On lui affecte le motif suivant : « Cata »	On doit apercevoir un seul résultat dans la DataGridView : le livre de titre « Catastrophe au Brésil »	OK
Contrôler le fonctionnement de la recherche de livre par genre	On instancie la fenêtre, on récupère le ComboBox des genres, on sélectionne le genre « science	On observe l'unique livre nommé La Planète des singes dans la	OK

1

	fiction »	DataGridView	
Contrôler le fonctionnement de la recherche de livre par rayon	On instancie la fenêtre, on récupère le ComboBox qui représente les rayons, on sélectionne le type de rayon "Jeunesse BD".	On observe le seul livre présent dans la DGV nommé "Sacré Père Noël"	OK
Contrôler le fonctionnement de la recherche de livre par public	On instancie la fenêtre, on récupère le ComboBox des types de public, on sélectionne le type « Jeunesse »	On observe le seul livre présent dans la DGV nommé "Sacré Père Noël"	OK

### Tests fonctionnels dans Postman (fonctionnalités de l'API)

But du test	Action de contrôle	Résultat attendu	Bilan
Contrôler le fonctionnement de la récupération de données.	On envoie une requête HTTP GET ayant comme paramètre de table « utilisateur » et le champs {« login »: « root »}	La réponse contient toutes les informations de l'utilisateur concerné.	OK
Contrôler le fonctionnement de l'insertion de données	On envoie une requête HTTP POST ayant comme paramètre de nom de table « insert_commande » et les champs suivant : { "dateCommande": "2025-01-01", "montant": 12.99, "idSuivi": 1, "idLivreDvd": "00017", "nbExemplaire": 2 }	La nouvelle commande est bien ajoutée au niveau des deux tables concernées.	OK
Contrôler le fonctionnement de la mise à jour de données	On envoie une requête HTTP PUT ayant comme paramètre de table « commande » et l'id « 00016 » puis le/s champ/s qu'on souhaite mettre à jour : { "montant": 6.99 }	Le nouveau montant de la commande d'id « 00016 » est bien passé à 6.99.	OK
Contrôler le fonctionnement de la suppression de données.	On envoie une requête HTTP DELETE avec comme paramètres get la table « commandedocument » et « {« idLivreDvd »: « 00017 »}	Toutes les commandes dont l'idLivreDvd correspond ont été supprimées.	OK

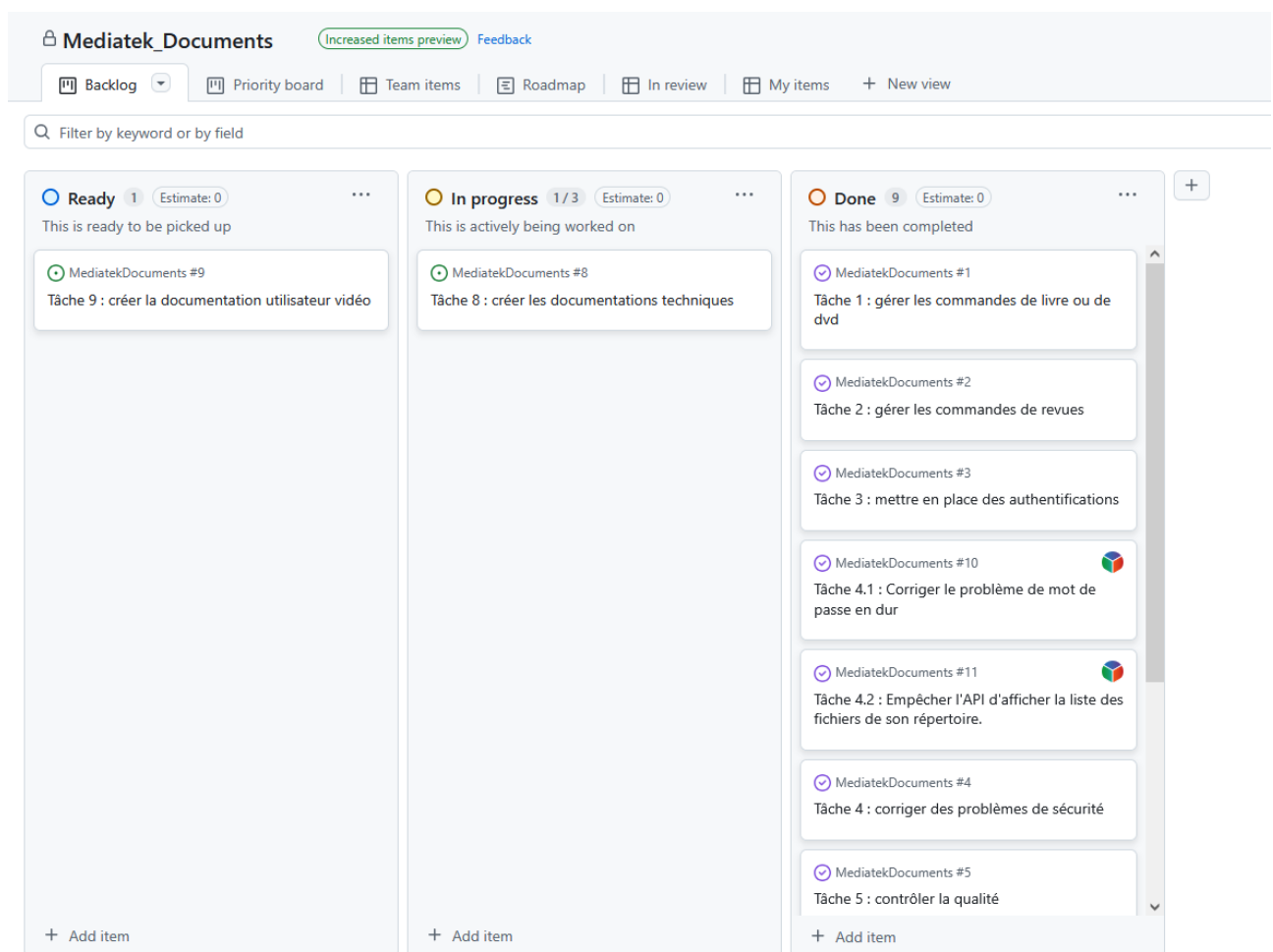
## Tâche n°8 : générer les documentations techniques.

Contrôler, dans chaque application, que les commentaires normalisés sont bien tous ajoutés et corrects. Générer la documentation technique de l'application C#. Générer la documentation technique de l'API REST puis transférer les documentations dans les dépôts.

Durée estimée : 1h

Durée effective : 2h55

## **Aperçu du kanban de la tâche en cours.**



## **Documentation technique de l'application C#.**

La documentation de l'application C# a été générée à l'aide du logiciel SandCastle qui permet, à partir d'un fichier XML représentant les commentaires normalisés, de générer la documentation technique au format HTML, pratique pour être présenté dans un navigateur.

Exemple de commentaire normalisé pour la méthode `CryptoTools.VerifyPassword` :

```
/// <summary>
/// Méthode de vérification d'identité entre un mot de passe en claire et un
hash de référence
/// </summary>
/// <param name="plaintext">Mot de passe en clair</param>
/// <param name="hashReference">HASH de référence (base64)</param>
/// <returns>true si le mot de passe en clair correspond au hash de référence,
false sinon</returns>
```

Le sommaire « summary » permet de décrire l'utilité de la méthode.

Les paramètres « param name =... » permettent de préciser le rôle et le type des paramètres.

Enfin « returns » donne des précision sur le type d'objet renvoyé par la méthode.

Après génération de la documentation, on obtient le résultat suivant :

## CryptoTools.VerifyPassword Méthode

Méthode de vérification d'identité entre un mot de passe en claire et un hash de référence

### ▼ Definition

Espace de nom: [MediaTekDocuments.utils](#)

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

C#

 Copie

```
public static bool VerifyPassword(
    string plaintext,
    string hashReference
)
```

### Parameters

**plaintext** [String](#)

Mot de passe en clair

**hashReference** [String](#)

HASH de référence (base64)

### Valeur de retour

[Boolean](#)

true si le mot de passe en clair correspond au hash de référence, false sinon

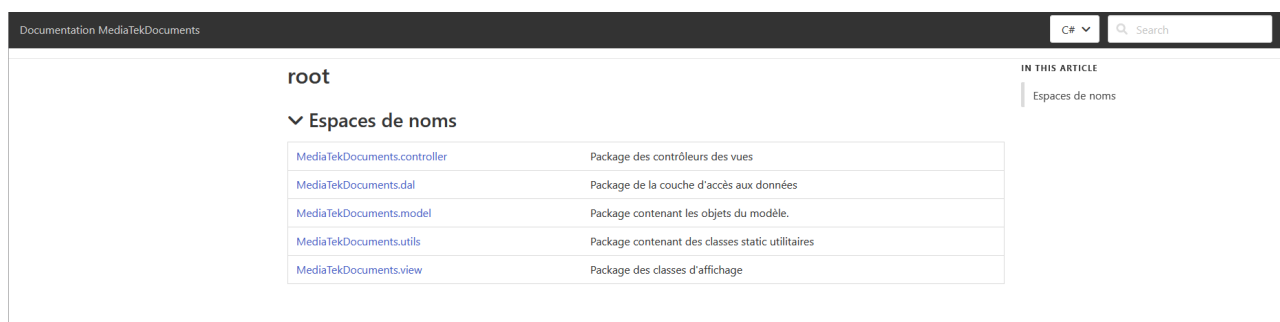
### ▼ Voir aussi

#### Référence

[CryptoTools Classe](#)

[MediaTekDocuments.utils Espace de nom](#)

Vue d'ensemble des espaces de noms (ou packages) de l'application :



## Documentation technique pour l'API PHP.

La documentation technique de l'API a été générée à l'aide de l'utilitaire phpDocumentor. Ce programme est capable de parser les commentaires normalisés présents dans le code de l'application, puis de générer une documentation au format HTML, présentable dans un navigateur.

Exemple de commentaire normalisé concernant une méthode de l'API :

```
/**
 * Insère un nouvel abonnement pour une revue
 * @param array|null $champs
 * @return int|null
 */
```

Aperçu du rendu correspondant dans la documentation HTML :

insérerAbonnementRevue()

MyAccessBDD.php : 354

*Insère un nouvel abonnement pour une revue*

```
private insérerAbonnementRevue(array<string|int, mixed>|null $champs) : int|
null
```

Parameters

\$champs : array<string|int, mixed>|null

Return values

int|null

Aperçu global de la documentation :

## Packages

### Application

## Reports

### Deprecated

### Errors

### Markers

## Indices

### Files

### RepositoryInterface

### StoreInterface

## Classes

### AccessBDD

Classe qui sollicite ConnexionBDD pour l'accès à la BDD MySQL Elle contient les méthodes appelées par Controle et les méthodes abstraites que MyAccessBDD doit redéfinir pour construire les requêtes

### Connexion

Classe de connexion à la BDD MySQL (singleton) et d'exécution des requêtes en retournant : - pour les requêtes LID : contenu du curseur au format tableau associatif - pour les requêtes LMD : nbre d'enregistrements impactés Dans tous les cs, 'null' est renvoyé si la requête échpue.

### Controle

Contrôleur : reçoit et traite les demandes du point d'entrée

### MyAccessBDD

Classe de construction des requêtes SQL hérite de AccessBDD qui contient les requêtes de base Pour ajouter une requête : - créer la fonction qui crée une requête (prendre modèle sur les fonctions existantes qui ne commencent pas par 'traitement') - ajouter un 'case' dans un des switch des fonctions redéfinies - appeler la nouvelle fonction dans ce 'case'

### Url

Singleton car la récupération des données ne peut se faire qu'une fois Permet de gérer le contenu de l'URL qui sollicite l'API

Cette documentation est accessible publiquement via le lien suivant :

[http://mediatek-documents.atwebpages.com/rest\\_mEDIATEKdocuments/Documentation/](http://mediatek-documents.atwebpages.com/rest_mEDIATEKdocuments/Documentation/)

# Tâche n°9 : Déployer le projet

## Mettre en ligne l'API :

- Renforcer la sécurité de l'API : changer les username/password de l'accès à l'API (les informations de connexions à la BDD, mises dans le fichier '.env', devront être modifiées en tenant compte des informations prises chez l'hébergeur).
- Déployer l'API et la BDD.
- Tester l'API avec Postman

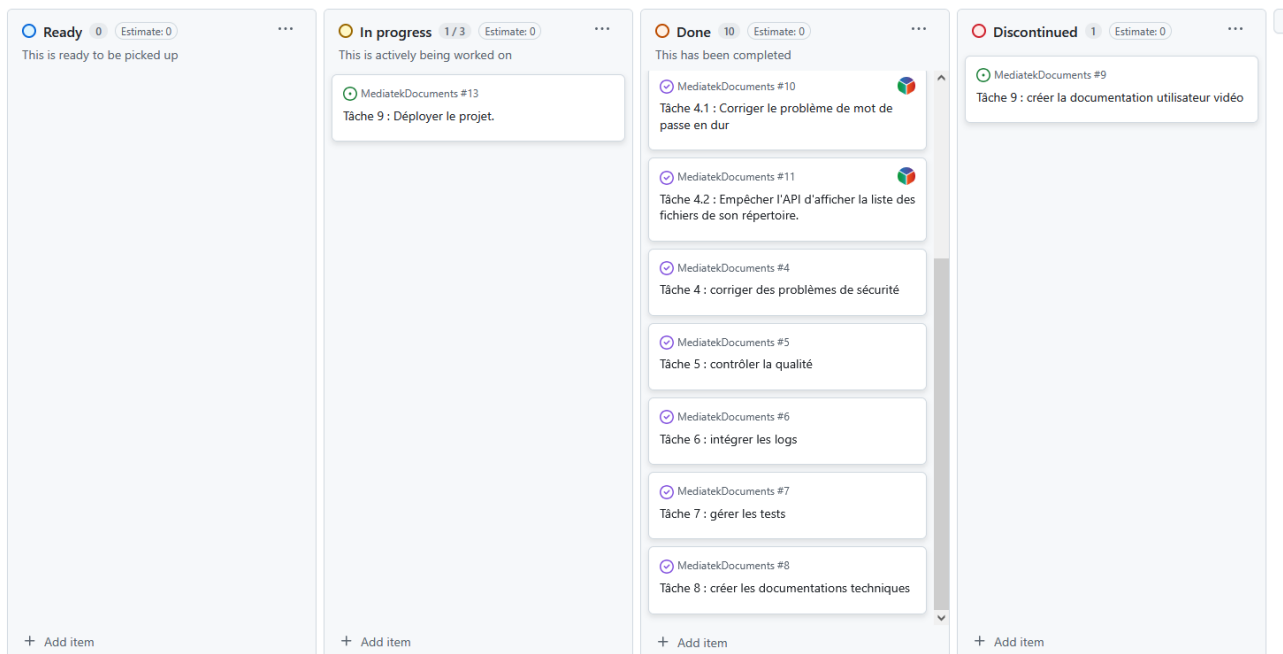
## Créer un installeur pour l'application C# :

Modifier le code pour l'accès à l'API distante. Créer l'installeur, tester son installation et l'utilisation de l'application installée. L'envoyer vers le dépôt.

Durée estimée : 3h

Durée effective : 5h

## Aperçu du kanban de la tâche en cours.



## Déploiement de l'API et de la base de données.

Le déploiement de l'application nécessite la mise en place d'une API accessible depuis l'internet. Pour ce faire, on a choisi l'hébergeur gratuit AwardSpace (<https://www.awardspace.com/>) qui permet l'utilisation de tous les verbes HTTP (POST, PUT, DELETE etc..) et des triggers MySQL sans restriction.

FTP :

Un serveur FTP sécurisé (SFTP) est mis à disposition par l'hébergeur afin de téléverser les fichiers de l'API sur le serveur web. On utilisera FileZilla pour y accéder.

MySQL :

Une interface de création de base de données MySQL est disponible sur le panel de l'hébergeur. Après création, on peut administrer la base via phpmyadmin.

Sous-Domaines :



L'hébergeur met à disposition des sous-domaines gratuits, malheureusement l'utilisation d'un certificat SSL est une option payante.

Serveur-Web :

Le serveur mis à disposition est Apache2 avec PHP 8.

Après téléversement des fichiers via FTP, quelques changements dans le code et dans le fichier d'environnement sont nécessaires. En effet, il faut modifier les identifiants d'authentification à l'API avec un mot de passe plus fort et d'autre part, remplacer les identifiants de connexion à la base de données par ceux fournis par l'hébergeur.

Comme mentionné dans le guide de déploiement, le serveur web ayant du mal à interpréter la fonction *filter\_input* dans la méthode *recupererMethodeHTTP*, on la remplacera par la fonction *htmlspecialchars* car il n'est vraiment pas recommandé de faire l'impasse sur le filtrage dans un environnement de production.

```
public function recupMethodeHTTP() : string{  
    return htmlspecialchars($_SERVER['REQUEST_METHOD']);  
}
```

Par ailleurs, il est nécessaire de modifier le fichier .htaccess en lui rajoutant les lignes suivantes :

```
RewriteCond %{HTTP:Authorization} ^(.+)$  
RewriteRule ^(.*)$ - [E=HTTP_AUTHORIZATION:%1]
```

Sans cette modification, l'authentification à l'API via l'application C# ou Postman n'est pas fonctionnelle.

## Déploiement du client C#

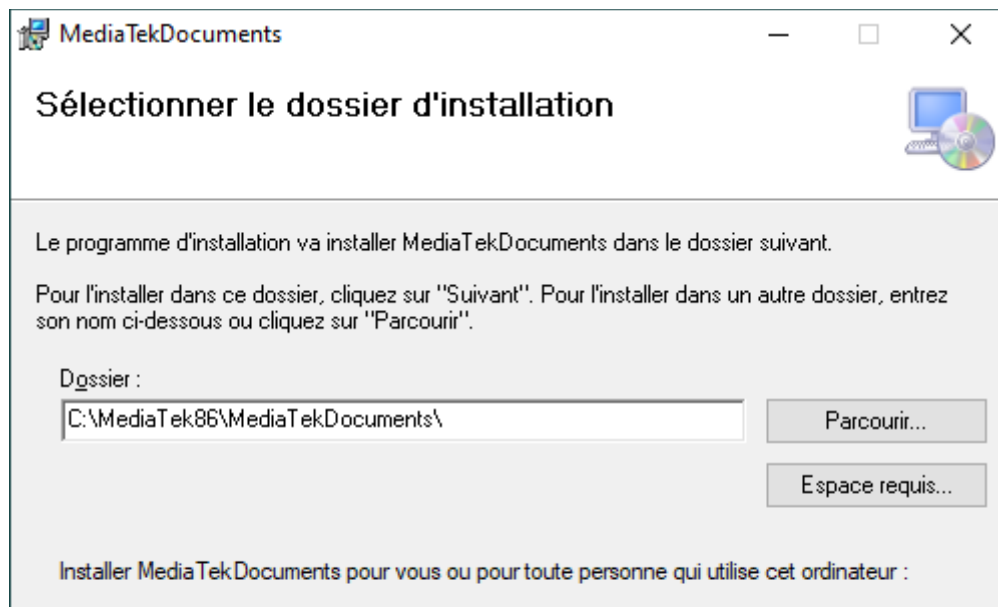
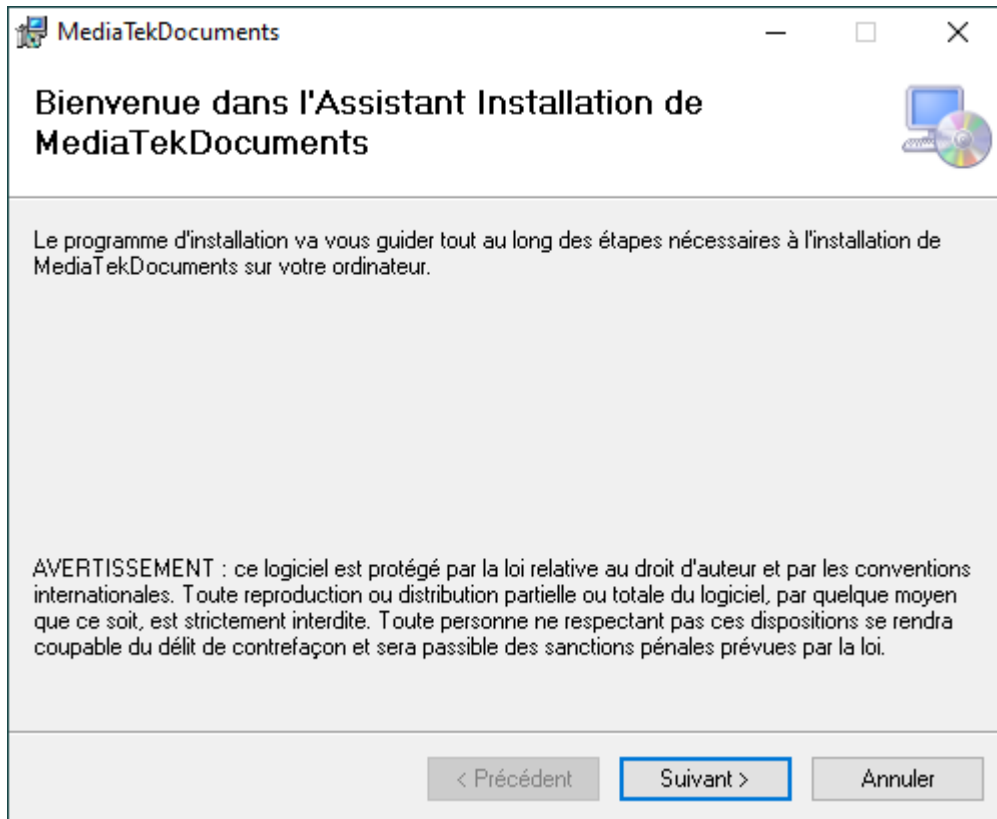
Une fois l'accès distant à l'API opérationnel et testé via Postman, on peut modifier la configuration du projet.

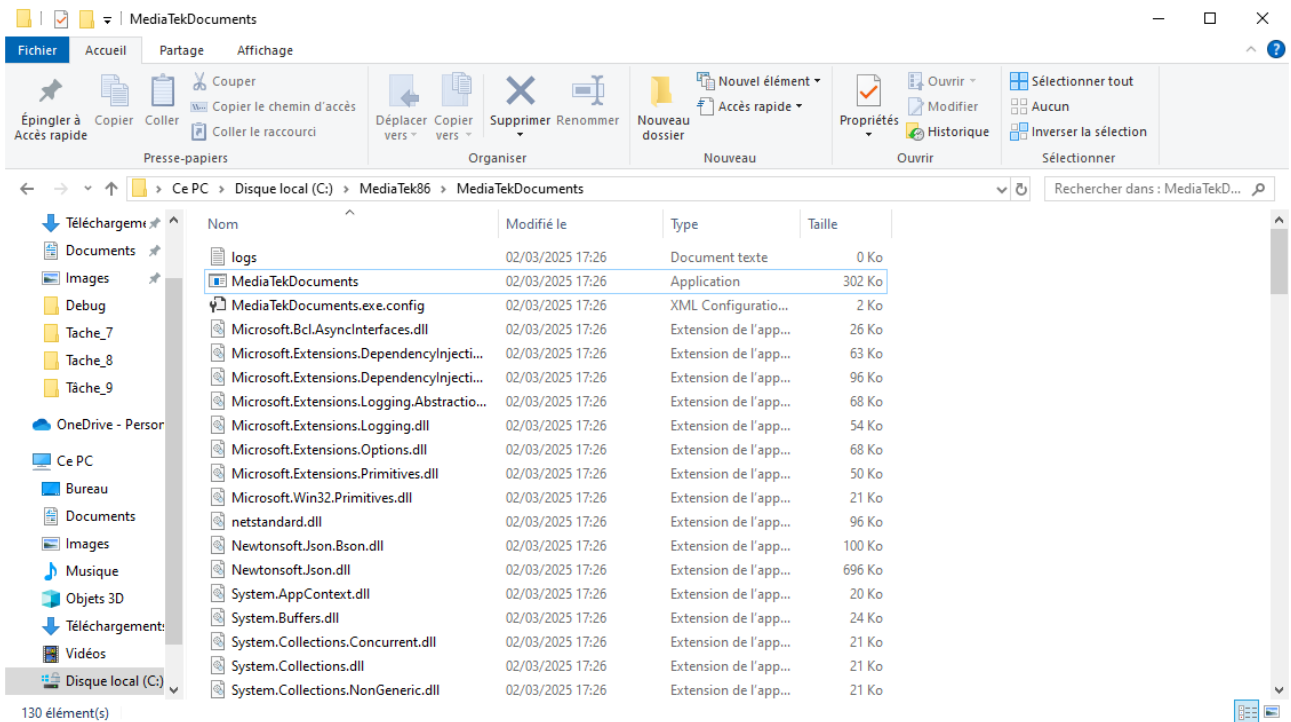
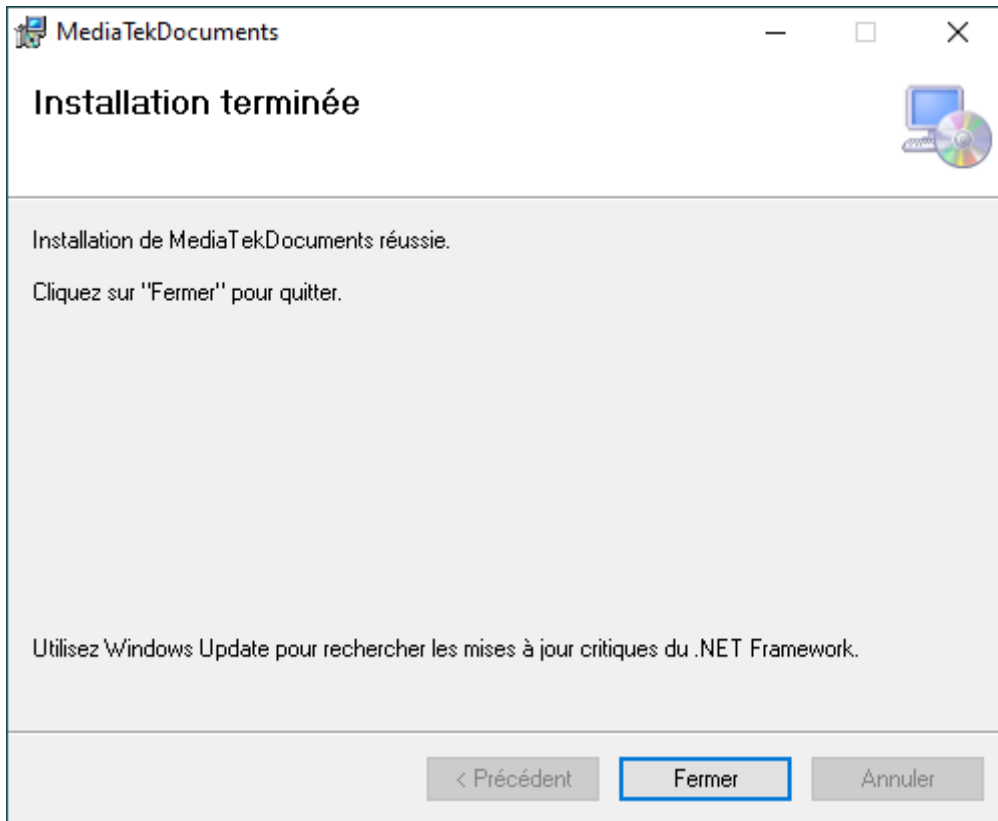
Tout d'abord, on change la configuration actuelle du projet en mode « Release » dans Visual Studio. Ensuite, on installe l'extension « Visual Studio Installer Project », on crée un nouveau projet de type « Setup » à l'intérieur de la solution, on spécifie le chemin vers l'exécutable du programme, les dépendances sont ajoutées automatiquement, il reste cependant à inclure le fichier de journalisation « logs.txt ».

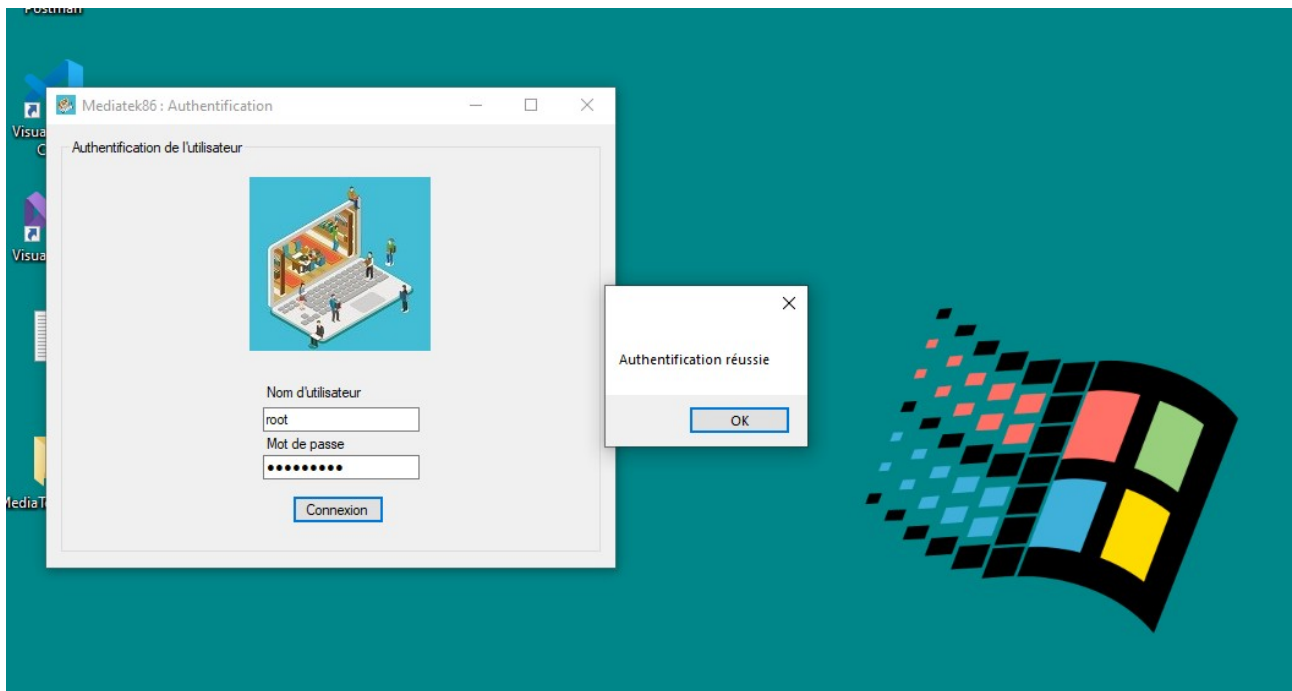
Note : les permissions du fichier de logs étant difficilement paramétrables sans script externe post installation, on choisira par simplicité de placer l'application en dehors de Program Files (x86). De cette façon, le programme aura les droits d'écriture sur ce fichier.

Le chemin par défaut de l'application est C:\MediaTek86\

## **Aperçu des étapes d'installation du logiciel.**

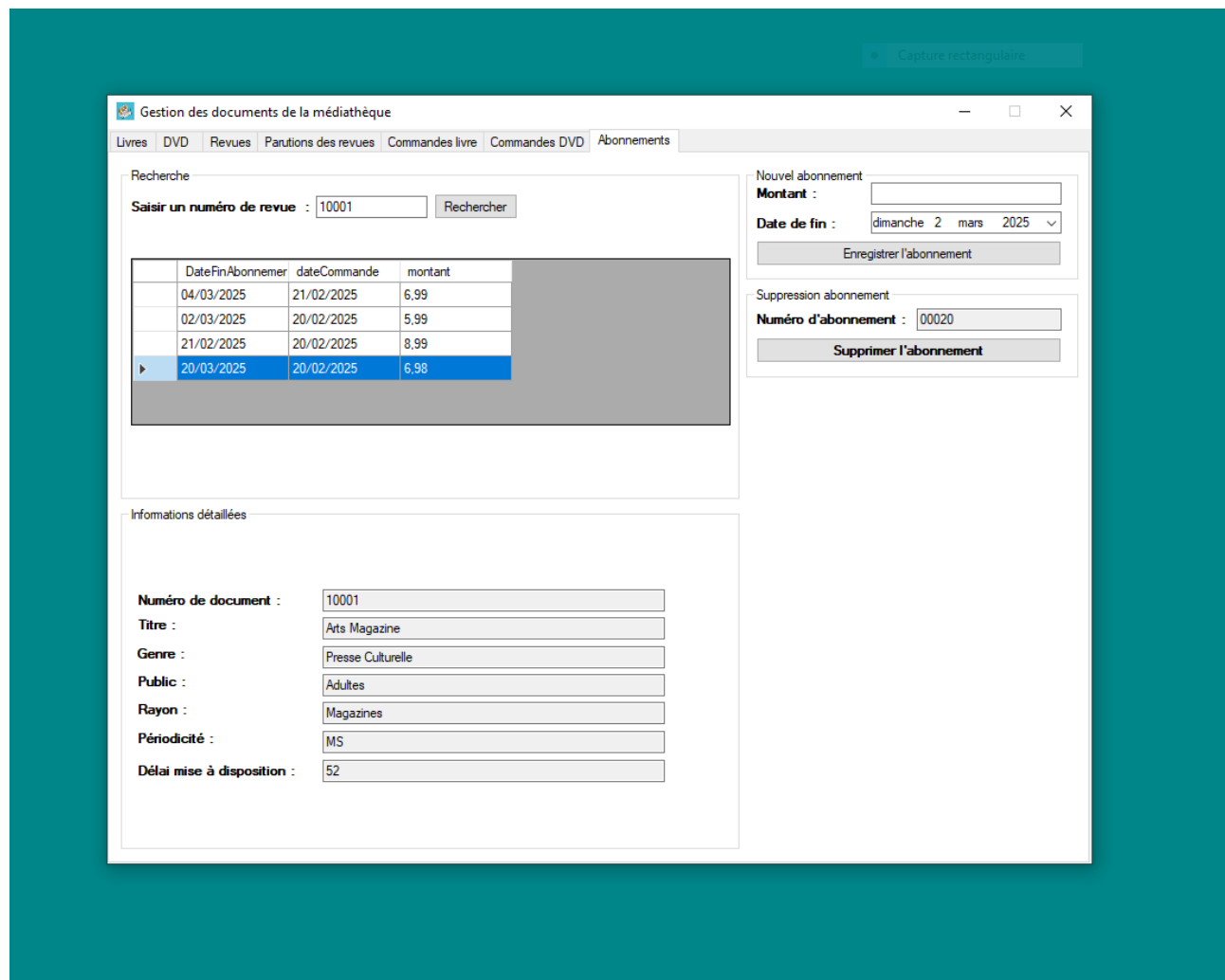






Nom d'utilisateur : root

Mot de passe : P@\$\$word1



L'installateur et l'application ont été testés sur deux postes, l'un disposant de Windows 10 Professionnel version 22H2 et l'autre de Windows 10 Famille version 22H2.

## **Tâche n°10 : automatiser la sauvegarde de base de données.**

Une sauvegarde journalière automatisée doit être programmée pour la BDD.

La restauration pourra se faire manuellement, en exécutant le script de sauvegarde.

Durée estimée : 1h

### **Limitations techniques liées aux paramètres de l'hébergeur.**

L'automatisation de la sauvegarde n'est malheureusement pas possible avec l'offre actuelle.

D'une part, la création d'une tâche CRON implique de sortir de l'offre gratuite, d'autre part, on aurait pu externaliser cette tâche à l'aide du site <https://cron-job.org> pour qu'il appelle automatiquement un script de sauvegarde PHP basé sur mysqldump, néanmoins, les privilèges nécessaires à l'utilisation de mysqldump ne sont pas assignés à l'utilisateur de base de données, et il n'est pas possible de s'attribuer ce privilège avec « GRANT process [...] ».

Trace de l'erreur généré par le script de sauvegarde :

```
"mysqldump: Error: 'Access denied; you need (at least one of) the PROCESS privilege(s) for this operation' when trying to dump tablespaces"
```


Script de sauvegarde PHP :

```
<?php  
require '../rest_mEDIATEKdocuments/vendor/autoload.php';
```

```
use Dotenv\Dotenv;
$dotenv = Dotenv::createImmutable(__DIR__);
$dotenv->load();
$database = $_ENV['BDD_BD'];
$user = $_ENV['BDD_LOGIN'];
$pass = $_ENV['BDD_PWD'];
$host = $_ENV['BDD_SERVER'];
$dir = dirname(__FILE__) . '/dump.sql';
exec("mysqldump --user={$user} --password={$pass} --host={$host} {$database} --
result-file={$dir} 2>&1", $output);
var_dump($output);
```

Le seul moyen de sauvegarder la base de données est de passer par le panel de l'hébergeur et de cliquer sur la fonctionnalité de sauvegarde.

### Database SQL Export Download



Download in SQL Format

Exports the database to a file in SQL format, and begins downloading it to your computer.

- Saves the database structure and data to your computer
- Provides you with a permanent offline copy of your database
- Can be imported on any MySQL-compatible server

Il serait possible d'automatiser l'action de sauvegarde avec une tâche planifiée (ou une tâche CRON locale) et un programme python utilisant la bibliothèque request, il faudrait simuler l'authentification au panel, puis le clic sur le bouton de sauvegarde par exemple.

Néanmoins, cette méthode nécessite que la machine cliente soit démarrée en permanence pour que l'automatisation soit fonctionnelle, ce qui n'est pas le but recherché dans l'automatisation d'une sauvegarde.

Dans le cadre de ce projet, nous n'automatiserons donc pas la sauvegarde journalière de la base de données.

## Bilan Final.

A la fin de la réalisation de ce projet, on obtient une application de gestion et une API pour gérer les commandes et abonnements concernant les documents de la médiathèque. L'intérêt majeur de cette application est qu'elle peut être utilisée de façon concurrente par plusieurs employés de la médiathèque car tout est centralisé dans une seule base de données grâce à l'API, il suffit d'installer l'application sur les différents postes informatiques. Les technologies utilisées lors de cet atelier sont variées, ce qui permet d'élargir les connaissances et de voir un exemple concret du

fonctionnement d'un système d'information appliqué à un secteur spécifique. Certains problèmes ont tout de même été rencontrés lors de la création du projet. Tout d'abord, le descriptif des tâches était parfois un peu succinct par rapport aux ateliers précédents, ce qui rendait la compréhension de l'objectif à atteindre difficile, et a demandé du temps supplémentaire pour comprendre le fonctionnement attendu. D'autre part, la gestion des identifiants des tables à l'aide d'un VARCHAR a obligé l'utilisation d'un trigger là où un simple AUTO INCREMENT aurait pu suffire. Le déploiement de l'API a également été contraint par les paramètres de l'offre gratuite d'hébergement. En effet, il n'a pas été possible d'automatiser la sauvegarde journalière de la base de données pour les raisons expliquées dans la partie déploiement de ce document. Il est aussi impossible de demander l'installation d'un certificat SSL sans sortir de l'offre gratuite, ce qui représente tout de même un risque de sécurité lors de l'authentification à l'API, les données d'authentification transitent en clair sur le réseau, et peuvent donc être dérobées par une personne en position d'homme du milieu. L'utilisation d'un VPN peut atténuer ce problème de sécurité.